Improving Induction Motor Fault Classification Accuracy Through Enhanced Multimodal Preprocessing, Artificial Image Synthesis, Deep Learning and Load-Adaptive Graph-Based Methods

A thesis submitted to Cardiff University in the candidature for the degree of Doctor of Philosophy (Engineering)

Name: Shahd Ziad Hejazi

ID #: 1963733

Supervisors: Dr. Michael Packianather

Prof. Ying Liu

School of Engineering

Cardiff University

Wales, United Kingdom

November 2024

Abstract

This thesis aims to improve the accuracy of fault classification in Induction Motor (IM) bearings by developing and applying advanced Artificial Intelligence (AI) and Machine Learning (ML) techniques for condition monitoring data. The proposed framework utilises several approaches, namely, Multimodal Data Preprocessing, Artificial Thermal Image Creation, Customised Radial Load Assessment, Multimodal Systems Decision Fusion, and Graph Convolutional Networks (GCN) on Tabular Datasets to achieve better classification accuracies over existing methods.

This study's first significant contribution is the proposed novel approach in the preprocessing of multimodal condition monitoring data for classifying induction motor faults that employs Convolutional Neural Networks (CNNs), such as Residual Network-18 (ResNet-18) and SqueezeNet, to fuse vibration signals and thermal images. This approach enhances fault classification accuracy by 14.81% and proves exceptionally effective in scenarios with compromised image quality. Further refinement using Gramian Angular Field (GAF) processing enhances the detection of subtle fault indicators, achieving better accuracy than Continuous Wavelet Transform (CWT).

Secondly, this thesis explores the creation of high-quality artificial thermal images using Wasserstein GAN with Gradient Penalty (WGAN-GP) and its conditional variant, conditional Wasserstein GAN with Gradient Penalty (cWGAN-GP), to address the scarcity of thermal imaging data. The artificial thermal images replicate complex thermal patterns of IMs under various fault conditions with remarkable accuracy, as evidenced by the improved Maximum Mean Discrepancy (MMD) scores and a 40.00% reduction in training times. The high fidelity of these artificially generated images, validated against real images, underscores their practical use in fault classification.

Thirdly, the Customised Load Adaptive Framework (CLAF) introduces a novel approach to incorporating load variations into fault classification. Through a two-phase process involving ANOVA and optimal CWT, load-dependent fault subclasses—Mild, Moderate, Severe, and Normal (fault-free) or Healthy—are identified. The CLAF achieved an accuracy of 96.30% ± 0.50% in 18.155 s during five-fold cross-validation using a Wide

Neural Network (WNN), demonstrating its ability to detect subtle fault variations across different Load Factors (LFs).

Fourthly, building upon the CLAF's load-dependent fault subclass structure, the research proposed two key methodologies for enhancing load-specific condition monitoring accuracy while optimising training time relative to complexity using the MFPT bearing dataset namely, the Load-Dependent Multimodal Vibration Signal Enhancement and Fusion (LD-MVSEF) method, and the Hybrid Graph-CNN Decision Fusion (HG-CDF) method. The LD-MVSEF employs a multimodal approach across multiple channels, with different signal encoding techniques achieving a fault classification accuracy of 99.04% ± 0.22% over five runs in 18 min 30 s. It performed particularly well in the Moderate class, achieving 99.15% ± 0.89% testing accuracy, and scored 97.20% ± 1.75% in the Mild class.

The proposed HG-CDF combines the structural strengths of Graph Convolutional Networks (GCNs) with the pattern-detection capabilities of 1D-Convolutional Neural Networks (1D-CNNs) for CLAF load-dependent fault subclass classification. The study began by optimising the GCN through Taguchi experiments, converting tabular data into graph structures using the k-Nearest Neighbours method and achieving a mean accuracy of 89.01% ± 1.25 across nine configurations. HG-CDF further improved performance, reaching an overall accuracy of 99.19% in just 3 minutes and 28 seconds, surpassing LD-MVSEF in the Mild class with 98.92% accuracy while also providing a faster and more efficient solution.

The methodologies proposed in this research significantly enhance the IM fault classification task, improve the decision-making process, and offer scalable solutions adaptable to other domains.

Acknowledgements

I am sincerely grateful to my first supervisor, Dr Michael Packinather, for his invaluable guidance, unwavering support, and insightful feedback throughout my research. Dr Packinather's expertise and dedication have been instrumental in shaping the outcome of this thesis. I am genuinely grateful for his mentorship and encouragement.

I am also deeply indebted to my second supervisor, Prof. Ying Liu, from Cardiff University. Prof. Liu's expertise in the field and constructive criticism have been invaluable in refining the direction of my research. Their guidance and encouragement have played a significant role in completing this work.

I express my sincere appreciation to the Saudi Arabian Cultural Bureau for their financial support and sponsorship throughout my doctoral studies, which have enabled me to pursue my academic goals and fostered valuable opportunities for growth and learning. Additionally, I extend my gratitude to Cardiff University for their staunch support.

Furthermore, I sincerely thank my family for their unwavering support, love, and understanding throughout this journey. Their constant encouragement and belief in my abilities have strengthened and motivated me. I am genuinely grateful for their presence in my life.

I also thank my husband for his continuous support, patience, and encouragement. His steady belief in me and willingness to listen and provide insightful advice have been invaluable. I am grateful for his love and companionship, which have made this challenging journey more manageable.

Lastly, a special and heartfelt thanks goes to my daughter, Ghalia, who came into this world during the final stages of this research. Her arrival marked the most beautiful chapter of my life and gave deeper meaning to the completion of this journey.

Table of Content

List of Figures

List of Tables

List of Abbreviations

Adaptive Channel Mixing (ACM)

Acoustic Emission (AE)

Analysis of Variance (ANOVA)

Artificial Intelligence (AI)

Adaptive Moment Estimation (Adam)

Adaptive Synthetic Sampling Technique (ADASYN)

Artificial Neural Network (ANN)

Asynchronous Advantage Actor-Critic (A3C)

Autoregressive (AR)

AutoRegressive Integrated Moving Average (ARIMA)

AutoRegressive Integrated Moving Average with exogenous variables (ARIMAX)

Auxiliary Classifier GAN (ACNN)

Back Propagation Neural Network (BPNN)

Batch Normalisation (BN)

Case Western Reserve University (CWRU)

Centre Particle Swarm Optimisation (CPSO)

Centre Frequency (CF)

Conditional Generative Adversarial Networks (cGANs)

Conditional Wasserstein Generative Adversarial Network with Gradient Penalty (cWGAN-GP)

Contact-based Non-Invasive Inspection (CNI)

Continuous Wavelet Transform (CWT)

Convolutional Neural Network (CNN)

Cubic Support Vector Machine (CubicSVM)

Customised Load Adaptive Framework (CLAF)

Data-driven Fault Diagnosis (DFD)

Deep Belief Networks (DBNs)

Deep Boltzmann Machines (DBMs)

Deep Convolutional Generative Adversarial Network (DCGAN)

Deep Learning (DL)

Deep Neural Networks (DNNs)

Design of Experiments (DOE)

Decision Tree (DT)

Discrete Wavelet Transform (DWT)

Discriminator (D)

Earth Mover's Distance (EMD)

Electrocardiogram (ECG)

Electromyographic (EMG)

Electronic Design Automation (EDA)

Ensemble AdaBoost Decision Tree (EADT)

Fast Fourier Transform (FFT)

False Negative (FN)

False Positive (FP)

Feature Fusion Convolutional Neural Network-Support Vector Machines (FFCNN-SVM)

Fréchet Inception Distance (FID)

Gated Recurrent Unit (GRU)

Generative Adversarial Network (GAN)

Generator (G)

Gradient Angular Difference Field (GADF)

Gradient Penalty (GP)

Gramian Angular Field (GAF)

Gramian Angular Summation Field (GASF)

Graph Convolutional Network (GCN)

Graph Neural Networks (GNNs)

Hybrid Graph-CNN Decision Fusion (HG-CDF)

Infrared (IR)

Infrared Thermography (IRT)

Induction Motors (IMs)

Inner Race Defect (IRD)

Inner Race Fault (IRF)

Jensen-Shannon Divergence (JSD)

k-Nearest Neighbours (kNN)

k-Nearest Neighbour Graphs (k-NNG)

Knowledge Graphs (KG)

L2-Support Vector Machine (L2-SVM)

Load-Dependent Multimodal Vibration Signal Enhancement and Fusion (LD-MVSEF)

Load Factor (LF)

Long Short-Term Memory (LSTM)

Machine Learning (ML)

Magnetic Anomaly Detection (MAD)

Machinery Failure Prevention Technology (MFPT)

Markov Decision Processes (POMDPs)

Maximum Mean Discrepancy (MMD)

Mean Square Deviation (MSD)

Monte Carlo Tree Search (MCTS)

Multi-Level Features Fusion Network (MLFNet)

Multi-Scale Neural Transformation Graph (MNT-G)

Multilayer Perceptron (MLP)

Multimodal Two-stream GNN Framework for Efficient Point Cloud and Skeleton Data Alignment (MTGEA)

Neural Networks (NNs)

Noncommunicable Diseases (NCDs)

Non-Contact-Based Non-Invasive Inspection (NCNI)

Non-Invasive Inspection (NII)

One-Dimensional (1D)

One-Dimensional Convolutional Neural Network (1D-CNN)

One-Dimensional Ternary Patterns (1D-TP)

Outer Race Defect (ORD)

Outer Race Fault (ORF)

Power Factor (PF)

Principal Component Analysis (PCA)

Proximal Policy Optimisation (PPO)

Radio Frequency (RF)

Random Seed (S)

Rectified Linear Unit (ReLU)

Recurrent Neural Networks (RNNs)

Reinforcement Learning (RL)

Residual Network (ResNet)

Residual Network-18 (ResNet-18)

Resource Description Framework (RDF)

Root Mean Square (RMS)

Root Mean Square Propagation (RMSProp)

Root-Mean-Square Frequency (RMSF)

Signal-to-Noise and Distortion Ratio (SINAD)

Signal-to-Noise Ratio (S/N)

Spatial-Temporal Graph Convolutional Network (ST-GCN)

Squeeze Ratio (SR)

Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent with Momentum (SGDM)

Stationary Wavelet Transform (SWT)

State-Action-Reward-State-Action (SARSA)

Support Vector Machine (SVM)

Super Resolution - Graph Neural Network (SR-GNN)

Synthetic Minority Over-sampling Technique for Regression with Gaussian Noise (SMOGN)

Synthetic Minority Oversampling Technique (SMOTE)

Synchronous Reluctance Machines (SynRMs)

Short-Time Fourier Transform (STFT)

Temporal Graph Convolutional Network (TGCN)

Time and Frequency Domain (TFD)

Total Harmonic Distortion (THD)

Traditional Statistical Features (TSFs)

True Negative (TN)

Two-Dimensional (2D)

Two-Dimensional Convolutional Neural Network (2D-CNN)

True Positive (TP)

t-Distributed Stochastic Neighbour Embedding (t-SNE)

Visual Geometry Group Network (VGGNet)

Vibration Signal Analysis (VSA)

Wasserstein Generative Adversarial Network (WGAN)

Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP)

Wavelet Packet Transform (WPT)

Wavelet Scattering Transform (WST)

Wavelet Singular Entropy (WSE)

Wavelet Transform (WT)

Wide Neural Network (WNN)

List of Symbols

a:	Scaling factor in the Wavelet Transform (WT) that stretches or compresses the wavelet function.
a_p:	Autoregressive coefficients of the AR model.
A:	Adjacency matrix of the graph, where an element Aij=1 indicates the presence of an edge between node i and node j.
A_i:	Amplitude of the i-th harmonic.
A_I :	Amplitude of the fundamental frequency.
$AR(P)$:	Autoregressive model of order P, where P indicates the number of past values the model considers.
C_g:	Covariance matrix of the feature distribution for generated images, used in the context of the Fréchet Inception Distance (FID).
C_r:	Covariance matrix of the feature distribution for real images, used in the context of the FID.
$\cos\beta$:	Cosine of the load angle, reflecting the angular displacement under load.
$cos(\phi_1 + \phi_1)$:	Cosine of the sum of angles, used in the computation of GASF.
$cos(\phi_1 - \phi_1)$:	Cosine of the difference between angles, used in the computation of GADF.
D:	The discriminator component of the Generative Adversarial Network (GAN) that evaluates data points. In the context of WGAN, referred to as the "critic" instead of the discriminator.
D_{ball}:	Diameter of the Induction Motor (IM) balls.
D_{cage}:	Diameter of the IM cage.
$D(x\|y)$:	Discriminator's output when provided with a real data sample x conditioned on y in GANs.
$\mathbb{E}$:	Average value over a probability distribution, used in Maximum Mean Discrepancy (MMD) to measure the average similarity between samples from real and generated distributions

$e[n]$:	Random noise component at time index n.
F:	Constraint that ensures the function f is Lipschitz continuous, a requirement for the critic in WGAN.
f:	Function applied by the critic to evaluate images for WGAN.
f_{Ball}:	Frequency associated with faults in the IM balls of the bearing.
f_{IRF} :	Frequency of the Inner Race Fault in IM.
f_m:	Rotational frequency of the IM.
f_{ORF}:	Frequency of the Outer Race Fault of the IM.
f_1:	Trainable function used to process features from neighbouring nodes, parameterised by θ_1^I in Graph Neural Networks (GNNs)
f_2:	Trainable function used to update the node feature after aggregation, parameterised by θ_2^I in GNNs
$f(t)$:	Input signal as a function of time t.
G:	The generator component of the GAN creates new data points.
G:	Represents the graph structure in GNNs.
$G(z)\ or\ x'$:	The output of the Generator in a GAN, where z is a noise vector, and $G(z)$ is the synthetic data generated to resemble real data.
$G(z\|y)$:	Generator's output, which is a fake data sample, generated from noise vector z conditioned on y in GNNs.
$k(x, y)$:	Gaussian kernel function applied to sample x from the real distribution and sample y from the generated distribution in MMD.
$k(y, y')$:	Gaussian kernel function applied to samples y and y' from the generated distribution in MMD.
m:	Target value in the nominal-the-best scenario where the goal is to achieve a specific dimension or value. Used in S/N ratio.
N:	Constant used to adjust the scale of the radius in polar coordinates to prevent image distortion over time. N is used as a constant to adjust the scale of the radius r in polar coordinates to prevent distortion in Gramian Angular Field (GAF) images.
N_{ball}:	Number of balls in the IM bearing.

N_i:	Set of neighbouring nodes for node i in GNN.
n:	Length of the time series, which also defines the dimensions of the resulting GAF matrix, $n \times n$.
P_g:	Probability distribution of data generated by G in GNNs.
P_r:	Probability distribution of real images in GNNs.
P_z:	Prior distribution from which the noise vector z is sampled.
p:	The order of the Autoregressive (AR) model represents the number of terms used in the model.
$p = (x\|y)$:	The conditional probability distribution of x given y, is not considered in GANs.
P_{signal}:	The power of the signal, used to calculate the Signal-to-Noise Ratio (S/N).
P_{noise}:	The power of the background noise used in the S/N calculation.
$P_{distortion}$:	Power of harmonic distortion.
$P_z(z)$:	The distribution from which the noise vector z is sampled. Used as the input for generating data in GANs.
r:	Radius in the polar coordinates, representing the timestamp t_i in Gramian Angular Field (GAF)
s:	Scaling parameter, which adjusts the width of the wavelet function, influencing its frequency characteristics.
Tr:	Trace of a matrix, used here to calculate the sum of the diagonal elements of a matrix.
t:	Time variable, representing the point in time at which the signal is analysed.
t_i:	Timestamp corresponding to the i-th element of the time series.
V:	Set of nodes in the graph in GNN.
$V(D, G)$:	The value function for the GAN is used in the optimisation process.
x:	Real data points from the training set.

X:	Set of initial node features for all nodes in the graph, $X=\{x_i^0 \mid i \in V\}$ in GNN.
X:	The original time series data, consisting of elements x_1, x_2, ..., x_n.
x_{max}:	Maximum value in the time series data.
x_{min}:	Minimum value in the time series data.
x_i^0:	Initial feature vector associated with node i in GNN.
x_i^l:	Feature vector of node i at the l-th layer of the GNN after processing.
$\tilde{x}_i$:	Normalised value of the time series data at index i.
x_i:	Individual element of the time series.
$\bar{x}_i$:	Normalised value of the time series element x_i.
$x[n]$:	The current value of the signal at time index n.
$x(f)$:	The complex value of the signal at frequency bin f.
y:	The additional input representing the label or class information, conditioning both the Generator and Discriminator in cWGAN-GP.
y:	In both MMD and EMD, it represents a sample from the generated data distribution P_g.
(x, x'):	Independent samples were drawn from the real image distribution P_r in GANs.
(y, y'):	Independent samples were drawn from the generated image distribution P_g in GANs.
$\prod(P_r, P_g)$:	Set of all possible joint distributions (couplings) between P_r and P_g that describe how mass is transported from one distribution to the other.
$\|x - y\|$:	Euclidean distance between points x and y, representing the cost of transporting mass from x to y.
$\|\mu_r - \mu_g\|$:	Euclidean norm (or L2 norm) of the difference between the mean vectors of real and generated images.

γ: Transport plan that specifies the amount of mass to be moved from each point in P_r to each point in P_g.

$\gamma(x, y)$: Specific transport plan or joint distribution that details the work needed to transform Pr into Pg from point x to point y.

β: Load angle affects the distribution of forces within the IM bearings.

ϕ: The angle computed using the inverse cosine of the normalised time-series signal $\tilde{x}_i$. Used in transforming the time series into polar coordinates for GAF image creation.

$\psi(t)$: The mother wavelet function is a basic wave that is scaled and translated to match the signal.

$\psi_{(s,\tau)}(t)$: A scaled and translated version of the mother wavelet function at time t, used in WT.

$\psi^*\left(\frac{t-\tau}{s}\right)$: The Continuous Wavelet Transform (CWT) of the signal $x(t)$ using the wavelet function ψ, scaled by ss and translated by τ.

λi: The i-th singular value from the WT represents the magnitude of coefficients in the analysis.

μ_g: The mean feature vector for generated images, used in the FID calculation to compare generated and real images.

μ_r: The mean feature vector for real images, used in calculating the FID.

τ: The translation factor in the WT shifts the wavelet function along the time axis.

$\bar{\varphi}$: The mother wavelet function is used in WT to analyse the signal.

φ: Angle in polar coordinates, calculated as the inverse cosine of x_i in GAF.

θ_1^l, θ_2^l: Parameters of the functions $f1$ and $f2$, respectively, at layer l in a GNN.

List of Publications

Journal Articles:

1. Hejazi, S.Z., Packianather, M. and Liu, Y. 2024. A Novel Customised Load Adaptive Framework for Induction Motor Fault Classification Utilising MFPT Bearing Dataset. *Machines* 12(1), p. 44. Available at: https://www.mdpi.com/2075-1702/12/1/44.

Conference Papers:

1. Hejazi, S., Packianather, M. and Liu, Y. 2023. A Novel approach using WGAN-GP and Conditional WGAN-GP for Generating Artificial Thermal Images of Induction Motor Faults. *Procedia Computer Science* 225, pp. 3681–3691. Available at: https://doi.org/10.1016/j.procs.2023.10.363.
2. Hejazi, S., Packianather, M. and Liu, Y. 2024. Using DCGAN and WGAN-GP to Generate Artificial Thermal RGB Images for Induction Motors. In: Proceedings of the Cardiff University Engineering Research Conference 2023. Cardiff University Press, pp. 113–117. Available at: https://cardiffuniversitypress.org/site/chapters/e/10.18573/conf1.aa/.
3. Hejazi, S., Packianather, M. and Liu, Y. 2022. Novel Preprocessing of Multimodal Condition Monitoring Data for Classifying Induction Motor Faults Using Deep Learning Methods. In: *2022 IEEE 2nd International Symposium on Sustainable Energy, Signal Processing and Cyber Security (iSSSC)*. Gunupur Odisha, India, 15–17 December 2022: IEEE, pp. 1–6. Available at: https://ieeexplore.ieee.org/document/10051321/

Chapter 1: Introduction

1.1 Background

Induction Motors (IMs) play a crucial role across various industries, but a significant percentage of IM failures, estimated at 40% to 50%, stem from issues related to rolling bearings (Frosini and Bassi, 2010). Recent studies have highlighted that bearing faults account for up to 50% of mechanical failures in high-power IMs, underscoring their critical importance in modern machinery (Nishat Toma et al., 2021). Furthermore, a 2020 IEEE survey focused on 200 hp motors revealed that bearing faults constituted more than 40% of all IM faults (Sihag and Sangwan, 2020).

IMs are widely recognised in manufacturing for their simplicity, affordability, and reliability, powering nearly 40% of global electric consumption across diverse industrial sectors. These motors, characterised by rotating components, such as rotors, bearings, and gears, rely heavily on bearings for smooth operation. Bearings typically consist of inner and outer races enclosing rolling balls within a cage to maintain uniform ball spacing. IM faults due to excessive loads, fatigue, inadequate lubrication, and misalignment pose significant operational and safety risks (Toma et al., 2022a).

Bearings play a critical role in supporting IM components to ensure smooth rotation. Typically, a bearing consists of inner and outer races enclosing rolling balls within a cage, maintaining consistent ball spacing. Bearing faults often develop gradually, underscoring the importance of early detection to minimise their impact and associated risks. As these faults progress, they can impair IM performance, threaten worker safety, disrupt operational efficiency, compromise product quality, and lead to substantial maintenance costs (Frosini and Bassi, 2010; Sihag and Sangwan, 2020; Wei et al., 2021).

Hence, the urgent need to establish robust condition monitoring systems for IM machines is evident. Leveraging Industry 4.0 capabilities and available data to develop Data-driven Fault Diagnosis (DFD) systems incorporating Deep Learning (DL) techniques for feature extraction and pattern recognition is essential to address these challenges (Niu et al., 2020; Nishat Toma et al., 2021). In the Artificial Intelligence (AI) era of advanced manufacturing, a dependable condition monitoring system for fault detection and recognition is indispensable to uphold stringent quality standards and effectively manage the production process (Niu et al., 2020).

In the realm of maintenance, Non-Invasive Inspection (NII) is a well-established tool for monitoring the health of machinery. This technique enables the assessment of the current health status without disrupting ongoing operations. NII can be categorised into two main approaches based on sensing methods. The first is Contact-based Non-Invasive Inspection (CNI), which involves placing sensors directly on the machine's body. This includes techniques such as magnetic flux sensing, voltage analysis, machine current analysis, vibration analysis, and wear debris monitoring. The second approach is Non-Contact-Based Non-Invasive Inspection (NCNI), whereby sensors are not directly attached to the inspected part of the machinery system. NCNI methods encompass technologies like Radio Frequency (RF), radar technology, ultrasonic sensing, camera-based imaging, Acoustic Emission (AE) sensing, thermographic sensing (which utilises Infrared (IR) technology), and laser-based techniques (Alotaibi et al., 2021).

Among these approaches, Vibration Signal Analysis (VSA) is recognised as the conventional method for fault classification (Jia et al., 2019). However, in the domain of rotational-machine fault diagnosis using signals, signal preprocessing can be conducted using various techniques, including time-domain, frequency-domain, or time-frequency domain analysis (Sinitsin et al., 2022). On the other hand, thermal imaging has demonstrated its superiority in terms of fault classification accuracy compared to vibration signals, as supported by research conducted by Jia et al. (2019), McGhan and Feayherston (2020), and Shao et al. (2021). Thermal image-based condition monitoring can achieve nearly 100% accuracy by leveraging Convolutional Neural Network (CNN) transfer learning capabilities, with the added benefit of requiring less preprocessing than vibration signal fault classification (Choudhary et al., 2021; Khanjani and Ezoji, 2021). Furthermore, thermal images exhibit less sensitivity to speed fluctuations, making them more efficient in specific scenarios (Shao et al., 2023).

However, it is essential to acknowledge the limitations of thermal images, including the installation costs for cameras and the potential for camera misalignment, which can affect the recognition process (Gangsar and Tiwari, 2020). Additionally, the limited availability and imbalanced distribution of thermal images across specific or all health conditions can significantly affect the performance of condition monitoring systems (Niu et al., 2020). Consequently, each input has its strengths and limitations. The motivation for the

current research stemmed from recognising the complementary nature of vibration signals and thermal images and the need to address data availability issues and incorporate load-dependent factors. This comprehensive exploration of various aspects of condition monitoring involves combining modalities, enhancing accuracy, and considering load-dependent factors.

This thesis proposes a multifaceted approach to enhance IM condition monitoring in light of these considerations. Firstly, the thesis introduces a novel preprocessing technique which combines contact- and non-contact-based sensing methods, specifically vibration signals and thermal images. This approach addresses the limitations of thermal image fault classification found in the literature, including noise and local blur, which can hinder fault recognition (Fan et al., 2022). By integrating vibration signals as a complementary data source, it is possible to develop a more reliable condition monitoring system capable of mitigating noisy data through a holistic view and valuable knowledge extraction from diverse factors. This approach contributes to the multimodal paradigm and multi-sensor fusion by proposing a holistic multi-sensor fault classification methodology with a novel preprocessing technique that creates a fused image incorporating valuable knowledge extracted from various sources using CNNs and DL capabilities. The thesis also explores signal encoding techniques, including Continuous Wavelet Transform (CWT) and Gradient Angular Difference Field (GADF).

Second, this thesis addresses the need to generate an artificial thermal image dataset mimicking real images under seven health conditions. These conditions include bearing damages, such as the Inner Race Fault (IRF) type, Outer Race Fault (ORF) type, and ball damage, as well as rotor damages, including one broken bar, four broken bars, and eight broken bars, in addition to a Normal (fault-free) or Healthy condition. This approach offers a promising solution to address the lack of public datasets containing IM thermal images representing different health states. This is achieved by utilising various Generative Adversarial Network (GAN) architectures, namely, the Deep Convolutional Generative Adversarial Network (DCGAN), Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP), and conditional Wasserstein Generative Adversarial Network with Gradient Penalty (cWGAN-GP).

Third, this thesis advances traditional fault classification methodologies by introducing a Customised Load Adaptive Framework (CLAF), which accounts for load variations and dataset customisation. The CLAF represents a pioneering approach, employing a meticulous two-phase process to reveal load-dependent fault subclasses that are not readily identified by traditional methods. The study explores how radial load characteristics influence fault behaviours, employing advanced techniques such as Time and Frequency Domain (TFD) feature extraction, feature reduction, CWT for time-frequency analyses, Wavelet Singular Entropy (WSE), and CWT energy to identify novel load-dependent fault subclasses. The CLAF is customised and tested on the Machinery Failure Prevention Technology (MFPT) bearing dataset to reveal intricate load-dependent patterns, providing a profound understanding of the interplay between load dynamics and bearing fault behaviour. Various Machine Learning (ML) classifiers, including Wide Neural Network (WNN), Cubic Support Vector Machine (CubicSVM), and Fine Tree, are incorporated to validate the proposed framework.

Fourth, the thesis proposes the Load-Dependent Multimodal Vibration Signal Enhancement and Fusion (LD-MVSEF) methodology to improve the classification accuracy of CLAF load-dependent fault subclasses. Identifying appropriate features has been recognised as a challenge, as it can be time-consuming and, in certain cases, impractical, particularly for specific faults or complex machinery (Resendiz-Ochoa et al., 2018). This method employs a three-channel decision fusion technique, integrating GADF, CWT, and time and frequency domain features. By utilising this multimodal approach, LD-MVSEF enhances fault classification accuracy and enables more precise, load-specific condition monitoring. It consolidates data from multiple channels, optimising classification across various load conditions and facilitating informed decision-making.

Fifth, the thesis introduces the Hybrid Graph-CNN Decision Fusion (HG-CDF) approach, which also focuses on improving the classification accuracy of CLAF load-dependent fault subclasses. A key challenge in using Graph Neural Networks (GNNs) lies in handling multivariate sensor data, where each sensor represents different factors, often causing confusion during training (Deng and Hooi, 2021). In contrast, HG-CDF focuses exclusively on tabular vibration signals, transforming features from the TFD into graph structures using the k-Nearest Neighbours (kNN) method. It combines the strengths of

Graph Convolutional Networks (GCNs), which capture complex relationships within graph data, and One-Dimensional Convolutional Neural Networks (1D-CNNs), which excel at detecting sequential patterns in time-series data.

While the previous literature has seldom explored the use of GCNs for fault classification in IMs, particularly for fault classes derived from the CLAF, GNNs and k-Nearest Neighbour Graphs (k-NNGs) have been shown to be complementary techniques for analysing graph data. In particular, k-NNG is essential for constructing graphs from data points (Wang et al., 2021b; Rangel-Rodriguez et al., 2023), while GNNs excel at identifying patterns and relationships within graph data, as demonstrated in various fields such as micro-service systems (Zhang et al., 2023b), power systems (Su et al., 2021), and fault location in power networks (Mo et al., 2023).

1.2 Aim and Objectives

The aim of this research is to enhance the accuracy of fault classification in IM bearings by developing and implementing novel artificial intelligence (AI) and Machine Learning (ML) techniques utilising condition monitoring data. The research objectives are organised into five main themes as follows:

1) **Multimodal Data Preprocessing Methodology:** To develop a preprocessing methodology that integrates multimodal data (thermal images and vibration signals) to improve fault classification accuracy. *Discussed in Chapter 3*.
2) **Artificial Thermal Image Creation:** To create high-quality artificial thermal images using conditional Generative Adversarial Networks (cGANs) to represent various IM health conditions. *Explored in Chapter 4*.
3) **Customised Radial Load Assessment:** To develop a Comprehensive Load-dependent Analysis Framework (CLAF) for classifying IM faults into load-dependent subclasses. *Detailed in Chapter 5*.
4) **Multimodal Systems Decision Fusion Approach:** To develop a Load-Dependent Multimodal Vibration Signal Enhancement and Fusion (LD-MVSEF) method for improved CLAF load-dependent fault subclass classification accuracy. *Described in Chapter 6.*

5) **Graph Convolutional Network (GCN) on a Tabular Dataset Application:** To apply a Graph Convolutional Network (GCN) for classifying CLAF load-dependent fault subclasses. *Outlined in Chapter 7.*

1.3 Alignment of Research Objectives with Methodologies and Chapter Structures

The research objectives outlined in Section 1.2 are systematically addressed throughout the chapters of this thesis, ensuring a structured approach to achieving the overall aim. The objectives and chapters are linked through the integration of multimodal data, which includes four key inputs:

1) Raw Vibration Signal (from the Machinery Failure Prevention Technology (MFPT) Bearing Dataset): This dataset contains unprocessed vibration signals from MFPT bearing, serving as primary indicators of IM bearing conditions. The signals were recorded under various conditions, including healthy states, ORF, and IRF.
2) Lab-Collected Thermal Images (Cardiff University): Thermal images were captured using an FLIR thermal camera at Cardiff University's Wolfson Magnetics Laboratory. These images document the thermal profiles of IM bearings under different load conditions (8 bars, 4 bars, and 1 bar) and specific faults (IRF, ORF, and ball faults). A baseline image representing a Normal (fault-free) or Healthy condition is also included.
3) Compromised Quality Thermal Images (Simulating Real-World Conditions): These images are artificially degraded versions of the lab-collected thermal images. They simulate real-world scenarios where thermal images may be noisy or unclear, helping to test the robustness of fault detection algorithms under suboptimal conditions.
4) Categorised Load Factor (LF) (from the MFPT Bearing Dataset): The MFPT dataset is categorised by different operational load conditions (e.g., 50, 100, 150, 200, 250, and 300). This categorisation allows for the analysis and classification of faults with respect to varying loads, which is crucial for developing load-specific monitoring and fault classification techniques.

Figure 1.1 illustrates how each research objective aligns with the specific methodologies and analyses detailed in the thesis chapters, ensuring a coherent progression toward improving fault classification accuracy in IM bearings using AI and ML techniques.

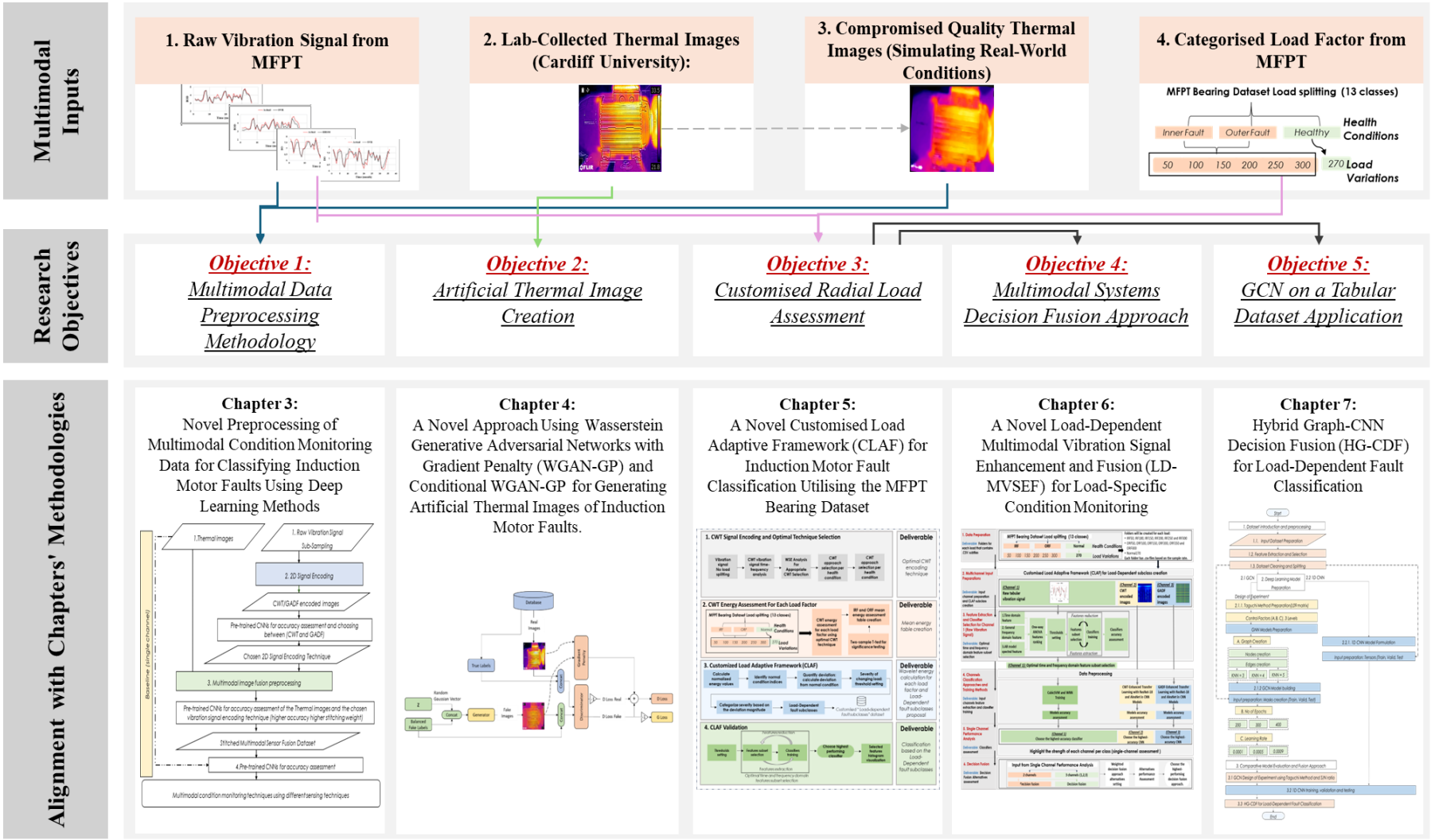


Figure 1.1: Alignment of Research Objectives with Chapters' Methodologies.

The research objectives are explored in depth in the following chapters (refer to Figure 1.1):

1) Objective 1: Develop a Multimodal Data Preprocessing Methodology

Chapter 3: This chapter addresses the preprocessing stage, where noisy thermal images (input 3) and vibration signals (input 1) from different datasets are merged into a unified image. It also assesses different signal encoding methodologies to enhance fault classification accuracy in induction motor (IM) bearings.

2) Objective 2: Create High-Quality Artificial Thermal Images

Chapter 4: This chapter focuses on the creation of high-quality artificial thermal images using Generative Adversarial Networks (GANs). It explores the generation of artificial thermal images from (input 2) that represent various induction motor (IM) health conditions, thereby improving fault detection capabilities.

3) Objective 3: Develop a Comprehensive Load-dependent Analysis Framework (CLAF)

Chapter 5: This chapter introduces the Customised Radial Load Assessment, developing a Comprehensive Load-dependent Analysis Framework (CLAF) that classifies induction motor faults into load-dependent fault subclasses based on varying LFs from the MFPT dataset (input 4).

4) Objective 4: Establish a Load-Dependent Multimodal Vibration Signal Enhancement and Fusion (LD-MVSEF) Method

Chapter 6: Building on Chapter 5 load-dependent fault subclasses, this chapter presents the LD-MVSEF method, which integrates multimodal data (including GADF and CWT. The thesis also explores signal encoding techniques, including Continuous Wavelet Transform (CWT) and Gradient Angular Difference Field (GADF) images) to improve the classification accuracy of CLAF load-dependent fault subclasses using advanced decision fusion techniques tailored to specific load conditions.

5) Objective 5: Apply Graph Convolutional Networks (GCNs) to Tabular Datasets

Chapter 7: Building on Chapter 5, this chapter explores the application of Graph Convolutional Networks (GCNs) for classifying load-dependent fault subclasses. It involves constructing graphs from tabular datasets and applying GCNs to enhance classification accuracy and efficiency by leveraging the relational dynamics within the

data. Additionally, it proposes a Hybrid Graph-CNN Decision Fusion (HG-CDF) approach, which focuses on further improving the classification accuracy of CLAF load-dependent fault subclasses.

1.4 Research Questions

Based on the above objectives, this research aims to answer the following questions:

- **Q1:** How can the integration of multimodal data sources, specifically thermal images and vibration signals, enhance the accuracy of fault classification in induction motor (IM) bearings, especially under compromised thermal image quality?
- **Q2:** How effective are conditional Generative Adversarial Networks (cGANs) in generating high-quality artificial thermal images for IM health conditions, and how do these images compare with the original dataset?
- **Q3:** How can a Comprehensive Load-dependent Analysis Framework (CLAF) be developed to classify IM bearings faults into load-dependent subclasses ('Mild,' 'Moderate,' and 'Severe') based on varying LFs?
- **Q4:** How can the Load-Dependent Multimodal Vibration Signal and Energy Fusion (LD-MVSEF) method achieve high accuracy in load-specific condition monitoring of IMs, and what are the benefits of using a weighted decision fusion technique?
- **Q5:** How can transforming time and frequency domain features into k-Nearest Neighbour Graphs (k-NNGs) and applying them to a Graph Convolutional Network (GCN) enhance the accuracy and efficiency of load-dependent fault classification in IM bearings, and how does integrating a GCN with a 1D-CNN in a hybrid approach further improve this classification?

1.5 Thesis Outline and Contribution

This thesis is organised as follows:

Chapter 1: Introduction

This chapter provides an overview of the research context, articulates the study's purpose, sets out the objectives and research questions, outlines the thesis structure, and discusses the limitations and assumptions of the research.

Chapter 2: Literature Review

This chapter explores Artificial Intelligence (AI) techniques applicable to IMs, focusing on ML and DL. It covers AI algorithms, DL architectures, two-dimensional (2D) vibration signal encoding techniques, and feature extraction methods. The chapter highlights state-of-the-art research across five key themes: Multimodal Data Preprocessing, Artificial Thermal Image Creation, Customised Radial Load Assessment, Decision Fusion in Multimodal Systems, and GCNs on Tabular Datasets. It also identifies research gaps within each theme.

Chapter 3: Novel Preprocessing of Multimodal Condition Monitoring Data for Classifying Induction Motor Faults Using Deep Learning Methods

This chapter presents a novel preprocessing approach for multimodal data in fault classification of IMs using DL methods. The Stitched Multimodal Image Dataset Encoding Technique integrates vibration signals and thermal images through signal-to-image encoding techniques, such as CWT and Gramian Angular Field (GAF). By applying CNN architectures like Residual Network (ResNet) and SqueezeNet, the study demonstrates that this multimodal feature fusion enhances classification accuracy, particularly under IRF conditions, even with lower-quality thermal images. The proposed approach improved classification accuracy by 12.50%, achieving 99.10% ± 0.50% when using both ResNet-18 and SqueezeNet compared to using compromised thermal images alone.

This chapter's main contribution is as follows:

1) Proposing a novel preprocessing method for multimodal condition monitoring data to classify IM faults using DL techniques.

Chapter 4: A Novel Approach Using Wasserstein Generative Adversarial Networks with Gradient Penalty (WGAN-GP) and Conditional WGAN-GP for Generating Artificial Thermal Images of Induction Motor Faults

This chapter investigates the use of GANs for generating artificial thermal images of IM faults. Initially, the DCGAN is evaluated to establish a baseline for generating these images. The chapter then introduces the Wasserstein GAN with Gradient Penalty (WGAN-GP) and the cWGAN-GP, which produce thermal images closely resembling real ones. The cWGAN-GP model achieved a Maximum Mean Discrepancy (MMD) score of 1.023, indicating strong similarity to real images, while the WGAN-GP outperformed it with an Earth Mover's Distance (EMD) score of 4.663 compared to cWGAN-GP's 4.816.

Additionally, the cWGAN-GP dataset achieved a classification accuracy of 98.41% using a pre-trained AlexNet model.

This chapter's main contributions are as follows:

1) Exploring the use of DCGAN and WGAN to generate artificial thermal images that closely mimic real thermal images of IM bearing faults.
2) Introducing a novel approach using WGAN-GP and cWGAN-GP for generating artificial thermal images of IM faults.

Chapter 5: A Novel Customised Load Adaptive Framework (CLAF) for Induction Motor Fault Classification Utilising the MFPT Bearing Dataset

This chapter presents the CLAF for classifying IM faults into load-dependent subclasses: ''Normal (fault-free) or Healthy condition,' 'Mild,' 'Moderate,' and 'Severe.' The framework improves traditional fault classification by incorporating load variations and tailoring the analysis to specific conditions. Developed in two phases, the first phase identifies load-dependent patterns using statistical ranking and ML classifiers, while the second phase refines classification with CWT techniques. The chapter details the design, methodology, and application of the CLAF for enhanced fault classification.

Thus, the contributions of this chapter are as follows:

1) Conducting a comprehensive TFD analysis under six load conditions to reveal patterns and variations in fault severity.
2) Selecting an optimal CWT approach using Wavelet Singular Entropy (WSE) to improve feature extraction, denoising, and pattern recognition.
3) Introducing a method for identifying and classifying load-dependent fault subclasses, including 'Mild,' 'Moderate,' and 'Severe,' which enhances the understanding of fault severity under different load scenarios.
4) Proposing the CLAF, extending traditional fault classification methods by incorporating load variations and customising the analysis for different IM datasets.

Chapter 6: Novel Load-Dependent Multimodal Vibration Signal Enhancement and Fusion (LD-MVSEF) for Load-Specific Condition Monitoring

This chapter introduces the LD-MVSEF approach, which advances load-specific condition monitoring by building on the CLAF. The LD-MVSEF method improves the classification accuracy of CLAF load-dependent fault subclasses by integrating raw

vibration feature extraction with signal encoding techniques such as CWT and GADF image conversion. It employs various classifiers to enhance load-dependent fault classification accuracy, achieving 99.04% ± 0.22% across five runs, with an average training time of 18 min and 30 s, providing a valuable methodology for monitoring machinery conditions.

This chapter's main contributions are as follows:

1) Proposing the LD-MVSEF approach, which integrates information from GADF, CWT, and time-frequency domain data to enhance Load-Dependent Fault Classification, building on the CLAF. This approach improves accuracy, particularly by using a weighted decision fusion method.
2) Combining diverse analytical dimensions, including one-dimensional (1D) vibration signals and two-dimensional (2D) RGB images (CWT and GADF-encoded), to improve classification accuracy.

Chapter 7: Hybrid Graph-CNN Decision Fusion (HG-CDF) for Load-Dependent Fault Classification.

This chapter highlights the potential of GCNs for condition monitoring, particularly in scenarios requiring fast model training and accurate fault categorisation. By transforming tabular data into graph structures using the kNN method, GCNs demonstrated a strong performance in load-dependent fault classification, with a mean accuracy of 89.01% ± 1.25% across nine experiments using the Taguchi design, where each experiment takes around 28 s. However, the GCN performed lower in the Mild class, prompting the introduction of the HG-CDF method, which integrates GCN and 1D-CNNs. The hybrid approach significantly improved accuracy across all CLAF load-dependent fault subclasses, achieving 99.19% overall accuracy while maintaining computational efficiency with a total training time of 3.28 min.

The chapter's main contributions are as follows:

1) Utilising CLAF subclasses in GCNs by refining fault classification through incorporating CLAF load-dependent subclasses ('Mild,' 'Moderate,' 'Severe,' and 'Healthy'), offering a tailored approach to condition monitoring.
2) Advancing Feature Extraction by moving beyond raw VSA by applying advanced feature extraction techniques from both time and frequency domains, enhancing node relationship modelling, and improving fault detection accuracy.

3) Proposing GCNs using Taguchi, which involved transforming data representation by introducing GCNs for fault classification in IMs, using the k-NNG method to transform traditional tabular data into graph-based structures, capturing relational dynamics and advancing the potential of GCNs in this field and selecting the optimal GCN configuration using Taguchi.
4) Proposing GCN using Taguchi with Selective Weighted Loss (SWL) to enhance class-specific accuracy, with a particular focus on improving the performance of the Mild class. By adjusting the model's focus, SWL effectively boosted the accuracy of the Mild class while ensuring strong results were maintained across other classes.
5) Proposing an HG-CDF, which combines GCNs and 1D-CNNs to address the GCN's limitations in the Mild fault class and in order to improve classification performance across all subclasses.

Chapter 8: Conclusion

The final chapter summarises the key contributions of the thesis, identifies the study's limitations, and offers recommendations for future research.

1.6 Thesis Limitations and Assumptions

While this thesis primarily focuses on the MFPT bearing dataset for IM condition monitoring (a publicly available resource), it is essential to acknowledge this research's specific scope and context. The proposed frameworks are highly detailed, providing step-by-step procedures that facilitate easier customisation of different datasets or industrial settings. However, the current research is tailored to the MFPT bearing dataset and may not cover all possible real-world scenarios. While efforts have been made to ensure the representativeness of the MFPT bearing dataset, inherent limitations associated with any dataset may exist. The study primarily explores vibration signals and thermal images as data sources, leaving scope for the potential exploration of other types of data. The thermal images used in this study were collected in a controlled laboratory environment at Cardiff University, representing seven distinct health conditions with artificially created faults. Although carefully designed, these conditions may not perfectly replicate all real-world scenarios.

Chapter 2: Literature Review

2.1 Induction Motors (IMs)

Induction Motors (IMs) play a crucial role in the manufacturing sector and are valued for their straightforward operation, cost-effectiveness, and dependability. They account for nearly 40% of global electricity consumption and are integral across diverse industries (Toma et al., 2022a). A defining characteristic of any rotating machinery, including IMs, is its components, such as rotors, bearings, and gears. Bearings ensure smooth motor operation, comprising inner and outer races, rolling balls, and a cage that maintains uniform ball spacing. Potential IM faults can arise from excessive loads, fatigue, insufficient lubrication, and misalignment (Toma et al., 2022a).

2.1.1 Induction Motor Bearing Fault Frequency

Each bearing element has a rotating frequency. When a defect occurs, and the rolling part moves across this damaged part, the vibration energy also deviates at a fixed rate, generating periodic impulses. So, each defect (Outer Race Fault (ORF), Inner Race Fault (IRF), and balls fault) has a unique frequency, as shown in Equation (2.1), (2.2) and (2.3), where, N_{ball} represents the number of balls, f_m is the rotational frequency, β is the load angle, and D_{Cage} and D_{Ball} refer to cage and ball diameter, respectively (Toma et al., 2022a).

$$f_{ORF} = \frac{N_{ball}}{2} \times f_m \times \left(1 - \left(\frac{D_{ball}}{D_{cage}} \times cos\,\beta\right)\right) \quad \textbf{(2.1)}$$

$$f_{IRF} = \frac{N_{ball}}{2} \times f_m \times \left(1 + \left(\frac{D_{ball}}{D_{cage}} \times \cos\beta\right)\right) \quad \textbf{(2.2)}$$

$$f_{Ball} = \frac{D_{Cage}}{2D_{Ball}} \times f_m \times \left(1 - \left(\frac{D_{ball}}{D_{cage}} \times \cos\beta\right)^2\right) \quad \textbf{(2.3)}$$

2.2 Artificial Intelligence (AI)

Machine Learning (ML) and Deep Learning (DL) fall under the umbrella of Artificial Intelligence (AI), as shown in Figure 2.1. ML, a component of AI, operates autonomously with minimal human intervention and typically relies on structured data. In contrast, DL, a subset of ML, employs Artificial Neural Networks (ANNs) to emulate the learning mechanisms of the human brain. DL thrives on vast datasets and can handle structured and unstructured data (Martin, 2021).

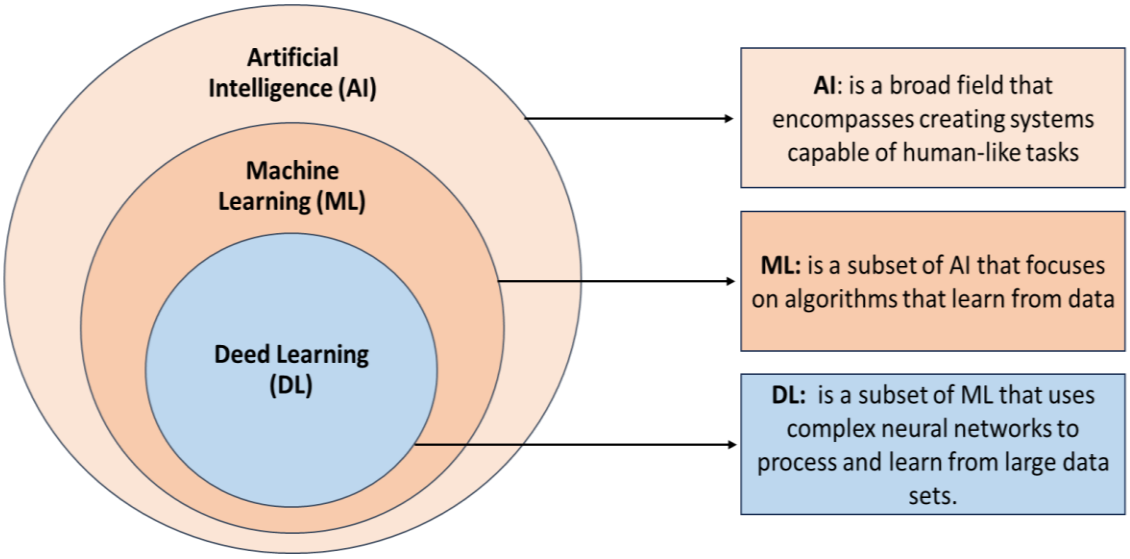


Figure 2.1: The Relationship Between Artificial Intelligence, Machine Learning and Deep Learning (Martin, 2021).

2.2.1 Machine Learning (ML)

ML is a branch of computer science and an AI component that enables computers to learn and make decisions without explicit programming. It is applied across different computational tasks with the primary goal of training machines using provided data, which may be labelled in supervised learning scenarios or unlabelled in unsupervised learning cases, to enhance results for specific problems. The key emphasis in ML is on enabling computers to learn from previous experiences (Mehmood and Selwal, 2020). The choice of data representation is crucial for the performance of ML models in fault classification, impacting accuracy, speed, and generalisability. Key factors include feature selection, which ensures essential information is captured while avoiding overfitting (Kareem and Hur, 2022), data normalisation to balance feature scales (Jang and Cho, 2021), and data augmentation to expand the training set and reduce overfitting (Yousuf et al., 2024). Dimensionality reduction techniques like Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbour Embedding (t-SNE) help by minimising noise (Wodecki and Michalak, 2021). Advanced methods, including graph-based representations (Jang and Cho, 2021), knowledge graphs (Radtke et al., 2023), domain-specific ontologies (Delgoshaei et al., 2022), and hybrid approaches (Chao et al., 2019), integrate domain knowledge and further optimise model performance.

ML can be divided into three categories. Figure 2.2 shows how supervised learning uses labelled data to train an agent with predetermined correct actions, optimising a policy based on explicit feedback like rewards or penalties. Unsupervised learning, lacking labelled data and explicit feedback, involves the agent discovering patterns through trial and error (Tangirala, 2020). Reinforcement learning (RL), the third category, trains an agent through interactions with its environment, where it learns from rewards or penalties to develop a strategy that maximises cumulative rewards, making it suitable for complex environments (Borga and Carlsson, 1992).

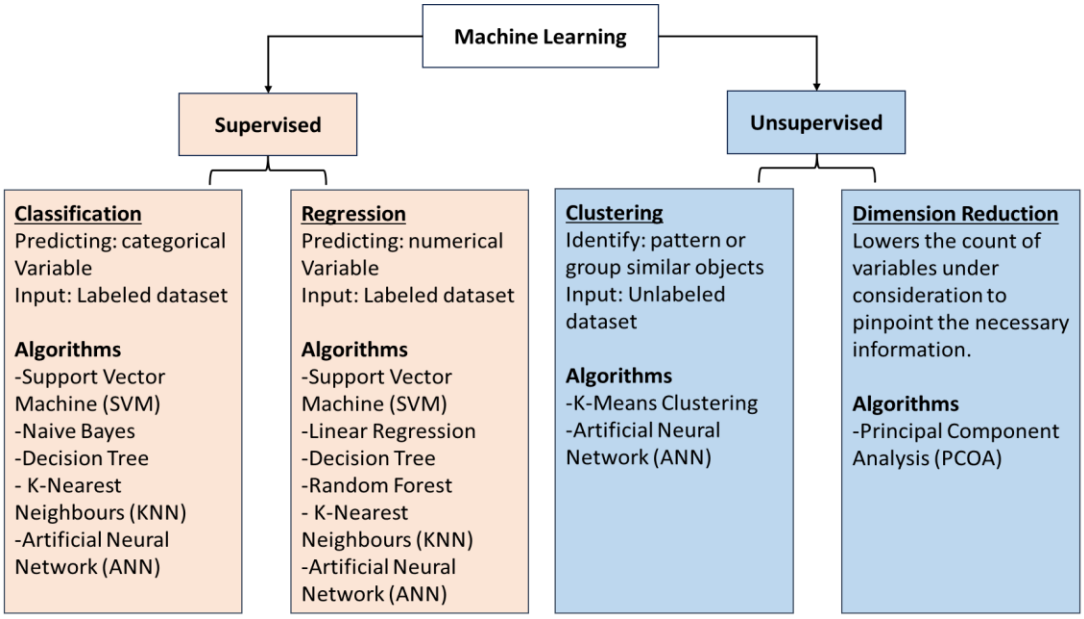


Figure 2.2: Supervised and Unsupervised Machine Learning Techniques (Tangirala, 2020; Edeh et al., 2022).

2.2.1.1 Supervised Learning

Supervised learning in ML features a range of algorithms designed for specific data types and predictive needs. Below are concise descriptions of some frequently utilised supervised learning algorithms (Mehmood and Selwal, 2020; Martin, 2021):

1. Support Vector Machine (SVM): An SVM is a powerful classification method that identifies the optimal hyperplane to separate data into classes, making it particularly effective in high-dimensional spaces. It can also be applied to regression problems. SVM transforms data into a new space using a kernel function, enabling linear classification with the maximum margin between categories. The choice of the kernel—such as Linear, Polynomial, or Gaussian—determines the effectiveness of

this transformation (Khanjani and Ezoji, 2021; Rangel-Rodriguez et al., 2023). A Cubic Support Vector Machine (CubicSVM), a supervised learning classifier, is well-suited for high-dimensional data and structured datasets, making it popular for classification and regression tasks (MathWorks-3, 2024).

2. Naive Bayes: A simple yet powerful classifier based on Bayes' theorem, assuming predictor independence, Naive Bayes is effective for large datasets and is commonly used for tasks like spam filtering and sentiment analysis (Prabha, 2022). It can also be applied to fault detection and maintenance by analysing sensor data to classify system states (Bodo et al., 2021). It predicts defects in processes like investment casting using various process parameters in manufacturing. Different variants, such as Gaussian, Multinomial, Complement, and Bernoulli Naive Bayes, can be evaluated for the best fit (Sawant and Agashe, 2022).
3. Decision Tree (DT): This model visualises decisions through a graph showing various outcomes. It is easy to understand, works for classification and regression tasks, and accommodates categorical and numerical data (Diao and Zhang, 2021). It utilises a DT to classify pairs of managers dealing with maintenance outsourcing cases into either “abnormal” or “normal” behaviour patterns, presenting a binary classification challenge (Chen et al., 2021). Also, the DTs, specifically Classification and Regression Trees (CART), are used to solve classification and regression predictive modelling problems. It illustrates this with an example of predicting a college student’s first-year GPA based on high school GPA, SAT scores, and other relevant parameters (Njoku, 2019).

4. k-Nearest Neighbours (kNN): k-NN can be used for regression and classification tasks within supervised learning. It is not exclusively categorised under one or the other; its application depends on the specific task. It can also be used for classification and regression problems. For classification, kNN identifies the class of a new sample based on the majority vote of its nearest neighbours. For instance, kNN has been applied in classifying brain tumour images, achieving an average accuracy of about 62.00% (Najwaini et al., 2023). For regression, kNN predicts a continuous value for a new sample based on the average (or weighted average) of the values of its nearest neighbours. For instance, kNN is effectively used in stock market forecasting,

demonstrating its strength in numeric prediction tasks by processing relationships between numerical data and achieving an accuracy of 70.00% (Ltha et al., 2022).

5. Artificial Neural Networks (ANNs): ANNs are trained with labelled data in supervised learning to minimise prediction errors. They excel in tasks like gesture recognition using electromyographic (EMG) signals (Shamsin et al., 2018; Mustaqim et al., 2023) and regression tasks like residential load forecasting, outperforming traditional methods like AutoRegressive Integrated Moving Average (ARIMA) (Chandran et al., 2021). During the COVID-19 pandemic, ANNs such as the Multilayer Perceptron (MLP) were used for accurate time series forecasting of cases and deaths, surpassing classical approaches (Borghi et al., 2021). ANNs typically consist of an input layer, fully connected layers with Rectified Linear Unit (ReLU) activation, and a softmax layer for classification. Their complexity ranges from narrow networks with fewer neurons to wide and multi-layered networks that handle intricate data relationships but are harder to interpret (MathWorks-3, 2024; MathWorks-6, 2024)

- Ensemble Learning for Fault Detection

  Ensemble learning is employed to enhance ML model performance for fault detection in machinery. The techniques include a voting classifier, combining predictions from various models like DTs, Random Forests, SVMs, kNNs, and XGBoost, using soft and hard voting methods. The ensemble models developed using these base models show improved accuracy in detecting bearing faults in IMs through Vibration Signal Analysis (VSA). An Ensemble AdaBoost Decision Tree (EADT) method is proposed for defect detection, utilising features extracted via a Stationary Wavelet Transform (SWT). These approaches demonstrate the effectiveness of ensemble methods in achieving more accurate and reliable fault diagnosis in machinery (Jose et al., 2022).

2.2.1.2 Unsupervised Learning

Unsupervised learning is a branch of ML that focuses on identifying patterns in unlabelled datasets, making it valuable for discovering hidden relationships in data. Algorithms are provided with input data without output labels, aiming to detect patterns for

tasks like clustering, association, and rule prediction. Common unsupervised learning techniques include k-means clustering, hierarchical clustering, and PCA (Martin, 2021). The following are some frequently used unsupervised learning methods (Mehmood and Selwal, 2020; Martin, 2021):

1) k-Means clustering: k-Means is a widely used clustering algorithm that groups nearby points into clusters by assigning them to the nearest cluster centre based on distance (Edeh et al., 2022)While simple and effective, k-Means relies on the random selection of initial cluster centres, which can affect its results (Masud et al., 2019). It is also used in software engineering to cluster classes by their attributes, helping identify more maintainable software systems and reducing maintenance time and resources (Mathur and Kaushik, 2018).
2) Artificial Neural Networks (ANNs): While typically used in supervised learning for classification and regression, ANNs can also be adapted for unsupervised tasks to uncover hidden patterns. Autoencoders, a type of ANN, are effective for dimensionality reduction, anomaly detection, and generative modelling, especially with high-dimensional data (Sewak et al., 2020; Wu et al., 2021). Advancements in ANNs for clustering and unsupervised learning are highlighted in studies like those presented at the International Conference on Artificial Neural Networks (ICANN) (Ve et al., 2019). Notable research has demonstrated the integration of deep ANNs with clustering techniques for predicting Noncommunicable Diseases (NCDs), showcasing their effectiveness in disease prediction (Moreno-Gutierrez and Garcia-Lopez, 2023).
3) Principal Component Analysis (PCA): PCA is a statistical method that reduces data complexity by transforming correlated variables into uncorrelated principal components while preserving essential information. It is commonly used in data analysis and predictive modelling. For example, PCA has simplified industrial sensor data for better visualisation and decision-making (Grabowski et al., 2023), reduced pollutant indices in water quality assessments to highlight key contaminants (Xu et al., 2021), and improved efficiency in Magnetic Anomaly Detection for real-time applications (Sheinker and Moldwin, 2016).

2.2.1.3 Reinforcement Learning (RL)

RL is a branch of ML where an agent learns decision-making through rewards or penalties for actions without explicit instructions (Sun, 2020). It is applied in robotics, autonomous driving, healthcare, finance, logistics, and energy management (Thaipisutikul et al., 2019; Xiang and Foo, 2021). Key RL algorithms include Q-Learning, Deep Q-Networks (DQN), Policy Gradient Methods, Actor-Critic Methods, Proximal Policy Optimisation (PPO), Asynchronous Advantage Actor-Critic (A3C), Monte Carlo Tree Search (MCTS), Temporal Difference (TD) Learning, and State-Action-Reward-State-Action (SARSA), each of which is suitable for various complex environments and tasks (Fazel et al., 2018; Haarnoja et al., 2018; Speck and Bucci, 2018; Naresh et al., 2023; Niu et al., 2023; Zhu et al., 2023a; Ekpo and Eke, 2024).

2.2.2 Deep Learning Approaches for Fault Classification

Both ML and DL play a crucial role in improving fault diagnostics by minimising false alarms and enabling early prediction of equipment failures due to their capacity to process large datasets and learn complex, nonlinear relationships (Arellano-Espitia et al., 2020; Zhu et al., 2023b). DL techniques, such as Convolutional Neural Networks (CNNs), Deep Neural Networks (DNNs), and Deep Belief Networks (DBNs), have outperformed traditional ML methods like SVMs, ANNs, and kNN in fault detection, particularly in extracting features from vibration signals (Ye et al., 2020; Gao et al., 2023; Qiu et al., 2023). DL's multi-layered networks are particularly effective in complex diagnostics, enabling time-dependent modelling to capture time-shifted effects (Zhang et al., 2020).

Traditional ML techniques remain foundational in various industries, effectively handling complex datasets in manufacturing and power generation (Zhou et al., 2019; Ren et al., 2021; Elshenawy et al., 2022; Hakim et al., 2023). DL models, inspired by the human brain, consist of interconnected layers (neurons) that excel in areas like image recognition and natural language processing, learning complex patterns from extensive datasets (Abdel-Jaber et al., 2022; Kufel et al., 2023). Table 2.1 summarises commonly used DL algorithms, their applications, advantages, and limitations.

Table 2.1: Common Deep Learning Networks.

Algorithm Type	Applications	Advantages	Limitations	References
Convolutional Neural Networks (CNNs)	Object detection, image classification, segmentation, facial recognition, autonomous driving, and medical image analysis.	Excel at identifying spatial and temporal relationships in data, inherently capable of handling data translation.	They require extensive labelled data and high computational demand.	(Abdel-Jaber et al., 2022) (Wang and Sng, 2015)
One-Dimensional Convolutional Neural Networks (1D-CNNs)	Handling and learning from 1D sequential inputs, such as financial and structural sensor data.	Feature a more straightforward design than higher-dimensional CNNs, enabling faster training and direct processing of 1D sequential data like time-series without conversion.	They are less effective for higher-dimensional data than deeper, multi-dimensional CNNs.	(Tran et al., 2024) (Liu and Si, 2022) (Xiao et al., 2021a)
Generative Adversarial Networks (GANs)	Image and video enhancement, data augmentation, cybersecurity.	Are known for their capability to produce high-quality, realistic synthetic data, especially valuable in areas where data are rare or costly to acquire.	GANs require considerable computational resources to train two models—the generator and discriminator—through an iterative process.	(Alqahtani et al., 2021) (Sauer et al., 2021)
Graph Neural Networks (GNNs)	Graph-structured data, including computer vision, bioinformatics, recommendation systems, traffic forecasting, anomaly detection in time series data.	Can effectively capture and leverage both node features and graph topology/structure	They can be computationally expensive for large graphs.	(Abdel-Jaber et al., 2022) (Chen et al., 2022a)
Recurrent Neural Networks (RNNs)	Language modelling, machine translation, speech recognition, and video analytics for urban surveillance,	Apt at processing sequential data, learning from past inputs	Prone to vanishing/exploding gradients, computationally intensive.	(Guney et al., 2021) (Wang and Sng, 2015)
Long Short-Term Memory (LSTM) / Gated	Language modelling, machine translation, speech recognition.	Superior at managing long-term dependencies in sequence data compared to traditional RNNs.	Its complex structure has a high computational load.	(Abdelrazik et al., 2023) (Xue et al., 2022)

Algorithm Type	Applications	Advantages	Limitations	References
Autoencoders	Dimensionality reduction, denoising, anomaly detection, data compression.	Capable of unsupervised learning, autoencoders excel at data representation and compressing input into smaller forms for effective dimensionality reduction and data compression.	Training is challenging and can result in data loss, with a sensitivity to hyperparameters and initialisation that requires careful tuning.	(Abdel-Jaber et al., 2022) (Koehler et al., 2021) (Refinetti and Goldt, 2022)
Deep Belief Networks (DBNs)	Dimensionality reduction, feature learning, collaborative filtering	Can be trained layer by layer, making the training process more efficient than training the entire network.	High computational costs, challenging training process.	(Li et al., 2023a) (Zambra et al., 2023)
Deep Boltzmann Machines (DBMs)	DBNs are well-suited for dimensionality reduction, feature learning, and initialising feedforward neural networks.	Utilise top-down feedback connections across layers for both training and inference. Unlike DBNs, which are trained layer-by-layer, all DBM layers are trained simultaneously, enabling them to capture dependencies and influences across layers.	Practical use of DBMs has been more limited than other DL models, primarily because of their complex training requirements.	(Taniguchi et al., 2023) (Li et al., 2023a) (Souza et al., 2017) (You et al., 2013)

2.2.2.1 Advancements in Deep Learning for Fault Classification

The evolution from basic neural networks to advanced models like CNNs and Graph Neural Networks (GNNs) has been driven by the need to process complex data and patterns across various domains. This progression has focused on optimising DL architectures to handle more intricate tasks and integrate graph structures for improved performance. In the 1990s, neural networks were primarily applied to RL in partially observable Markov Decision Processes (POMDPs). By the late 2010s, the focus had shifted to optimising network structures, hyperparameters, and training methods, enhancing computational power and complexity (Miikkulainen, 2023).

CNNs, developed in the 1980s, became essential in image processing and three-dimensional (3D) construction, often paired with Generative Adversarial Networks (GANs)

for cost-effective model creation (Lyu and Yu, 2021). GNNs, a newer development, extend neural network capabilities by incorporating graph structures, although they sometimes struggle in heterophilic environments where connected nodes differ. Recent advances, such as Adaptive Channel Mixing (ACM), address these limitations by dynamically adjusting information aggregation across nodes (Luan et al., 2022).

In predictive maintenance, particularly for Induction Motor (IM) fault classification, DNNs and GANs play critical roles. DNNs, primarily used in supervised learning, have shown high fault detection and diagnosis accuracy, surpassing GAN-based oversampling techniques (Lee et al., 2017). GANs, effective in unsupervised tasks like data generation and augmentation, enhance predictive maintenance by estimating missing values and predicting faults (Lee et al., 2020). Additionally, ANNs combined with Park's vector analysis have achieved over 99.00% accuracy in motor health classification (Mahesh et al., 2022).

2.2.3 Convolutional Neural Networks (CNNs)

CNNs are specialised Neural Networks (NNs) optimised for pattern recognition, particularly in computer vision and image processing. Inspired by the human brain's visual cortex, CNNs are structured to handle grid-like data, such as images, effectively, utilising local connectivity and spatial relationships to learn efficiently. Consequently, CNNs are considered powerful tools for image and video recognition. They consist of two main parts: the feature extractor and the classifier. The feature extractor uses specialised layers to find essential patterns in the input data. It includes convolutional layers that detect local features and pooling layers that reduce the data's size. These layers work together to create a hierarchical representation of the input. After the feature extractor, the classifier makes predictions based on the extracted features (LeCun et al., 1998).

All types of CNNs share three essential layers: the convolutional layer, the pooling layer, and the fully-connected layer. While the softmax function plays a crucial role in the output layer of a CNN, it is not considered one of the fundamental layers in a basic CNN architecture softmax function applied at the output layer for multi-class classification. The convolutional layer utilises convolutional operations to extract more advanced feature representations. These operations help to identify patterns and structures within the input data, enabling CNN to learn meaningful features. Next, the pooling layer downsamples the data through local averaging or selecting the maximum value. This downsampling process

concentrates the extracted features, enhancing the efficiency of the CNN. Lastly, the fully-connected layer aims to understand the relationship between the input and output of the CNN. It inputs the features that have passed through the convolutional and pooling layers. By analysing these features, the fully connected layer generates the final output of the CNN. Combining the convolutional layer, the pooling layer, and the fully connected layer forms the foundation of CNNs, allowing them to effectively process and extract relevant information from the input data (Yuan et al., 2020; Zhang et al., 2021a).

2.2.3.1 Transfer Learning with Convolutional Neural Networks

Transfer learning with CNNs involves leveraging previously acquired knowledge in classification tasks and applying that knowledge to similar problems within the same domain or experimental contexts using the same pre-trained classifier. This approach offers significant advantages by reducing the need for extensive training time, large datasets, and computational resources (Cinar, 2022). Training a DL model from scratch demands considerable time and numerous parameter adjustments, presenting significant challenges. However, transfer learning has been introduced to address these challenges. Transfer learning involves transferring knowledge and patterns from a source domain to a target domain, often entailing the reuse of a pre-trained model on a new dataset. As a result, similarities between datasets are identified (Bai et al., 2022).

Several pre-trained CNN transfer learning architectures are widely used across various domains, including medical imaging, traffic sign recognition, food image classification, and clinical predictions. Key architectures include Residual Network (ResNet), known for its benchmark-setting performance in image classification; Visual Geometry Group Network (VGGNet), effective on the ImageNet dataset; MobileNet, optimised for mobile applications; Inception-v3 and EfficientNet-B0, both renowned for their top-tier performance; Extreme Inception (Xception) and DenseNet-121, excelling in image classification benchmarks; and TimeNet, a deep recurrent neural network used for clinical predictions. These models facilitate the transfer of learned features, making them valuable in scenarios with limited labelled data or restricted computational resources (Gupta et al., 2018; Alzubaidi et al., 2021; Fatima Ezzahra et al., 2023; Singh and Susan, 2023).

2.2.4 Generative Adversarial Networks (GANs)

GANs are ML models that learn the distribution of each class without explicitly separating them into distinct categories, unlike traditional techniques, such as DTs or SVMs. Instead, GANs generate new data points ($\boldsymbol{x}$) similar to the training data without considering the relationship between x and y, i.e., p(x|y). This type of DL model will be extensively explored in Chapter 4, with a focus on Basic Deep Convolutional Generative Adversarial Networks (DCGAN), Wasserstein GAN with Gradient Penalty (WGAN-GP), and conditional WGAN-GP.

2.2.5 Graph Neural Networks (GNNs): Definition and Overview

GNNs are a class of DL models designed to analyse and learn from data that are structured as graphs. GNNs have found wide application in tasks related to graph data, such as node classification, link prediction, and graph classification (Wei et al., 2020). They leverage graphs' detailed structural and feature information to perform these tasks effectively (Huang et al., 2024).

These capabilities make GNNs highly effective for diverse applications that involve graph-based data. The following are key aspects of GNNs (Huang et al., 2024):

1) Message-Passing Framework:
   - GNNs operate using a message-passing mechanism that repeatedly aggregates and updates information from the nodes' local neighbourhoods within a graph. This process enables GNNs to develop representations incorporating the graph data's structural and feature-related aspects (Huang et al., 2024).
   - The graph structure is typically represented by $G = \{V, A\}$, where V is the set of nodes, and A is the adjacency matrix. In this matrix, an element $A_{ij=1}$ signifies the presence of an edge between node i and node j. Each node i is also associated with a feature vector x_i^0 (Huang et al., 2024).
2) GNN Framework:
   - The GNN framework processes graph data by taking an initial set of node features $X = \{x_i^0 \mid i \in V\}$ and the adjacency matrix A as inputs. This input is then utilised to gather iteratively and pool information from the neighbours of each node. For

instance, the feature update for a node i in the l-th layer of message passing can be described in Equation (2.4):

$$x_i^l = f_2\left(pool\left\{f_1\left(x_j^{l-1}|\theta_1^l\right)|\, j \in N_i\right\}, x_i|\theta_2^l\right\}\right) \quad \textbf{(2.4)}$$

where pool$\{\cdot\}$ is a function that aggregates the features from neighbouring nodes N_i, and f_1 and f_2 are trainable functions parameterised by θ_1^l and θ_2^l (Huang et al., 2024).

2.2.5.1 Graph Convolutional Network (GCN) Applications

Graph Convolutional Networks (GCNs) are utilised across various domains, including computer vision, social networks, bioinformatics, recommendation systems, and traffic prediction. In computer vision, they model label correlation for multi-label images, capturing spatial dependencies and contextual information between pixels or image regions (Cao et al., 2022). Additionally, GNNs are particularly effective at representing data with inherent graph structures, such as social networks, protein interfaces, and images, by highlighting relationships and dependencies among entities (Tepe and Bilgin, 2022). For data that do not naturally form graphs, like audio signals, techniques that employ deep features from pre-trained models as node information are used to facilitate graph construction (Castro-Ospina et al., 2024). This versatility makes GCNs and GNNs powerful tools for handling structured and unstructured data in various classification tasks.

The search results do not mention using GCN architecture for Induction Motor (IM) fault classification. However, they highlight the effectiveness of GCNs across various fields, such as image classification, graph analysis, and human activity recognition. For image classification, Fei et al. (2023) introduced a novel end-to-end GNN that integrates local and global attention features for more accurate predictions (Fei et al., 2023). This model includes a CNN block for local feature learning and a GCN for global feature assimilation. In graph analysis, Zhang et al. (2022) described a hybrid accelerator for GNNs that utilises the Xilinx Versal ACAP architecture (Zhang et al., 2022a). This system enhances GNN inference by dividing graphs into subgraphs for efficient processing using programmable logic and AI engines. In human activity recognition, Lee et al. (2023) developed the Multimodal Two-stream GNN Framework for Efficient Point Cloud and Skeleton Data Alignment (MTGEA), which leverages the Spatial-Temporal Graph Convolutional Network (ST-GCN) architecture

to enhance recognition accuracy by focusing on skeletal features extracted from Kinect models (Lee and Kim, 2023).

However, Table 2.2 summarises the state-of-the-art GCN application in fault classification over the years starting from 2019 to 2024 with studies that used GCN in their research.

Table 2.2: Summary of State-of-the-Art GCN Applications in Fault Classification.

Application	Description	Outcomes
GCN-Based Compound Fault Diagnosis in Gearboxes (Zeng et al., 2024)**.**	Employs GCNs to analyse correlations among single faults in gearboxes to improve multi-label fault diagnosis. Each fault is treated as a label node, with GCN mapping features to enhance the classification of compound faults.	Enhances gearbox fault diagnosis accuracy using GCNs and self-attention to analyse correlations between single faults.
Multi-Scale Neural Transformation Graph (MNT-G) in Micro-Service System Fault Classification (Zhang et al., 2023b)**.**	This framework combines graph structure adjacency matrix learning with multi-scale neural transformation to analyse adjacency matrices and temporal features of system metrics separately. It uses a GCN to integrate spatio-temporal features for classifying faults in micro-services.	Demonstrates superior performance over traditional methods on the Sock Shop benchmark, with a macro-F1 score improvement of 7.16%.
Super Resolution - Graph Neural Network (SR-GNN) for Fault Classification and Location in Power Networks (Mo et al., 2023)**.**	Integrates super-resolution techniques with GNNs to efficiently classify and pinpoint faults in power distribution networks, focusing on cost reduction and accuracy.	Shows strong noise resistance and adaptability to various network conditions on the IEEE 37 Bus system, enhancing classification accuracy.
Temporal GCN for Transient Stability in Power Systems (Su et al., 2021)**.**	Develops a rapid-response Temporal Graph Convolutional Network (TGCN) that combines GCN for topology analysis with temporal convolution layers to quickly assess transient stability in power systems.	Exceeds performance of existing models in stability classification and predicting critical generator statuses on the IEEE 39 Bus system.
GCN for Testability Analysis in EDA (Ma et al., 2019)**.**	A specialised GCN model processes non-standard graph representations of logic circuits in Electronic Design Automation (EDA). This classifier is trained to identify optimal observation point candidates in netlists, targeting hard-to-detect nodes.	Matches fault coverage of commercial tools while reducing observation points by 11.00% and test patterns by 6.00%.

Although GCNs are not directly associated with Induction Motor (IM) fault classification in the available literature, their demonstrated benefits in various applications—such as enhanced accuracy, robustness, and the capability to integrate global information from node connections—suggest potential utility in diverse domains.

2.3 Two-Dimensional (2D) Signal Encoding Techniques

2.3.1 Gram Angular Field Signal Encoding (GAF)

Wang and Oates introduced the concept of GAF encoding, a method that transforms time series data into images (Wang and Oates, 2015). GAF's distinctive matrix construction maintains the integrity of the original data while capturing relationships between neighbouring elements. This methodology proves beneficial for CNN models, enabling automatic feature extraction and enhancing classification performance (Wang and Oates, 2015). The core concept behind converting time-series data into images using GAF involves creating a matrix based on polar coordinates. This matrix preserves the temporal relationships within the one-dimensional (1D) time-series signal, maintaining accurate temporal correlations compared to Cartesian coordinates. The process yields two types of GAF images: Gramian Angular Summation Field (GASF) and Gramian Angular Differential Field (GADF) (Toma et al., 2022a), which will be discussed further in Chapter 3.

2.3.2 Wavelet Transform (WT)

The Wavelet Transform (WT) provides an alternative to the Short-Time Fourier Transform (STFT) for non-stationary signal analysis. WT is advantageous because it can capture both temporal and spectral details. It offers adaptability across various frequencies and time-based resolutions (Nishat Toma et al., 2021; Yang et al., 2023b). Distinguishing WT from STFT, which uses fixed windows, WT utilises wavelet families with predefined shapes, including Haar, Symlets, and Daubechies. The mother wavelet function ψ(t) can be computed as in Equation (2.5) (Ahmed and Nandi, 2022):

$$\psi_{(s,\tau)}(t) = \frac{1}{\sqrt{s}}\,\psi\left(\frac{t-\tau}{s}\right) \tag{2.5}$$

In this context, s represents the scaling parameter, t corresponds to time, and τ denotes the transformation parameter. In the original wavelet, s = 1 and τ = 0.

Wavelets offer three essential transformations: the Continuous Wavelet Transform (CWT), Discrete Wavelet Transform (DWT), and Wavelet Packet Transform (WPT). From the literature on the IM vibration signal encoding, the CWT has been extensively utilised by researchers to create vibration image representations where the family of time-scale waveforms is derived by adjusting the position and scale of the mother wavelet (Kaji et al.,

2020). While there is no universal method for choosing the mother wavelet, it is common practice to visually inspect and select a suitable mother wavelet function based on shape matching (Kaji et al., 2020). CWT can be computed in the following Equation (2.6) (Ahmed and Nandi, 2022;):

$$CWT_{x(t)}(s,\tau) = \frac{1}{\sqrt{s}} \int x(t)\psi^*\left(\frac{t-\tau}{s}\right) dt \quad \textbf{(2.6)}$$

Here, ψ^* represents the complex conjugate of $\psi(t)$, which can be shifted using the translation parameter τ and scaled using the scale parameter s. These coefficients measure the degree of correlation between the waveform and the wavelet at different translations and scales. These coefficients are often displayed in a scalogram, illustrating the energy distribution across the coefficients.

2.4 Feature Extraction Domains in Signal Processing

Feature extraction operates within three primary domains: temporal, spectral, and time-frequency. These distinct domains serve as tools to capture distinctive aspects of signal behaviour. The section starts with Time and Frequency Domain (TFD) feature extraction and moves to the 2D time-frequency domain features. The feature extraction from vibration signals in the time domain is a crucial component of machinery fault diagnosis, enabling the early detection and continuous monitoring of machinery faults. This method entails computing diverse statistical parameters from the original vibration signal, which can subsequently be employed to assess the machinery's condition and detect potential problems. Various key parameters are utilised in VSA to extract vital information. These parameters include the Peak or Max value, which denotes the highest observed amplitude in the signal, and the Root Mean Square (RMS), which provides insights into signal magnitude. Skewness assesses distribution asymmetry, whereas Standard Deviation (std) quantifies average deviation from the mean. Kurtosis indicates distribution "tailedness," potentially identifying outliers or impulses. The Crest Factor, calculated as the peak amplitude-to-RMS ratio, reflects peak sharpness. Peak-to-peak measures the range between maximum and minimum values, whereas the Impulse Factor accentuates impulsive behaviours often linked to machinery faults. These parameters contribute to a comprehensive understanding of vibration signal characteristics, facilitating effective fault diagnosis and condition

monitoring (Liu and Weng, 2019; Pinedo-Sánchez et al., 2020; Jain and Bhosle, 2021; Narayan, 2021).

On the other hand, extracting features from the frequency domain can provide insights into the data's periodic components and harmonic structures. The frequency domain analysis of vibration signals involves examining the amplitude changes for different frequencies (Ahmed and Nandi, 2018). These features capture frequency-specific aspects of the signal and contribute to a better understanding of the vibration behaviour (Shi et al., 2020). Analysing the frequency domain of vibration signals is crucial for understanding periodic components and harmonic structures. Key features include Root Mean Square Frequency (RMSF), Centre Frequency (CF), Mean Square Frequency (MSF), Frequency Variance (FV), and Root Frequency Variance (RVF), providing insights into signal characteristics and power distribution (Shi et al., 2020). Standard harmonic features, such as Total Harmonic Distortion (THD), quantify frequency content (Tian et al., 2022; Granados-Lieberman et al., 2023). Signal-to-Noise Ratio (S/N) and Signal-to-Noise and Distortion Ratio (SINAD) assess signal quality, particularly in gearbox fault analysis (Kumar et al., 2022). Spectral analysis transforms signals from the time domain to the frequency domain, with the AR model being a popular choice. Various methods, like Yule-Walker and Burg's, compute AR coefficients, whereas the forward-backwards approach enhances classification, especially in machinery fault diagnosis (Hu and Zhang, 2019; Metwally et al., 2020). Spectral features like Peak Amplitude, Peak Frequency, and Band Power offer comprehensive insights into frequency characteristics (Ahmed and Nandi, 2018; Hu and Zhang, 2019; Shi et al., 2020; Tian et al., 2022; Djemili et al., 2023; Granados-Lieberman et al., 2023).

2.5 Multimodal Fusion Techniques

Sensors serve as the foundation for any machine's condition monitoring systems. The concept of smart sensors is currently an active area of research, where sensor data are linked to a data processing unit. Algorithms and DL techniques enable advanced interpretations of the collected sensor data. In the past, thermography was used as a secondary approach to condition monitoring. However, Infrared Thermography (IRT) is increasingly recognised as a qualified primary or direct approach for condition monitoring (Alvarado-Hernandez et al., 2022). The data used in the fusion process may come from different sources. This leads to

two types of sensor fusion: heterogeneous and homogeneous. In heterogeneous sensor fusion, data are gathered from various types of sensors, like vibration and current sensors.

On the other hand, homogeneous sensor fusion involves using data from the same kind of sensors, such as vibration sensors measuring the X, Y, and Z axes. Depending on the stage at which the information sources are combined, the fusion process can be classified into three levels: data-level fusion, feature-level fusion, and decision-level fusion (Debie et al., 2021). However, data processing can occur in multimodal fusion at three levels, as illustrated in Figure 2.3.

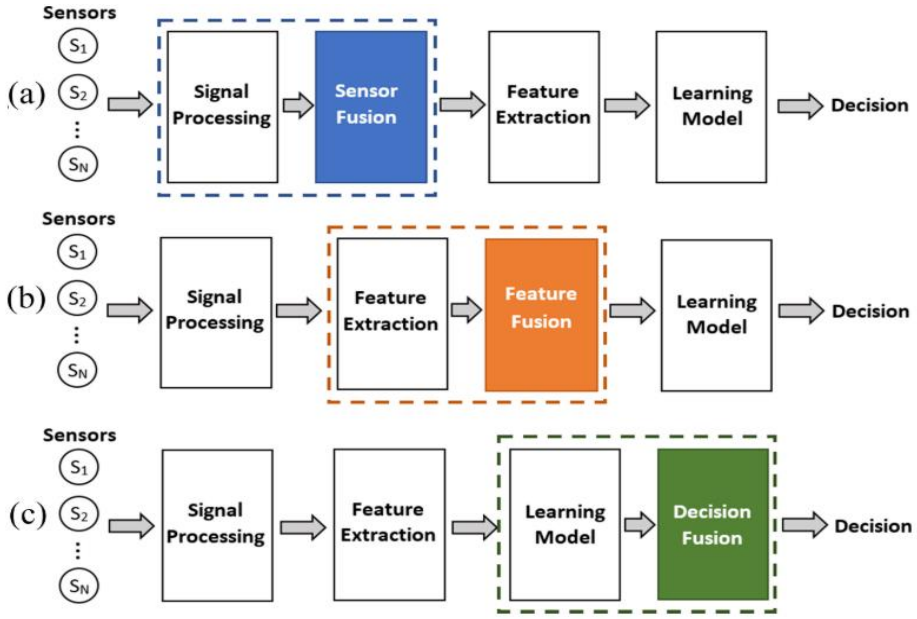


Figure 2.3: The Three Levels of Fusion (a) Sensor Fusion, (b) Sensor Data Represented by Feature Vectors, (c) Decision Fusion After the Classification Model (Debie et al., 2021).

Firstly, multiple sensors are employed at the sensor level fusion to capture raw data. Secondly, at the feature level, features are independently extracted from different sensors and then combined into a single feature vector, known as a fused vector. Thirdly, at the decision level, features are extracted independently and passed through separate classifiers to obtain individual decisions. The fusion process is then responsible for consolidating these decisions into a final classification decision. Furthermore, hybrid models can support multiple fusion levels (Debie et al., 2021) where, in fact, the effectiveness of sensor-level fusion and feature-level fusion strategies significantly relies on the characteristics of the data. Comparatively, the decision-level fusion strategy emerges as a more pragmatic choice among the three (Yang et al., 2022).

2.6 State of the Art, Research Gaps, and Directions in Each Research Theme

This section discusses the state of the art across five research themes: Multimodal Data Preprocessing Methodology, Artificial Thermal Image Creation, Customised Radial Load Assessment, Multimodal Systems Decision Fusion Approach, and Graph Convolutional Network (GCN) on Tabular Datasets.

2.6.1 Multimodal Data Preprocessing Methodology

Numerous researchers have sought to improve Induction Motor (IM) fault classification capabilities (Shao et al., 2020). Fault classification can be challenging, causing irrelevant rule generation for three main reasons: dataset size, noise, and overfitting problems (Packianather et al., 2019). According to the literature, Non-Invasive Inspection (NII) is a widely employed maintenance tool that monitors machines' health status. It helps to investigate the current health status without affecting or interrupting the operation. It can be divided into two categories according to the sensing approaches: contact and noncontact sensing (Alotaibi et al., 2021).

Contact-based Non-Invasive Inspection (CNI) does not require physical contact with the inspected parts; they must be attached to the machine system or body. For example, magnetic flux sensing, voltage sensing, machine current analysis, the vibration technique, and wear debris (Alotaibi et al., 2021). VSA is the traditional fault classification method (Jia et al., 2019), but different signals are used, such as current, acoustic, and temperature (Toma et al., 2021). However, in rotational-machine fault diagnosis using signals, signal preprocessing can be undertaken using time-domain, frequency-domain, or time-frequency domain analysis (Sinitsin et al., 2022). Time-domain analysis finds statistical parameters such as kurtosis, structural resonance, RMS, etc. On the other hand, frequency-domain analysis offers more benefits in signal analysis because it filters key frequency components, such as Fast Fourier Transform (FFT) and spectrum analysis (Sinitsin et al., 2022). Moreover, time-frequency analysis is used to empower frequency-domain analysis for volatile signals, for example, STFT, WT, and Empirical Mode Decomposition of the Hilbert-Huang Transform (HHT) (Nguyen et al., 2021).

Wavelet is considered the most recent and popular time-frequency analysis of the available methodologies, especially in bearing fault type recognition (Zhang et al., 2022d).

A recent study compared three CWT techniques on vibration signal encoding: Morse, Morlet, and Bump. All methods achieved more than 98.00% accuracy in bearing fault diagnosis. The PCA method was also used instead of wavelet and scored 10.00% lower than wavelet (Toma et al., 2021). Current signals were also used to classify IM faults using the GAF algorithm to generate 2D images. Subsequently, a two-layer deep CNN model was used for fault classification. The experiment showed that GAF images outperformed continuous wavelet images. Wavelet has fewer capabilities in generating significant patterns from current raw data relative to GAF due to the drawbacks of the current signal or low S/N (Toma et al., 2022a).

On the other hand, in Non-Contact-Based Non-Invasive Inspection (NCNI), sensors are not directly attached to the inspected part and nor is the machine system (e.g., Radio Frequency (RF), radar technology, ultrasonic sensing, camera-based imaging, Acoustic Emission (AE) sensing, thermographic sensing or Infrared (IR) and laser) (Alotaibi et al., 2021). Thermal images result in more accurate fault classification than vibration signals (Jia et al., 2019; McGhan and Feayherston, 2020; Shao et al., 2021). Hence, thermal image condition monitoring can achieve almost 100% accuracy when utilising CNN transfer learning capabilities (Choudhary et al., 2021; Khanjani and Ezoji, 2021).

Studies were not restricted to single-input modal creation. They also considered multimodal fault classification using the same sensing approach, combining current and vibration signals into one modal. Shao et al. (2020) used time-frequency distribution, continuous wavelet, and CNN capabilities in multi-signal fault diagnosis without utilising CNN transfer learning capabilities. The main contribution was using vibration and current signals in fault classification using CNN capabilities in image classification on time-frequency distribution images. The authors demonstrated that the multi-signal modal outperforms the single-signal modal input (Shao et al., 2020). A recent study proposed a multimodal neural network-based model using only vibration signals. Vibration signals are converted into time-frequency domain graphs using CWT and dot pattern graphs, creating two inputs for the CNN and two-level information fusion from the same signal. This model performed better than single-modal CNN (Ma et al., 2022).

In the most recent research, the vibration signal captured using CNI acquisition tools is considered the most popular input for bearing fault diagnosis. It contains valuable information about each fault type, giving a good prescription for the machine's health. At the same time, it has a high S/N and requires extra preprocessing and strict sensor installation requirements (Jia et al., 2019). The most common preprocessing methodology is converting a 1D signal into 2D time-frequency graphs using CWT, where GAF encoding was rarely used. Consequently, NCNI data acquisition techniques were introduced, primarily utilising IRT due to its non-contact nature, high accuracy, and reduced requirement for signal preprocessing knowledge. Nonetheless, many studies have concluded that thermal images offer an excellent substitute for vibration signals, resulting in higher fault classification accuracy. However, thermal image fault classification has some known limitations, including deviation in the region of interest due to camera misalignment and blurred images. This can result in fault misclassification, which has not been discussed in the previous literature (Li et al., 2021; Shao et al., 2021).

On the other hand, multimodal DL is a new paradigm in AI that requires further attention and exploration. Hence, relying on a single input using a single sensing approach often fails to extract the full knowledge from data, especially in abnormal operation conditions (Jia et al., 2019). However, few research studies have explored Multimodal DL using similar data types, such as numerical signals or signals captured using CNI to demonstrate its efficiency. Consequently, there is a need to research Multimodal further using different sensors, which will be explored in Chapter 3.

2.6.2 Artificial Thermal Image Creation

DCGAN has demonstrated its efficiency (Du et al., 2019) for image generation in solving imbalanced datasets in the chemical industry's fault diagnosis field. It was also used by He et al. (2021) for axial piston pump bearing fault diagnosis to mitigate data availability and missing fault labelling challenges. DCGAN has also been used in Induction Motor (IM) fault classification using the Case Western Reserve University (CWRU) bearing centre dataset, a well-regarded dataset in which CWT images were synthesised (Zhong et al., 2023). A previous paper focused on Wasserstein Generative Adversarial Network (WGAN) usage

in thermal images, with fault creation in IMs being used to increase fault samples, namely, IRF, ORF, and Normal (fault-free) or Healthy condition (Ma et al., 2023; Shao et al., 2023).

It is apparent from the literature that researchers have sought to generate highly trustworthy data to enhance training performance on limited fault types or generate look-alike vibration signals without focusing on thermal image fault creation (Ma et al., 2023). A recent paper explored the generation of thermal images on three conditions to enhance fault classification accuracy using a single input model (Shao et al., 2023).

..

Bearing VSA is the traditional means of fault classification, where raw vibration signals are rarely used; hence, vibration signals need to be pre-processed using either time-domain analysis or frequency-domain analysis (Sinitsin et al., 2022). On the other hand, thermal images result in more accurate fault classification with up to 100% accuracy whilst requiring less preprocessing time than vibration signal fault classification, as demonstrated by Choudhary et al. (2021) and Khanjani and Ezoji (2021). Thermal images are more stable than vibration signals; hence, they are less sensitive to speed fluctuation scenarios, making them more efficient (Shao et al., 2023).

However, thermal images have certain known drawbacks. For instance, the installation cost of cameras and the potential for camera misalignment can result in inaccurate recognition (Gangsar and Tiwari, 2020). Furthermore, the limited availability of data and the imbalanced distribution of thermal images across specific or all health conditions can significantly affect the performance of condition monitoring systems (Niu et al., 2020). To mitigate these limitations, various oversampling techniques have been employed to generate additional samples from the minority classes. One such technique is the Synthetic Minority Oversampling Technique (SMOTE), which uses interpolation based on nearest neighbours. Another approach is the Adaptive Synthetic Sampling Technique (ADASYN) (Liu et al., 2021). However, it is essential to note that oversampling techniques can be susceptible to overfitting and noise creation, especially when dealing with high-dimensional and sparse data. These techniques may also generate samples that are more similar to the majority class rather than to the desired class (Engelmann and Lessmann, 2021).

Moreover, while improving classification accuracy is a common approach, it may not be effective when the degree of imbalance is high unless more data are added to the training model (Han et al., 2020). Expanding image data by including noise and local blur can be seen as an artificial preprocessing technique. However, it is essential to note that these methods may not adequately capture the diversity in the original samples and so can potentially hinder fault recognition (Fan et al., 2022). In contrast, GANs offer a new and promising approach to sample generation. GANs provide a framework for learning complex features from high-dimensional, imbalanced, and small dataset distributions, and they have been widely utilised in fault diagnosis applications (Han et al., 2020; Engelmann and Lessmann, 2021; Liu et al., 2021; Fan et al., 2022).

The selection of an appropriate GAN for generating artificial images of thermal IM health conditions is critical to the current research. Previous studies in this field have been limited, with only a few papers released (Wu et al., 2019). Commonly used GAN models in fault diagnoses include DCGAN, Auxiliary Classifier GAN (ACGAN), Wasserstein GAN (WGAN), and variational auto-encoding GAN. However, it has been observed that the quality of data generated by the original GAN and improved DCGAN is still relatively low (Han et al., 2020; Fan et al., 2022). Meanwhile, the WGAN-GP has demonstrated improved training stability, mode collapse prevention, and the generation of high-quality images (Gulrajani et al., 2017; Pan et al., 2019; Gao et al., 2020). WGAN-GP has also proven its effectiveness in fault sample generation (Wang et al., 2021a) and in supplementing low-dimensional fault data (Zhong et al., 2023). Wasserstein distance in WGAN provides a more meaningful measure of the difference between probability distributions and leads to better convergence by avoiding vanishing gradients (Arjovsky et al., 2017a).

Additionally, the training process in WGAN-GP does not require a careful balance between the generator and discriminator (Arjovsky et al., 2017b). WGAN-GP has also been employed in the imbalance fault classification of bearings, overcoming convergence issues observed in the original GAN structures. WGAN-GP demonstrated faster convergence within 400 iterations and improved model performance compared to the original WGAN due to the gradient penalty (GP) (Han et al., 2020).

WGAN-GP was utilised to generate additional vibration signal spectra for imbalanced bearing fault classification problems, demonstrating improved convergence and

faster training speed with the GP (Shao et al., 2023). Chang et al. (2022) examined GANs and CNNs for imbalanced vibration signal datasets in IMs, confirming their efficiency. However, there was still room for improvement in utilising labelled data for IM fault classification because models trained on the generated data differed in accuracy compared to real data (Chang et al., 2022). Meanwhile, Ma et al. (2023) focused on WGAN-GP to create vibration signals in the rotor-bearing system, showing high-quality signal generation and increased diagnostic accuracy. Shao et al. (2023) generated thermal images for various health conditions in rotating machinery, achieving good results but suggesting the incorporation of label information in GANs training.

The scarcity of IM datasets collected under diverse health conditions poses challenges due to data availability, confidentiality, and time constraints. While GANs have been used to generate additional tabular vibration data for condition monitoring, utilising GANs for thermal image synthesis in IM condition monitoring is a promising but relatively new research area. Hence, GANs are commonly employed to generate supplementary tabular vibration data. At the same time, thermal image condition monitoring offers more accurate results with minimal preprocessing steps due to its lower sensitivity to noise, which will be explored in Chapter 4.

2.6.3 Customised Radial Load Assessment

Bearing fault diagnosis is recognised as a pattern recognition challenge, emphasising the importance of dominant eigenvectors for fault features. Accurate feature identification is critical to enhance the reliability of fault detection and diagnosis systems. Toma et al. (2022b) used Wavelet Scattering Transform (WST)-based features, whereas Nayana and Geethanjali (2020) employed statistical TFD features to contribute to IM fault classification. Other techniques include time-domain features from current signals (Toma et al., 2020), homogeneity and kurtosis from electrical current during motor startup (Martinez-Herrera et al., 2022), and the use of CWT for fault diagnosis (Yuan et al., 2020). This method, tested on the CWRU bearing centre dataset and Machinery Failure Prevention Technology (MFPT) bearing datasets, demonstrated superior diagnostic accuracy and stability.

The approach increasingly leans towards treating it as a pattern recognition challenge in bearing fault diagnosis, relying on dominant eigenvectors to represent fault features, enabling more reliable detection and categorisation of bearing faults (Nemani et al., 2022).

To determine the precise location and intensity of a bearing defect, various VSA techniques are available, broadly categorised into the time domain, frequency domain, and time-frequency domain analyses (Jain and Bhosle, 2022). Feature extraction in ML for bearing fault diagnosis is pivotal, particularly in analysing vibration signals, resulting in a multi-domain feature set. The goal is often to derive features with strong discriminatory capabilities (Shi et al., 2020). Time–domain features assume a stationary signal, but signals frequently exhibit changes in statistical properties over time (Sayyad et al., 2021). However, obtaining suitable features may require a long period of recorded signals, making it expensive, time-consuming, or even impossible for certain fault types or with complex equipment (Resendiz-Ochoa et al., 2018). RMS and kurtosis are commonly used in the time domain, especially kurtosis, which is highly effective in early fault detection (Pinedo-Sánchez et al., 2020).

In contrast, frequency-domain features require more significant computational effort than their time-domain counterparts and operate under the assumption of a wide-sense stochastic signal (Narayan, 2021). FFT, while powerful in stationary conditions, has limitations when applied to non-stationary data, that is, signals that change over time or exhibit variations in their frequency content. In such cases, FFT's assumption of a constant frequency spectrum over the entire signal duration does not hold. Alternative time–frequency signal processing techniques have been developed to address this limitation (Resendiz-Ochoa et al., 2018).

Nevertheless, transitioning to time-frequency domain analysis, which combines time and frequency information to understand the signal's frequency band over a specific time interval (He et al., 2010), offers a localised signal analysis by considering smaller time segments. This approach proves valuable for non-stationary signals where the frequency content changes over time (Zhang et al., 2021a). The CWT is a powerful tool for analysing non-linear and non-stationary data in the time-frequency domain. It outperforms other techniques, such as the STFT, Gabor transform, WT, and Wigner-Ville transform, effectively addressing the limitations of the FFT in dealing with such data (Toma et al., 2021; Guo et al., 2022). The WT can analyse specific regions within a more prominent signal without sacrificing spectral details, revealing concealed facets undetected by alternative methods (Kaji et al., 2020). This enables the distinctly different analysis of both frequency and time domains, breaking down signals into various frequency components and analysing each

element with the time domain corresponding to its specific scale (Ozaltin and Yeniay, 2023). It is crucial, however, to carefully consider or create the most suitable wavelet foundation (Guo et al., 2022). Pinedo-Sánchez et al. (2020) explored the effectiveness of three prevalent mother wavelet functions in conjunction with pre-trained CNNs on the automatic classification of an electrocardiogram (ECG) dataset. Specifically, the study used AlexNet and SqueezeNet, revealing that Amo (often called a Morlet wavelet) and Morse wavelet functions enhanced class recognition with AlexNet. In contrast, the Bump wavelet function demonstrated superior classification accuracy with pre-trained SqueezeNet (Pinedo-Sánchez et al., 2020).

Beyond CWT, techniques such as wavelet entropy, wavelet packet energy entropy, and Wavelet Singular Entropy (WSE) have also been utilised. Wavelet entropy, combining WT and Shannon entropy, captures complexity and information content within signals at different scales or frequencies. In the CWT realm, this approach is valuable for analysing time-frequency representations and revealing patterns associated with structural damage (Li et al., 2019a; Guo et al., 2022). Examined on IM bearings, selecting the optimal contentious transform wavelet (Guo et al., 2022) and indicating the complexity of the analysed transient signal in the time-frequency domain (He et al., 2010) makes it possible to distinguish between transients with different complexities intuitively and quantitatively. Wavelet energy, measuring the energy distribution across different scales in the WT of a signal, was used to track changes in energy over time for fault localisation and categorisation (Jayamaha et al., 2019). This information is then employed to create a set of features for classification, followed by Artificial Neural Network (ANN) training to categorise these features.

Researchers have increasingly focused on the fault detection and diagnosis systems of various operational parameters of bearings, such as friction torque, radial internal clearance, and slippage. In a notable study, Wu et al. (2023a) investigated the friction torque behaviours of thrust ball bearings with self-driven textured guiding surfaces. This study sought to facilitate the starved lubrication conditions often encountered in rolling bearings by introducing innovative textures on the guiding surfaces. Notably, the results indicated that implementing a gradient groove texture could significantly reduce the friction torque of bearings. This texture facilitates a one-way self-driving function for liquid droplets, highlighting its potential for practical applications in bearing design (Wu et al., 2023a).

Meanwhile, Ambrożkiewicz et al. (2023) explored the effect of various surface textures on thrust ball bearings' vibration and friction torque behaviours, including dimples, grooves, and gradient grooves. The study found that the gradient texture effectively reduces vibration acceleration and friction torque (Ambrożkiewicz et al., 2023). Furthermore, research on the slipping behaviour of H7006C angular contact ball bearings under operational conditions demonstrated similar benefits from this texture design in reducing vibration and friction torque, thus enhancing bearing performance (Yang et al., 2023a).

However, there remains a notable gap in our understanding of the influence of varying loads on the manifestation of faults (Zhang et al., 2022b). Previous research has delved into areas such as estimating the remaining useful life from run-to-failure datasets (Zhang et al., 2022b). Nevertheless, the domain of load's impact on faults remains relatively unexplored. Radial impact was discussed by Jain and Bhosle (2021), where traditional statistical indicators were used to study the effects of IRF and ORF in bearings under different loads. The MFPT bearing dataset was utilised to propose combinations of indicators, including Kurtosis × RMS, Kurtosis × Peak, and RMS × Peak for early fault detection, including IRF and ORF. A similar analysis was conducted on the CWRU dataset, thoroughly investigating various traditional and new vibration indicators for detecting bearing defects and monitoring their progression (Jain and Bhosle, 2022).

In recent years, detecting faults in IMs has attracted considerable attention, given their crucial role in various industries. As a result, there has been a concerted effort to develop reliable and cost-effective methods for diagnosing faults in IMs. The early detection of potential failures is of paramount importance because it can prevent significant damage to machinery (Nayana and Geethanjali, 2020; Toma et al., 2020; Yuan et al., 2020; Martinez-Herrera et al., 2022; Toma et al., 2022b). Despite the recognised significance of feature extraction and selection within intelligent diagnosis systems, relatively little attention has been paid to assessing load impact in the literature (Han et al., 2021b; Zhang et al., 2022b). A notable gap has emerged in intelligent diagnosis systems where feature extraction and selection are crucial, especially in evaluating load impact (Ahmed and Nandi, 2018). Extensive research has explored fault classification under varying loads, but the nuanced effects of load variations on the intrinsic nature of faults have not been thoroughly addressed.

Thus, Chapter 4 introduces the proposed novel Customised Load Adaptive Framework (CLAF) in detail.

2.6.4 Multimodal Systems Decision Fusion Approach

The field of fault detection in manufacturing systems has witnessed remarkable advances, particularly in analysing vibration signals for condition monitoring and fault detection. This dynamic area of research, focusing primarily on identifying faults in rolling element bearings amid substantial noise, highlights the critical role of feature selection in ensuring classification accuracy. Researchers have been actively developing various methods to extract robust statistical data from vibration signals, using techniques ranging from time, frequency, and spectral feature extraction to AR models. Integrating ML and the advanced capabilities of deep CNNs has facilitated a significant advance in this quest. Additionally, innovative classification fusion methods, including sensor, feature, and decision fusion, have been implemented to enhance accuracy. Techniques including GAF and CWT are utilised for deeper signal analysis. At the same time, transfer learning approaches with architectures such as AlexNet and ResNet are applied for more precise fault diagnosis. Despite these achievements, the field faces complex challenges, emphasising the need for ongoing exploration and innovation.

Lorenz et al. (2022) delved into various techniques for fault detection in manufacturing systems, employing vibration data analysis. They discuss time-domain features such as RMS, variance, and kurtosis alongside frequency-domain features such as spectral attributes. Time-domain features have shown particular efficacy for early fault detection (Pinedo-Sánchez et al., 2020), although real-world signals often exhibit temporal variations (Sayyad et al., 2021). The challenge of acquiring suitable features has been noted because it can be laborious and sometimes impractical for specific faults or in the context of complex machinery (Resendiz-Ochoa et al., 2018). The study investigates fault detection in rotating element bearings, emphasising time-domain features (median, peak-to-peak value, and mean) and frequency-domain features (spectral centroid and kurtosis). Considerable emphasis is placed on using ML classifiers, especially the quadratic SVM from MATLAB, for the multi-class classification of machinery health states. This method has demonstrated its effectiveness in accurately identifying faults (Lupea and Lupea, 2022). Advances in ML

have significantly increased model accuracy, primarily through ensemble learning, which combines multiple models to enhance overall prediction effectiveness (Jose et al., 2022).

The research described in the document introduces a unique feature extraction approach known as One-Dimensional Ternary Patterns (1D-TP) for bearing fault detection using vibration signals. This approach extracts statistical measures from these signals in both the time and frequency domains. For classification purposes, the study employs a variety of ML classifiers, including Random Forests, kNN, SVM, BayesNet, and ANNs. This methodology effectively pinpoints faults in bearings, highlighting the potential of 1D-TP and these classifiers in the field (Kuncan et al., 2020). Additionally, the paper explains the workings of SVM, a state-of-the-art algorithm primarily used for categorisation based on the principle of margin calculation. It effectively separates data groups by drawing a line between them, optimising margins to reduce the difference with labelled classes, thereby minimising classification errors.

Furthermore, the study touches upon DTs, which consist of nodes and branches used primarily for classification purposes. This method sorts attributes based on their values, grouping them accordingly, where each node represents a category attribute and each branch a specific value of that node (Kadam et al., 2021). Despite these notable contributions to fault diagnosis and ML, the thesis identifies a gap in load-dependent fault condition monitoring. It underscores the need for further exploration of advanced deep-learning techniques.

Moreover, there has been a significant focus on spectral feature extraction using AR models in bearing fault classification, as detailed in the existing research. AR models extract vital features from vibration signals, which have proven effective in identifying various operational states. Research indicates that AR-derived features are comparably compelling, achieving classification accuracies similar to those obtained with power spectral features in areas such as emotion recognition and signal processing (Ganapathy et al., 2014 . This has led to a growing interest in the Forward-Backward Autoregressive (FBAR) model, a variation of the AR model, particularly in feature extraction for diverse signal-processing applications (Vaibhaw et al., 2020). Therefore, using AR models for feature extraction is emerging as a promising approach to enhance the classification of bearing faults based on vibration signals. This exploration suggests that AR models could complement traditional TFD features, warranting further investigation.

The CNN, a potent model in DL (Nishat Toma et al., 2021), is employed in architectures such as AlexNet and ResNet), notably for diagnosing bearing faults. Utilising these CNNs through transfer learning has demonstrated significant efficacy in various applications, as highlighted by Lu et al. (2020). These collective findings emphasise the promise of pre-trained CNNs and transfer learning techniques in bearing fault classification using vibration signals, offering a compelling approach to automated fault diagnosis in machinery. AlexNet (Lu et al., 2020; Asutkar and Tallur, 2023) combined with transfer learning has effectively classified casting surface defects. Its efficacy lies in utilising pre-trained models for specific tasks, balancing simplicity with reliable performance (Thalagala and Walgampaya, 2021). The ResNet family has also shown promise within the bearing fault diagnosis domain. The principal advantage of ResNet-18 is its intricate architecture and residual connections, which enhance training efficiency and task performance, especially in complex scenarios (Chang et al., 2023; Wu et al., 2023b).

In contrast, AlexNet, known for its straightforward and dependable framework, is better suited to simpler classification tasks, making it a practical choice in environments with limited computational capacity (Ramzan et al., 2020). However, the advanced design of ResNet, with its deeper layers and residual blocks, requires greater computational power compared to AlexNet, presenting a need to balance efficiency and computational demands in CNN applications (Kadam et al., 2021; Thalagala and Walgampaya, 2021). Thus, both ResNet and AlexNet represent two well-established deep-learning approaches with the potential for further advances in condition monitoring.

Additionally, 2D signal encoding techniques, such as GAF and CWT, have significantly improved the feature extraction capabilities of CNNs. GAF, in particular, has shown promise in high-precision fault signal classification (Zhang et al., 2023a) and mental well-being state classification (Woodward et al., 2024). Toma et al. (2022a) demonstrated the efficacy of converting current signals into 2D images using GAF, followed by CNN classification, in bearing fault classification. On the other hand, CWT signal encoding has been highly effective, especially when paired with Ensemble Empirical Mode Decomposition (EEMD) for intrinsic mode function selection. This combination has achieved more than 99.00% accuracy in fault detection in some instances (Nishat Toma et al., 2021) and has been successful in early fault detection (Kaji et al., 2020). The synergy of

CWT with multiscale feature fusion and enhanced channel attention mechanisms has also been investigated to further refine feature extraction from vibration signals (Xiao et al., 2021b). In 2023, an innovative approach that combines the GASF with CWT was introduced for intelligent fault diagnosis in wind turbine gearboxes. This method leverages GAF and CWT techniques to improve fault detection accuracy (Yang et al., 2023b). However, it is essential to note that, based on current research trends and to the best of the author's knowledge, GAF and, notably, GADF are not yet widely recognised as established techniques for vibration signal encoding and have seen limited exploration. In contrast, CWT signal encoding is much more prevalent. This disparity indicates a significant opportunity for further research and development regarding 2D signal encoding techniques, particularly in exploring the potential of GAF and GADF for VSA.

Researchers have developed promising fusion techniques in rotating machinery fault diagnosis. One notable approach is the multi-sensor fusion technique, which has demonstrated enhanced fault classification accuracy and quicker convergence than single-source sensor data. A 2023 study introduced an innovative bearing fault classification method using multi-sensor fusion technology combined with an advanced binary one-dimensional ternary pattern (EB-1D-TP) encoding algorithm; this combination achieved classification rates over 98.00%, demonstrating its potential for broader industrial applications and integration into Industry 4.0 (Pan et al., 2023). Cinar (2022) highlighted data-level sensor fusion, achieving up to 100% validation accuracy. Inspired by standard CNN pooling methods, this technique merges sensor channels using overlaid spectrogram images to select the highest spectral power at each frequency and time point for improved classification with a pre-trained SqueezeNet model (Cinar, 2022).

Kullu and Cinar (2022) also utilised raw TFD data from two sensor types, transforming them into time-frequency images via the STFT. The time-frequency images were combined with raw time series data and used in a DL model for fault detection, demonstrating promising results in terms of fault classification on datasets from Paderborn University and Eskisehir Osmangazi University (Kullu and Cinar, 2022). Furthermore, a method employing current, vibration, and torque signals applied STFT to each, creating spectrograms that were combined into a single image for analysis with pre-trained SqueezeNet, demonstrating efficacy in fault diagnosis (Cinar, 2022). While the concept of

2D vibration signal encoding has previously been explored, there remains significant potential for advancement, particularly in identifying more efficient input combinations for the final condition monitoring system. This ongoing exploration is vital to develop tailored solutions to specific monitoring challenges.

Recent advances in feature fusion techniques for rotating machinery fault diagnosis have shown considerable promise, and the field remains ripe for innovative approaches. In 2022, a notable development was a multi-sensor feature fusion approach for rolling bearing fault diagnosis. This technique enhances accuracy by amalgamating data from various sensors. It incorporates Variational Mode Decomposition preprocessing and a deep autoencoder network, outperforming alternative methods which rely on single-sensor data. Toma et al. (2022b) introduced a feature fusion method for bearing fault classification in IMs. This method utilised the WST to extract features from current signal data, achieving 99.00% accuracy when combined with ensemble ML algorithms.

Further exploration in 2021 saw the integration of CNN knowledge transfer with time-frequency domain features in the Feature Fusion Convolutional Neural Network-Support Vector Machines (FFCNN-SVM) method. Multi-Level Features Fusion Network (MLFNet), an innovative CNN, also demonstrated its ability to extract and fuse multi-scale features from noisy vibration signals, attaining an exceptional 99.75% recognition accuracy (Ye and Yu, 2022). The potential of combining time-domain and frequency-domain data was also highlighted using the GAF method and processed by the ECA-ConvMixer model for motor fault diagnosis (Xie et al., 2023).. Decision fusion multi-dimensional feature extraction techniques have also been employed to create comprehensive feature vectors. These vectors are amalgamated using algorithms, such as the Yager algorithm, for extensive fault pattern recognition (Li et al., 2019b). In 2022, a fuzzy decision fusion strategy was developed, integrating outputs from CNN models trained on datasets processed through various transforms (Yang et al., 2022). Wang et al. (2023b) presented an innovative algorithm for industrial motor bearing fault diagnosis, which integrates multi-source information using a noise reduction autoencoder and bidirectional Long Short-Term Memory (LSTM) networks. While these fusion techniques have demonstrated promising results in numerous studies, there remains significant potential for further research. More specifically, there is an opportunity to tailor and customise fusion techniques for load-dependent condition

monitoring on specialised datasets. This offers a pathway for more nuanced and compelling IM bearings condition-monitoring solutions, which will be explored in Chapter 6.

2.6.5 Graph Convolutional Networks (GCNs) on a Tabular Dataset Application

GCNs are specialised neural networks designed to handle data structured in graphs, where the data points are nodes interconnected by edges. GNNs excel at managing the intricate relationships and patterns present within graphs, making them ideal for various applications, including drug discovery, fraud detection, and recommendation systems. GNNs can predict and analyse the interconnections between data points by applying DL techniques to graph data. They typically employ message-passing mechanisms to incorporate information about nodes and their adjacent nodes, enabling the network to identify patterns and make informed predictions based on the graph's structure. In recent years, GNNs have attracted considerable attention for their ability to represent complex relationships and patterns, which conventional neural networks may find challenging to handle (Li et al., 2023b).

Using GNNs for classification offers several advantages: GNNs are adept at capturing complex interdependencies between entities, thus providing significant benefits in structured data classification (Du et al., 2023; Lee et al., 2023). They frequently outperform traditional ML methods and CNNs in specific tasks, such as classifying colorectal histopathological images (Tepe and Bilgin, 2022). Additionally, GNNs can utilise constructed graphs in a self-supervised manner, facilitating knowledge transfer to pairwise neural networks for practical applications (Du et al., 2023).

CNNs are designed for grid-like data such as images and use convolutional layers with filters to learn features. In contrast, GNNs are for graph-structured data and employ message passing to incorporate information about nodes and their connections. The critical difference between CNNs and GNNs lies in the data they handle and the mechanisms they use to learn features. CNNs are commonly used in computer vision tasks such as image recognition, whereas GNNs are better suited for applications involving complex data relationships (Lin et al., 2021).

Applying 1D-CNNs in fault detection across various industries showcases their efficiency in processing time-series data and extracting meaningful features for diagnosis. For instance, Wang et al. (2024) combined features from multiple sensors using a 1D-CNN

to predict bearing faults in aircraft engines, while Abdeljaber et al. (2022b) demonstrated their effectiveness in detecting structural damage (Chen et al., 2022b). 1D-CNNs are particularly effective for detecting sequential patterns because they apply convolution operations across one dimension, which is well-suited for time series or any sequential data (Camacho-Bello et al., 2022; Zhang et al., 2023c; Ahmadzadeh et al., 2024).

K-Nearest Neighbour graphs (k-NNGs) have proven useful in diverse applications, including healthcare diagnostics and machinery fault detection. Chandaliya et al. (2023) used k-NNGs with GNNs to classify cough sounds for disease detection, revealing complex relationships even with limited labelled data (Chandaliya et al., 2023). Similarly, Rangel-Rodriguez et al. (2023) applied the kNN method to generate graphs from vibration signals for crack detection in rotating machinery, with these graphs serving as inputs for ML algorithms (Rangel-Rodriguez et al., 2023). Earlier, Wang et al. (2021b) employed the kNN method to create graphs from vibration signals for fault diagnosis in rotating machinery, using these graphs as input for a GCN for fault classification (Wang et al., 2021b).

GNNs and k-NNGs are complementary techniques for analysing and processing graph data. K-NNGs are crucial in constructing graphs from data points, while GNNs excel in identifying patterns and relationships within the graph data. For example, the NN-Descent algorithm efficiently handles the construction of k-NNGs by iteratively refining neighbour connections (Dong et al., 2011). While the GCN architecture is not explicitly mentioned in the search results for Induction Motor (IM) fault classification, it is evident that GCNs have shown promising results in various domains. GCNs have been effectively applied in diverse areas, enhancing fault classification and predictive maintenance. These applications include Multi-Scale Neural Transformation Graph Method (MNT-G) frameworks in micro-service systems, which improve classification accuracy (Zhang et al., 2023b), high-performance GCNs in electronic design for better testability analysis, Super Resolution - Graph Neural Network (SR-GNN) in power networks for precise fault location (Mo et al., 2023), temporal GCNs for rapid transient stability assessment in power systems (Su et al., 2021), and compound fault diagnosis in gearboxes using GCN-based models (Zeng et al., 2024). These practical implementations demonstrate the versatility and efficacy of GCNs in handling complex data and improving fault detection across various industries.

The general advantages of GCNs across various domains include improved accuracy and robustness and the ability to aggregate global information through the interconnected relationships of different nodes. Consequently, it is feasible to develop more robust and adaptable models for various graph-based applications by integrating these approaches. While the potential of GCNs for fault classification in IMs is theoretically plausible due to their ability to model complex graph structures, the current research does not explicitly discuss the specific application of GCNs in this context. Therefore, further research focusing on applying GCNs in the fault classification of IMs would be necessary to validate this assumption.

While GNNs have not been widely applied to IM fault detection, their ability to process graph-structured data is seen as valuable for analysing complex systems like IMs. Complex relationships within motor data can be captured by GNNs, aiding in anomaly detection and predictive maintenance. Similarly, 1D-CNNs have been shown to effectively analyse sequential motor signals to detect faults under various conditions. These neural networks, including GNNs and 1D-CNNs, have been recognised for improving fault detection accuracy in IMs (Skowron et al., 2020; Rahmawan et al., 2023)

In this study, tabular data will be represented as graphs for VSA using the kNN method, where nodes represent time points, and edges represent signal similarities. The Taguchi method will be employed to optimise key factors affecting performance in this new approach. To address the gaps identified in GNN performance, the 1D-CNN will be explored as a tentative candidate for a hybrid methodology, providing complementary strengths in fault classification.

2.7 Summary

This chapter provides an in-depth overview of IMs, the primary focus of the current research. It delves into the broader realm of AI, encompassing ML and DL. It outlines a variety of AI algorithms and networks pertinent to IM condition monitoring. This includes supervised, unsupervised, and RL techniques. The chapter further discusses DL architectures, particularly CNNs, CNN transfer learning, GANs, and GNNs. Each section explores the theoretical aspects and discusses the practical applications of these technologies.

Additionally, the chapter introduces signal encoding techniques such as the GAF and CWT. It also reviews multimodal fusion approaches. The exploration extends to encoding

tabular datasets using GNNs and the analysis of feature extraction in signal processing across the time, frequency, and time-frequency domains, illustrating applications in each context.

The chapter concludes by summarising the current state of the art and identifying research gaps within five key themes: Multimodal Data Preprocessing, Artificial Thermal Image Creation, Customised Radial Load Assessment, the Decision Fusion Approach in Multimodal Systems, and a GCN on Tabular Datasets. It highlights the gaps and outlines future research directions for each theme, aligning with the thesis' stated aim to enhance fault classification in IMs significantly. This enhancement seeks to improve decision-making accuracy and augment algorithms' intelligence within IM condition monitoring systems, directly supporting the thesis' objectives. The proceeding chapters will deal with the five themes mentioned above in that order.

Chapter 3: Novel Preprocessing of Multimodal Condition Monitoring Data for Classifying Induction Motor Faults Using Deep Learning Methods

3.1 The Impact of Data Representation on the Performance of Machine Learning Models in Fault Classification

The choice of data representation can significantly impact the performance of Machine Learning (ML) models in fault classification. The selection of relevant features, normalisation, data augmentation, dimensionality reduction, graph-based representations, knowledge graphs, and hybrid approaches can all contribute to improved model performance. The impact of data representation on the performance of ML models in fault classification is significant. The choice of data representation can affect the models' accuracy, speed, and generalisability. Some of the critical aspects of data representation that can influence the performance of ML models in fault classification include the following:

1) Feature selection: The choice of features used to represent the data can significantly impact the model's performance. Relevant features should be selected to ensure that the model captures the most essential information related to the fault. Inversive features sometimes lead to overfitting or reduced performance (Kareem and Hur, 2022). This will be addressed in this chapter.
2) Data normalisation: Normalising the data can help improve the performance of ML models by ensuring that all features are on a similar scale. This can prevent some features from dominating others and improve the model's accuracy (Jang and Cho, 2021).
3) Data augmentation: Augmenting the data can help improve ML models' performance by increasing the training set's size. This can help the model learn more robust features and reduce overfitting (Yousuf et al., 2024).
4) Dimensionality reduction: Reducing the dimensionality of the data can help improve the performance of ML models by reducing the noise and irrelevant features in the data. Techniques like Principal Component Analysis (PCA) and t-distributed Stochastic Neighbour Embedding (t-SNE) can reduce dimensionality (Wodecki and Michalak, 2021).
5) Graph-based representations: Representing the data as a graph can help improve the performance of ML models by capturing the relationships and interactions between different data entities. Graph-based representations can be used for similarity search,

clustering, and other data mining tasks (Jang and Cho, 2021). This will be explored in Chapter 7.

6) Knowledge graphs: Integrating Knowledge Graphs (KGs) into ML models can help improve their performance by incorporating domain-invariant knowledge. This can aid in solving specific tasks and handling domain shifts, such as variations in machine operation conditions (Radtke et al., 2023).
7) Domain-specific ontologies: Using domain-specific ontologies expressed in the Resource Description Framework (RDF) and Web Ontology Language (OWL) can enhance building analytics through multi-domain knowledge integration and facilitate numerical representation (Delgoshaei et al., 2022).
8) Hybrid approaches: Combining different techniques, such as physics-based performance models with Deep Learning (DL) algorithms, can help improve the performance of ML models in fault diagnostics (Chao et al., 2019).

3.2 Proposed Methodology

In condition monitoring, integrating diverse sensor data is a cornerstone for advancing fault classification capabilities and ensuring operational integrity in Induction Motors (IMs). The methodology presented herein capitalises on the fusion of thermal and vibration sensor outputs to create a robust multimodal monitoring framework. By initially establishing a baseline using thermal imagery, the approach sets a reference standard for comparison. Subsequently, vibration signals undergo a sophisticated transformation from one-dimensional (1D) time-series data to two-dimensional (2D) spatial representations suitable for image processing applications. These representations are further enhanced through Gramian Angular Field (GAF) and Continuous Wavelet Transform (CWT) encoding techniques, which encapsulate temporal dynamics and signal decomposition. Before fusion, thermal and vibration-derived images are meticulously pre-processed to ensure compatibility and maximised data integration. This process's culmination is synthesising a Stitched Multimodal Image Dataset, offering a comprehensive view of the monitored condition. A pre-trained Convolutional Neural Network (CNN), efficient in image-based analysis, will be trained and used to assess the model performance according to the accuracy metric.

3.2.1 Preprocessing Multimodal Data for Induction Motor Fault Classification Method

The proposed method presents a comprehensive and reliable multimodal feature fusion approach for improved fault classification accuracy. The methodology incorporates sensor fusion (combining images from different sensors into one image) and feature fusion (integrating features from these images for classification). It combines vibration signals and thermal images to identify Induction Motor (IM) bearing faults, as summarised in Figure 3.1. The methodology comprises the following steps:

1. Methodology Input Channels: The process begins with thermal images, which establish a baseline using a single-channel approach, serving as a reference for later multimodal analysis. The second input consists of raw vibration signals, which are sub-sampled and prepared for transformation in subsequent steps. This preparation includes creating sub-files and splitting the dataset for further processing.
2. Two-dimensional (2D) Signal Encoding: One-dimensional (1D) vibration signals are converted into a 2D format to enable image processing techniques. As a result, two datasets are created: one with CWT encoded images and another with Gramian Angular Difference Field (GADF) encoded images to capture temporal correlations and to decompose the signals into wavelets, respectively. This step then uses pre-trained CNNs for accuracy assessment (AlexNet, Residual Network-18 (ResNet-18)) and choosing between CWT and GADF. The chosen 2D encoding technique is then nominated for step 5.
3. Preprocessing for Image Fusion: Both thermal images and the chosen approach for the 2D encoded vibration data undergo a preprocessing step to ready them for image fusion. This process merges the encoded vibration images with thermal images, employing a novel methodology in multi-channel image fusion techniques. To ensure accurate correlation between fault types, images are paired based on their health condition. Each GADF image is matched with its corresponding thermal image, both labeled under the same health condition (e.g., Normal, Mild, Moderate, Severe). This alignment is verified by cross-referencing the fault labels from both the MFPT dataset and the lab experiment to ensure proper pairing of fault types. The resulting Stitched Multimodal Image Dataset is then prepared for input into the pre-trained CNN.

4. Pre-trained CNNs for Accuracy Assessment: The Stitched Multimodal Image Dataset images are input into a pre-trained CNN, capitalising on transfer learning capabilities. This approach utilises the pre-existing knowledge of CNNs trained on extensive datasets, thus improving their ability to analyse patterns and identify anomalies. Introducing transfer learning markedly enhances the efficiency and precision of the fault classification process. Specifically, SqueezeNet and ResNet-18 will be evaluated. It is also essential to preprocess the Multimodal Fusion Image Dataset to ensure the image dimensions are compatible with the required input size for the selected CNN, SqueezeNet or ResNet-18.

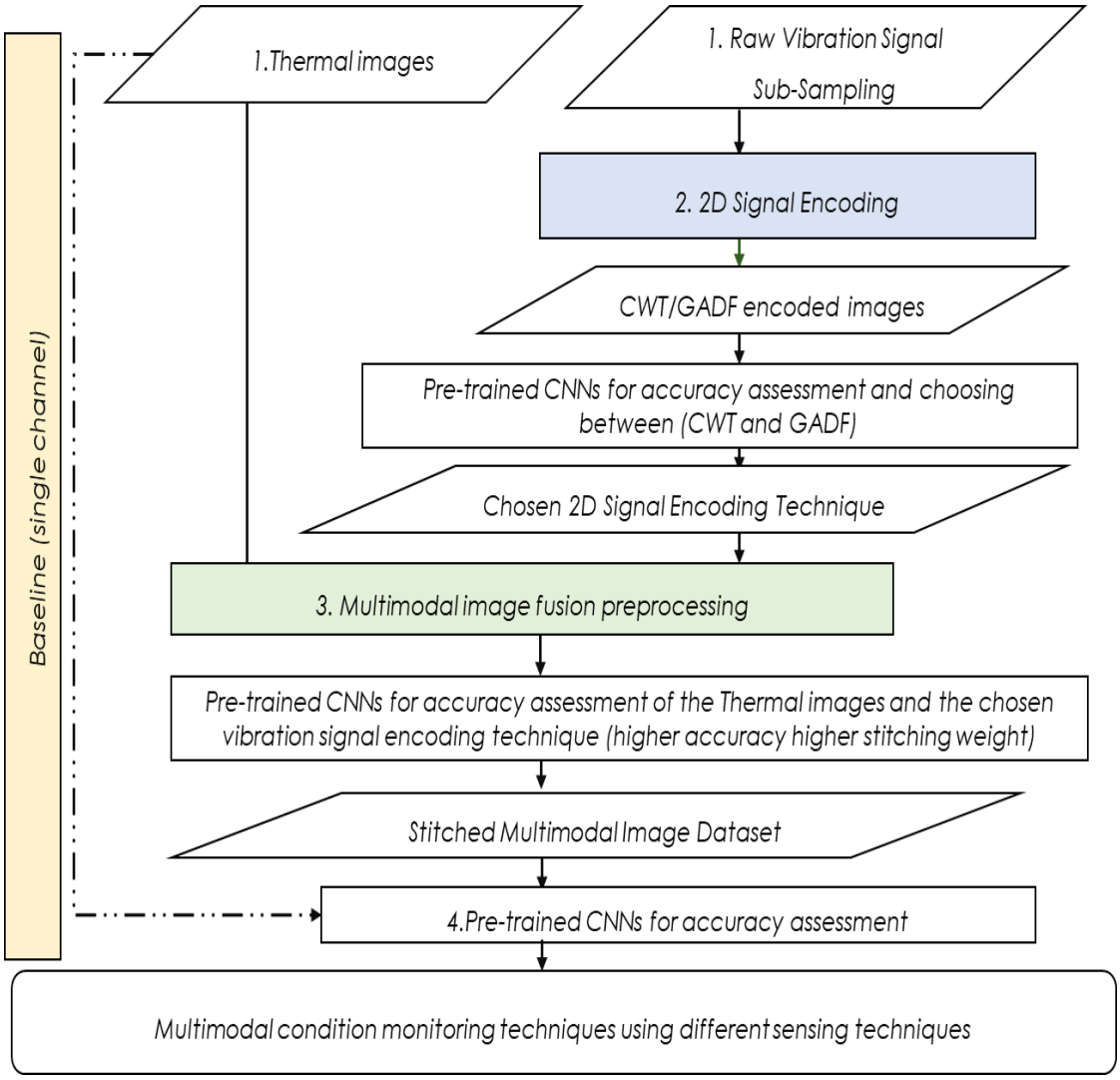


Figure 3.1: Preprocessing of Multimodal Condition Monitoring Data for Classifying Induction Motor Faults Using Deep Learning Methods.

3.2.2 Dataset

The proposed methodology was evaluated using the Machinery Failure Prevention Technology (MFPT) bearing dataset. The testing setup utilised a NICE bearing with eight elements or balls. For healthy conditions, three sets of data were collected, each sampled at a rate of 97,656 Hz for 6 s. Similarly, three sets of data were gathered for Outer Race Fault (ORF) conditions, also sampled at 97,656 Hz for 6 s. Furthermore, seven ORF conditions were recorded at a sample rate of 48,828 Hz for 3 s. Additionally, seven Inner Race Fault (IRF) conditions were sampled at the same rate of 48,828 Hz for 3 s. The test rig was equipped with a NICE bearing characterised by the following parameters (Bechhoefer, 2016):

- Roller diameter = 0.235
- Pitch diameter = 1.245
- Number of elements = 8
- Contact angle = 0

On the other hand, thermal images were captured in the Wolfson Magnetics Laboratory at Cardiff University School of Engineering using a Forward Looking InfraRed (FLIR) thermal camera connected to a computer. These images were taken under six artificially induced faulty conditions and one healthy condition (Al-Musawi et al., 2020; McGhan, 2020a). The dataset utilised here aligns with the health conditions presented in the MFPT bearing dataset, with a focus on methodology.

The proposed methodology is illustrated in Figure 3.1. It begins with evaluating thermal image fault classification performance based on images captured over a 20-minute period in ideal laboratory conditions, encompassing seven health conditions, as shown in Figure 3.2. The thermal images were extracted from the lab-collected images stored in a RawMotorData file. These thermal images were extracted using a Jupyter notebook *(APPENDIX 1)*. The selection of images depended on the health conditions presented in the MFPT bearing dataset, explicitly utilising the categories of Normal (fault-free) or Healthy condition, IRF, and ORF.

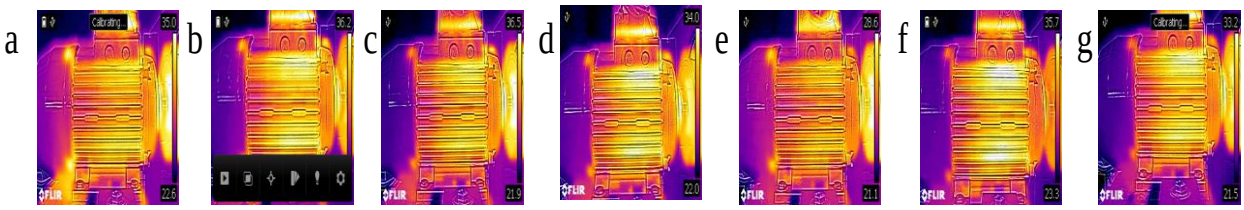


Figure 3.2: Thermal Images for all the Faults and Healthy Conditions: (a) 8 Bars; (b)IRF; (c) ORF; (d) Ball; (e) 4 bars; (f) Normal (fault-free) or Healthy condition; and (g) 1 Bar.

3.3 Results And Discussion

3.3.1 Input Channels

3.3.1.1 Thermal Images

The thermal image dataset is the first input in the framework. In practical scenarios, camera misalignment or mistracking can lead to zooming in or out and variations in image brightness. As a result, image preprocessing was conducted on the thermal images. Consequently, new datasets were generated using the Python OpenCV library. Functions for brightness adjustment, rotation, and zoom were developed and applied to the files. Median blur was also applied to the images to replicate common defects in thermal images and simulate real-world conditions. The thermal image dataset presented challenges due to its inherent noise and small size.

Subsequently, new datasets were generated using the OpenCV library *(APPENDIX 1)*. Functions for brightness adjustment, rotation, and zoom were created and applied to the files. Median blur was also used in the images to replicate typical defects in thermal imagery. Pre-processed examples are displayed in Figure 3.3.

A total of 180 images for each fault type were used, where 60.00% of the dataset was used for training, resulting in 115 images for training, 20.00% for validation (29 images), and 20.00% for testing (26 images). The dotted line in Figure 3.1 represents single-channel input for classification using thermal images only. It is the baseline data for comparing the proposed methodology to determine if it improves the classification accuracy. Only Normal (fault-free) or Healthy condition, IRF, and ORF were used in this study.

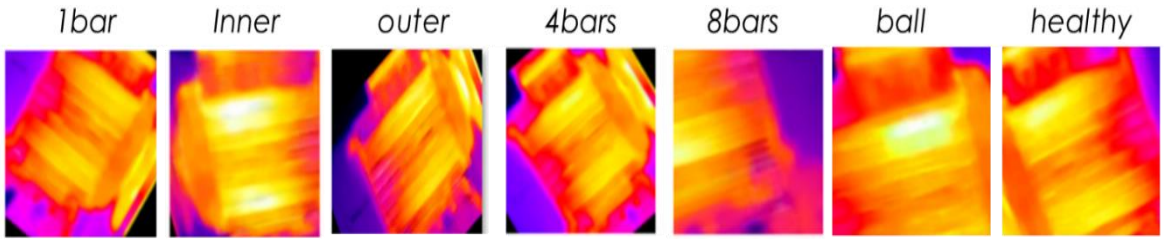


Figure 3.3: Compromised-Quality Thermal Images **(**Preprocessing Stage).

3.3.1.2 Raw Vibration Signal Sub-Sampling

Data were prepared from raw vibration signals to reasonably split folders into subsamples (CSV files) and produce useful 2D images. Data for 0.1 s were extracted from each fault condition (two Normal (fault-free) or Healthy condition files called baseline, five IRF, and seven ORF), resulting in 14 datasets for training and validation. On the other hand, one Normal (fault-free) or Healthy condition file called baseline, IRF, and three ORF were used, resulting in six datasets for testing, as shown in Table 3.1.

Table 3.1: Dataset Used and Subfiles Splitting Count.

Dataset	**Sampling Rate (Hz)**	**Duration (s)**	**Subfiles**
	Training		
baseline_1	97,656	6	117
baseline_2	97,656	6	117
InnerRaceFault_vload_1	48,828	3	58
InnerRaceFault_vload_2	48,828	3	58
InnerRaceFault_vload_3	48,828	3	58
InnerRaceFault_vload_4	48,828	3	58
InnerRaceFault_vload_5	48,828	3	58
OuterRaceFault_1	97,656	6	117
OuterRaceFault_2	97,656	6	117
OuterRaceFault_vload_1	48,828	3	58
OuterRaceFault_vload_2	48,828	3	58
OuterRaceFault_vload_3	48,828	3	58
OuterRaceFault_vload_4	48,828	3	58
OuterRaceFault_vload_5	48,828	3	58
	Testing		
baseline_3	97,656	6	117
InnerRaceFault_vload_6	48,828	3	58
InnerRaceFault_vload_7	48,828	3	58
OuterRaceFault_3	97,656	6	117
OuterRaceFault_vload_6	48,828	3	58
OuterRaceFault_vload_7	48,828	3	58

3.3.2 Two Dimensional Signal Encoding Techniques

The choice of data representation can significantly impact the performance of ML models in fault classification. Selecting relevant features, normalisation, data augmentation, dimensionality reduction, graph-based representations, knowledge graphs, and hybrid approaches can improve model performance by affecting the models' accuracy, speed, and generalisability. Key aspects of data representation that influence ML model performance include feature selection, where relevant features must be chosen to ensure the model captures essential information related to the fault. Conversely, irrelevant features can lead to overfitting or reduced performance (Kareem and Hur, 2022). In this chapter, various signal encoding techniques, such as GAF and CWT spectrograms, will be utilised to enhance data representation. Moreover, hybrid approaches combine different techniques, such as physics-based performance models with DL algorithms, which can help improve the performance of ML models in fault diagnostics (Chao et al., 2019).

3.3.2.1 Gramian Angular Field (GAF)

A methodology that transforms time series into images using two steps: time-series data normalisation and polar coordinates representation of normalised data. There are two types of GAF: Gramian Angular Summation Field (GASF) and GADF. Time series data *x* are first normalised to values between 0 and 1, shown in Equation (3.1) (Han et al., 2021a): where x_i is the raw time-series signal at timestamp *i* and $\tilde{x}_i$ is the normalised signal. Further,

$$\tilde{x}_i = \frac{x_i - x_{min}}{x_{max} - x_{min}} \quad \textbf{(3.1)}$$

x_{min} is the minimum value in the time series data and x_{max} is the maximum value in the time series data (Han et al., 2021a). After that, polar coordinates are used to represent normalised data $\tilde{X}_i$ rather than regular cartesian coordinates by computing the angular cosine value. Equation (3.2) (Ferraro et al., 2020; Han et al., 2021a):

$$\begin{cases} \phi = arccos(\tilde{x}_i), 0 \le \tilde{x}_i \le 1, \tilde{x}_i \in \tilde{X} \\ r = t_i, t_i \in \mathbb{N} \end{cases} \quad \textbf{(3.2)}$$

Here, t denotes the timestamp code at moment *i*, and radius r defines the timestamp.

In contrast to the Cartesian coordinate system, GAF preserves temporal features by constructing an image from the upper-left to the lower-right corner over time. It quantifies temporal correlations across various time intervals using an angular perspective. Specifically, GAF represents either the triangular GASF or the difference GADF between individual points, as detailed in Equations (3.3) and (3.4). This method defines temporal correlations across different intervals using an angular perspective, illustrated by the triangular GASF or the difference GADF between points, as shown in Equations (3.3) and (3.4) (Han et al., 2021a; Kou et al., 2022):

$$GASF = \begin{bmatrix} cos(\phi_1 + \phi_1) & \dots & cos(\phi_1 + \phi_n) \\ cos(\phi_2 + \phi_1) & \dots & cos(\phi_2 + \phi_n) \\ \vdots & cos(\phi_i + \phi_i) & \vdots \\ cos(\phi_n + \phi_1) & \dots & cos(\phi_n + \phi_n) \end{bmatrix} \quad \textbf{(3.3)}$$

$$GADF = \begin{bmatrix} cos(\phi_1 - \phi_1) & \dots & cos(\phi_1 - \phi_n) \\ cos(\phi_2 - \phi_1) & \dots & cos(\phi_2 - \phi_n) \\ \vdots & cos(\phi_i - \phi_i) & \vdots \\ cos(\phi_n - \phi_1) & \dots & cos(\phi_n - \phi_n) \end{bmatrix} \quad \textbf{(3.4)}$$

where ϕ_i represents the angle polar coordinates of the ith timestamp. The diagonal positions keep the original information, while other positions measure the relationship between different time sequences. Consequently, for a time series signal of length n, a numerical matrix of $n \times n$ size can be concluded by the GAF encoding technique, resulting in a 2D image (Han et al., 2021a).

The core concept behind converting time-series data into images using GAF involves creating a matrix based on polar coordinates. This matrix preserves the temporal relationships within the 1D time-series signal, maintaining accurate temporal correlations compared to Cartesian coordinates. The process yields two types of GAF images: GASF and GADF (Toma et al., 2022a).

To transform a given time series $X = x_1, x_2, ..., x_n$ into a range of [-1, 1], we use Equation (3.1) to normalise and scale X where x_i is the element of the time series (Cui et al., 2022; Toma et al., 2022a). The normalisation and scaling process is further detailed in Equation (3.5). This ensures that the data are appropriately scaled for the creation of GAF images:

$$\bar{x}_i = \frac{((x_i - \max(X)) + (x_i - \min(X)))}{\max(X) - \min(X)} \quad \textbf{(3.5)}$$

The angle φ is the inverse cosine of x_i, the radius r is the timestamp, and the time series X is converted into polar coordinates as shown in Equation (3.6) (Cui et al., 2022; Toma et al., 2022a).

$$\begin{cases} \varphi = \arccos(\bar{x}_i), & -1 \leq \bar{x}_i \leq 1, \bar{x}_i \in \bar{X} \\ r = \dfrac{t_i}{N}, t_i \in N \end{cases} \quad \textbf{(3.6)}$$

where t_i is the timestamp, and N is a constant for adjusting image distortion in polar coordinates with time progression. In this context, a mapping is termed a bijection when ϕ is within [0, π], ensuring a unique result for any time series in polar coordinates, preserving distinct temporal relationships, unlike Cartesian coordinates.

3.3.2.2 Continuous Wavelet Transform (CWT)

Wavelet deals with highly fluctuated signals, making it a widespread method of mechanical fault diagnosis. CWT outperforms other methodologies focusing on time-frequency approaches, such as the Short-Time Fourier Transform (STFT) and Gabor Transform (Nishat Toma and Kim, 2020). In general, Wavelet Transform (WT) is a mathematical tool commonly used to reduce the signal's noise effect by splitting the selected signal into small sub-signals, consequently projecting signals into the frequency-time domain utilising subsets of wavelet functions (Divya and Devi, 2021). Hence, the most common faults in bearing components occur periodically and affect the outer, ball, cage, and inner races. Therefore, noise is isolated or shrinks from the signal in the wavelet domain. The periodic impulse of a specific fault will be represented as “energy” in a few significant magnitude coefficients. On the other hand, incoherent noises are translated into many small-magnitude coefficients (Zhang et al., 2022d).

WT is a powerful tool in signal processing, and CWT is a wavelet type. It converts time-domain signals into a time-frequency domain using a convolution operation that leads to correlation coefficients between the mother wavelet function and the original signal (Nishat Toma et al., 2021). The convolution operation is as in Equation (3.7) (Wei et al., 2021):

$$cwt(\alpha,\beta) = \alpha^{\frac{1}{2}} \int x(t)\varphi\ *\ \left(\frac{t-\beta}{\alpha}\right) dt \quad \textbf{(3.7)}$$

where α and β represent the scale factor and the shifting parameter, respectively, x(t) is the selected signal over time-domain t. φ is the wavelet function. Where * represents the operation of the complex conjugate. Hence, CWT converts 1D time-domain signals into 2D time-frequency images (Wei et al., 2021).

However, the mother of the wavelet has two control parameters. First, the scaling parameter is responsible for stretching and contracting the shape of the mother wavelet. Second, the shifting parameter is responsible for the control of the mother wavelet movement along the studied signal. By changing the control parameters on the mother wavelet, the dynamic frequency characteristic of the signal can be revealed (Nishat Toma et al., 2021). In machine fault diagnosis, the Morlet wavelet is combined with the CWT to examine vibration signals. This technique generates time-frequency images, which can be leveraged by CNNs for fault identification. The method is highly efficient and capable of handling complex data rapidly, making it ideal for real-time machinery fault detection (Łuczak, 2024). Therefore, this chapter uses the analytical Morlet (Gabor) wavelet, a kind of CWT that uses the vibration signal subsampling rate as the unit step.

3.3.2.3 Two-Dimensional (2D) Encoded Images: Gramian Angular Difference Field (GADF) and Continuous Wavelet Transform (CWT)

This section tested two signal encoding methodologies to convert vibration signal subsamples into 2D images. The first method was the GADF, a type of GAF. Second, the analytical Morlet (Gabor) wavelet, a CWT type, uses the vibration signal subsampling rate as the unit step.

Each vibration signal subfile presented in Table 3.1 was used as input for the GAF and CWT 2D image signal encoding step; the code is provided *(APPENDIX 2).* On the other hand, the GAF images were created using Python libraries and shown in *(APPENDIX 1).* Vibration images are more extensive than thermal images, resulting in 1,398 images for each encoding methodology. Four hundred sixty-six images were set aside for testing, representing 33.33% of the entire dataset. The remaining data were divided into an 80:20

ratio for training and validation purposes, with 745 images allocated for training and 187 for validation. These images were generated from each subfile,

as indicated in Table 3.1, where the number of images matches the number of subfiles. Therefore, these images retain the information of the signal in 2D diagrams. Figure 3.4 illustrates the GADF signal encoding image, while Figure 3.5 depicts the CWT 2D vibration signal encoding.

Figure 3.4: GADF Encoded Images Demonstration.

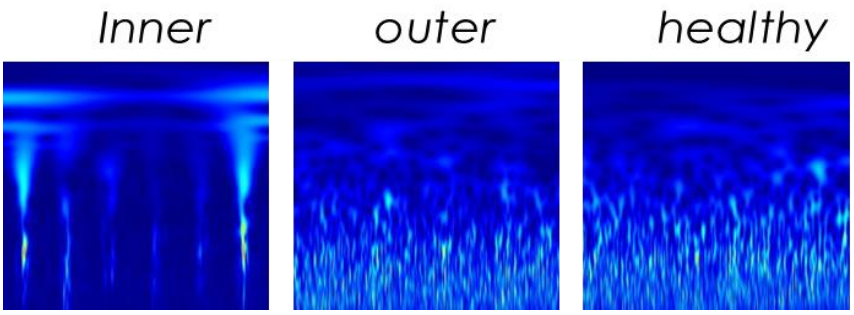


Figure 3.5: CWT Encoded Images Demonstration.

3.3.2.4 CNN Transfer Learning Examples Using AlexNet, ResNet-18 and SqueezeNet

Pre-trained CNNs offer numerous advantages, such as improved accuracy, as they often achieve state-of-the-art performance on various image classification benchmarks (Salehi et al., 2023). These models reduce training time since they come equipped with fundamental features and require only fine-tuning for specific tasks (Alzubaidi et al., 2021). This efficiency extends to using computational resources, making them ideal for handling large datasets or scenarios with limited processing power (Alzubaidi et al., 2021; Salehi et al., 2023). Additionally, pre-trained CNNs can help address class imbalances in datasets, such as those in medical imaging, where some classes may lack sufficient data (Salehi et al.,

2023). They can automatically feature extraction, which is crucial in medical imaging for accurate analysis (Zheng et al., 2023).

Moreover, these models maintain translation invariance, which is vital for consistent image recognition, and include regularisation techniques like Pseudo-task Regularisation (PtR) to dynamically enhance network training without relying on specific regularisation objectives or additional annotations. Their flexibility allows for adaptation to various applications, including image classification, object detection, and segmentation (Salehi et al., 2023). Pre-trained CNNs also benefit from being trained on large-scale datasets, enhancing their performance on downstream tasks, and they are well-suited for domain adaptation in fields like medical imaging, where data availability may be limited (Salehi et al., 2023).

CNN transfer learning can be applied with any pre-trained CNN architecture. The idea is to start with a pre-trained CNN model and adjust it to meet the specific needs of a new task by training it further on a different dataset. This method is especially beneficial when the new dataset is too small or lacks sufficient data to develop a CNN from scratch (Hussain et al., 2019). This chapter will focus on ResNet-18 and SqueezeNet, which are commonly used in Induction Motor (IM) fault diagnosis. Conversely, ResNet-18 and AlexNet were used for performance evaluation, as will be discussed in Chapter 6. On the other hand, AlexNet will also be used in Chapter 4.

In Cinar's (2022) study, SqueezeNet showed promising accuracy levels in fault detection. On the other hand, transfer learning can be utilised on customised CNN architecture (Ye et al., 2021). SqueezeNet was used in 2022 for fault detection in IMs, achieving a high classification score (Cinar, 2022). ResNet-18 was also used by Yuan et al. in 2020 for rolling bearing fault diagnosis on the two publicly available datasets widely used, namely, the MFPT bearing and Case Western Reserve University (CWRU) datasets (Yuan et al., 2020). AlexNet was also utilised by Pinedo-Sánchez et al. (2020) on an unlabelled dataset of vibration signal-encoded images from Intelligent Maintenance Systems. The study exhibited encouraging outcomes in contrast to various other CNN architectures. Notably, using AlexNet to diagnose bearing failure through vibration images was rare.

Pre-trained CNNs can be customised and fine-tuned to the desired dataset to learn features faster and more efficiently than creating a CNN from scratch. Hence, it was proved from the literature in section 2.2.3.1 that using simple CNN architecture resulted in better

performance; SqueezeNet and ResNet-18, which is a short form of the residual net, were explored. Moreover, some applications showcase the versatility of SqueezeNet in addressing challenges related to IMs, ranging from fault diagnosis to image classification. The SqueezeNet model, a type of CNN, has found applications in IMs. One such application is in the fault diagnosis of IMs. Research has been conducted using DL CNN architectures, including SqueezeNet, for fault diagnosis of such defects as broken rotor bars in IMs (Barrera-Llanga et al., 2023).

SqueezeNet networks have also been evaluated for document image classification, demonstrating SqueezeNet's applicability in this domain (Hassanpour and Malek, 2019). A recent study has shown the effectiveness of using SqueezeNet combined with CWT for bearing fault detection in IMs. It achieved outstanding classification accuracies: 99.79% with Morse Wavelet, 98.71% with Bump Wavelet, and 97.64% with Morlet Wavelet. These results highlight the potential of DL models, like SqueezeNet, for precise and efficient fault diagnosis in industrial settings (Boudiaf et al., 2024).

The evolution of DL architectures over the years has been marked by significant milestones, particularly in image classification tasks. AlexNet, introduced in 2012, revolutionised the field by winning the ImageNet challenge, demonstrating the power of Deep Neural Networks (DNNs) (Krizhevsky et al., 2007). Following this, ResNet-18 emerged in 2015 with its innovative residual blocks, enabling the training of even deeper networks by addressing the vanishing gradient problem (He et al., 2016). ResNet-18 is moderate, with 18 fully connected layers and 11.7 million parameters (Kadam et al., 2021; MathWorks-5, 2023). SqueezeNet, published in 2016, further advanced the domain by offering a highly efficient model that achieves comparable accuracy to AlexNet with significantly fewer parameters (Iandola et al., 2016a). SqueezeNet is a simple network with 18 fully connected layers and 1.24 million parameters (Kadam et al., 2021; MathWorks-5, 2023). Table 3.2 compares this thesis's CNN transferred learning architecture (MathWorks-5, 2023).

Table 3.2: CNN Architecture Comparison (MathWorks-5, 2023).

Layer Type	AlexNet (2012)	ResNet-18 (2015)	SqueezeNet (2016)
Input	227x227x3	224x224x3	224x224x3
Convolutional	5 layers	Conv1: 7x7, 64, stride 2	Fire modules (squeeze and expand layers)

Layer Type	AlexNet (2012)	ResNet-18 (2015)	SqueezeNet (2016)
Max Pooling	3 layers	Pool1: 3x3, stride 2	-
Fully Connected	3 layers	1 layer (classifier)	1 layer (classifier)
Output	1000 classes	1000 classes	1000 classes

The following section also discusses the CNN architectures presented in Table 3.2 in detail:

1) AlexNet: AlexNet is a deep CNN composed of five convolutional layers followed by three fully connected layers. It achieved victory in the ImageNet Large Scale Visual Recognition Challenge in 2012, thanks to the work of Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, as presented in their paper titled "ImageNet Classification with Deep Convolutional Neural Networks" (Krizhevsky et al., 2017). AlexNet's architecture comprises eight layers, including five convolutional layers and three fully connected layers (Yu et al., 2021). It also has a 1,000-way softmax output layer for classification. It introduced the Rectified Linear Unit (ReLU) activation function for faster training and implemented overlapping max pooling to reduce representation size and computational load. AlexNet employed normalisation layers, dropout techniques, and data augmentation strategies to prevent overfitting for improved model robustness (Thalagala and Walgampaya, 2021).
2) SqueezeNet: The SqueezeNet model is a CNN that is 18 layers deep and can classify images into 1,000 object categories. It has been trained on over a million images and has learned rich feature representations for many images (MathWorks-4, 2023). Fine-tuning the pre-trained SqueezeNet model with domain-specific data can also enhance its performance for the specific application, allowing it to learn features relevant to the fault diagnosis task (Iandola et al., 2017).

   In the domain of DL for image classification, the SqueezeNet architecture stands out for its strategic reduction of parameters without compromising accuracy. Remarkably, SqueezeNet attains comparable accuracy to the well-established AlexNet on the ImageNet dataset, yet with a model size that is 50 times smaller (Iandola et al., 2016b; Iandola et al., 2017). This parameter reduction enhances computational efficiency, a critical advantage in resource-constrained environments. A key factor in optimising SqueezeNet's performance is manipulating the Squeeze

Ratio (SR), which is defined as the ratio of filters in the squeeze layers versus those in the expand layers. Adjusting this ratio upwards can improve ImageNet's top-5 accuracy up to a certain threshold. Beyond this threshold, the benefits plateau, suggesting that excessively high squeeze ratios may inflate the model's size without yielding accuracy gains. This observation underscores the importance of SqueezeNet's design principles, such as incorporating fire modules and the deliberate balance between minimising parameters and maintaining accuracy (Iandola et al., 2017).

3) ResNet-18: ResNet-18 is a CNN architecture introduced in the paper "Deep Residual Learning for Image Recognition" by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, published in December 2015. This paper introduced the concept of residual learning, which marked a significant advancement in DL architectures. ResNet-18 is a member of the ResNet family, notable for its depth and the utilisation of residual blocks, which effectively address the vanishing gradient problem when training DNNs (He et al., 2016).

   ResNet-18's deep residual network architecture is known for its balance between depth and performance for anomaly detection in Scanning Electron Microscope (SEM) images of nanofibrous materials. ResNet-18, chosen for its optimal trade-off between computational efficiency and accuracy, includes five convolutional stages and is pre-trained on the ImageNet dataset. This architecture facilitates the detection of unexpected anomalous patterns in SEM images, demonstrating its effectiveness in recognising complex scenes and objects, which is crucial for identifying anomalies within the intricate textures of nanofibrous materials (Napoletano et al., 2018).

   The architecture of ResNet-18 consists of a total of eighteen layers, including seventeen convolutional layers and a fully-connected layer; it also has an additional softmax layer for classification tasks. The convolutional layers use 3 x 3 filters, doubling the number of filters as the output feature map size halves. Downsampling is achieved through convolutional layers with a stride of 2, followed by average pooling and a fully connected layer leading to the softmax layer. A key feature of ResNet-18 is the inclusion of residual shortcut connections between layers, which

help address the vanishing gradient problem and facilitate the training of networks (Ramzan et al., 2020; Pandey and Srivastava, 2023).

This stage is pivotal in selecting the proposed methodology's 2D encoding technique. Table 3.1 enumerates all the models created, with the outcomes of this stage detailed in models (3-6). For this analysis, SqueezeNet and ResNet-18 were employed. CWT was identified as yielding superior accuracy. To comply with network specifications, the image sizes were adjusted as required: ResNet and AlexNet necessitate an input size of 224 x 224 pixels. The last fully connected layer in both networks was also modified to address a three-class problem.

3.3.3 Multimodal Image Fusion Preprocessing

The number of generated stitched images was 180, which is comparable to the baseline data or thermal images. These images were merged using the Excel Power Query, which links each image with its saved path. The first column holds the path of the GADF images, the second column contains the path of the thermal images, and the third column indicates the health condition. Consequently, images with the same row number will be stitched together using the proposed methodology, as shown in Figure 3.6. Each thermal image is paired with a unique GADF image, grouped by health condition as depicted in Figure 3.7; the condition is denoted as Normal (fault-free) or Healthy.

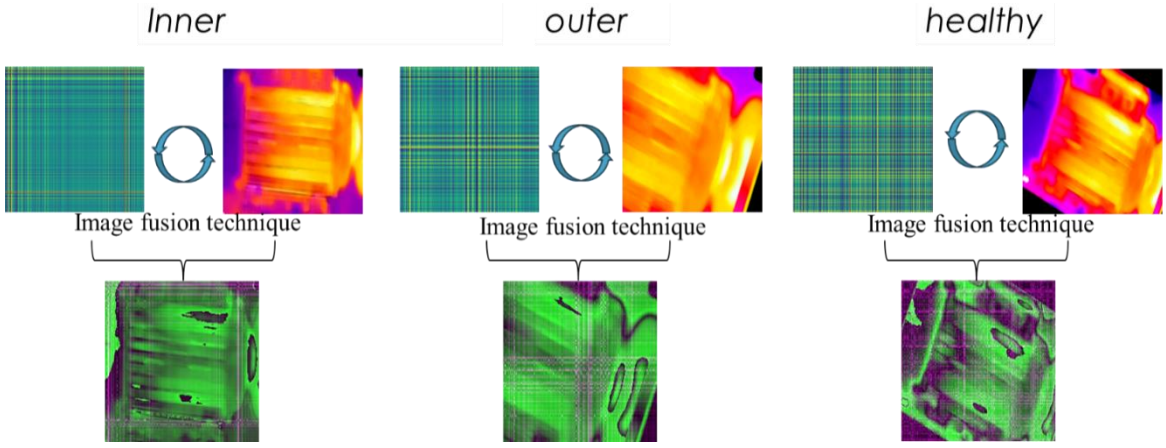


Figure 3.6: Stitched Multimodal Image Dataset Samples Per Health Condition.

Vibration (GADF)	Thermal Images	Fault
C:\Users\Shahd\Documents\MA	C:\Users\Shahd\Documen	healthy
C:\Users\Shahd\Documents\MA	C:\Users\Shahd\Documen	healthy
C:\Users\Shahd\Documents\MA	C:\Users\Shahd\Documen	healthy
C:\Users\Shahd\Documents\MA	C:\Users\Shahd\Documen	healthy
C:\Users\Shahd\Documents\MA	C:\Users\Shahd\Documen	healthy
C:\Users\Shahd\Documents\MA	C:\Users\Shahd\Documen	healthy
C:\Users\Shahd\Documents\MA	C:\Users\Shahd\Documen	healthy
C:\Users\Shahd\Documents\MA	C:\Users\Shahd\Documen	healthy
C:\Users\Shahd\Documents\MA	C:\Users\Shahd\Documen	healthy
C:\Users\Shahd\Documents\MA	C:\Users\Shahd\Documen	healthy
C:\Users\Shahd\Documents\MA	C:\Users\Shahd\Documen	healthy
C:\Users\Shahd\Documents\MA	C:\Users\Shahd\Documen	healthy
C:\Users\Shahd\Documents\MA	C:\Users\Shahd\Documen	healthy

Figure 3.7: Microsoft Excel PowerQuery CSV. File for The Stitched Multimodal Image Arrangement.

After that, image stitching was done by merging similar RGB channels, giving vibration images a higher portion in stitched dataset generation. Image stitching was done using Python Jupyter Notebook, as shown in Figure 3.8. The vibration image has a higher weight in stitched image generation. This is because the encoded GADF vibration signal images shown in Table 3.1 proved more accurate than the thermal images in classifying the health types. Consequently, in the stitched dataset, the vibration images are given a higher weight, accounting for 66.66% of the image, while the thermal images comprise 33.33%. This weighting emphasises the knowledge extracted from the vibration data while leveraging the unique insights of thermal imaging. This is demonstrated in model 5, where the GADF images dataset scored 99.14% using SqueezeNet and 97.64% using ResNet-18. On the other hand, the thermal images dataset scored 87.96% using SqueezeNet and 85.19% using ResNet-18. Consequently, a stitched image comprises 66.66% vibration encoded image and 33.33% thermal image, emphasising the knowledge from vibration image; see the complete code in *(APPENDIX 2)*.

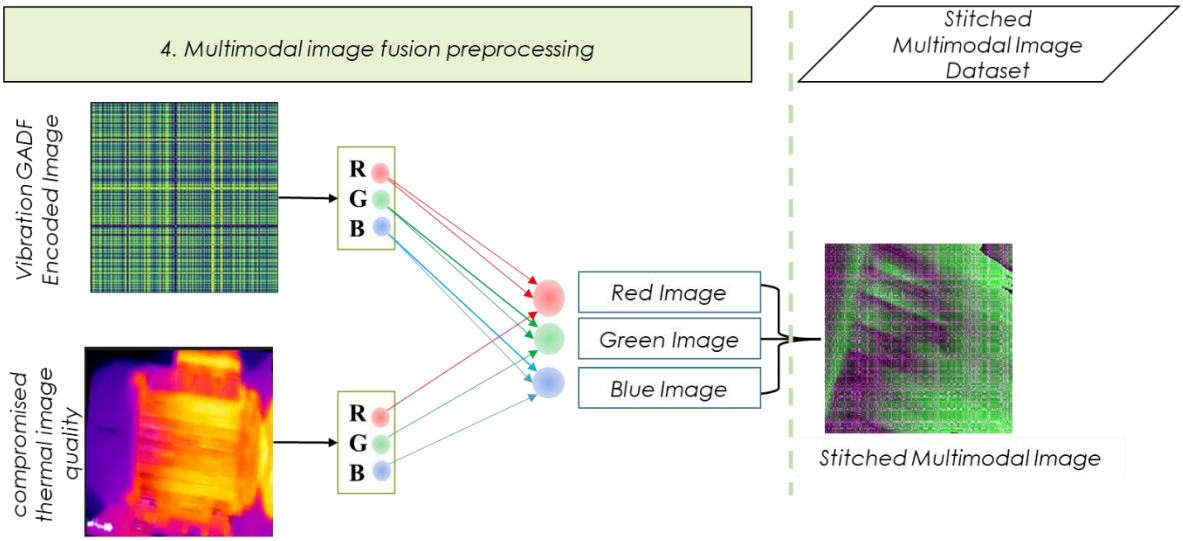


Figure 3.8: Stitched Multimodal Image Dataset Encoding Technique.

3.3.4 Pre-trained CNNs for Fault Classification

CNNs are suitable for high-dimensional feature extraction. CNNs are stable in terms of pattern recognition from images (Han et al., 2021b). Furthermore, CNNs are known for their feature extraction capability from images but encounter difficulties with 1D time series signals (Zhou et al., 2022). Furthermore, CNNs have proved their rotating machinery fault classification capabilities through Vibration Signal Analysis (VSA) (Pinedo-Sánchez et al., 2020). Therefore, CNNs have improved our ability to recognise and understand visual information by automatically learning and capturing relevant patterns in images and videos (Reshadi et al., 2023). Researchers have discovered a variety of CNN architectures to improve classification performance, starting with LeNet-5, designed explicitly for handwritten digit recognition tasks introduced by Yann LeCun, along with his colleagues, in 1998 (LeCun et al., 1998). AlexNet, a notable deep CNN model designed by Krizhevsky et al. (2017), effectively countered overfitting in the 2012 ImageNet challenge through tactics like ReLU activation, dropout, and data augmentation. It comprises an input layer ($224 \times 224 \times 3$), five convolutional layers, and three fully connected layers. Activation functions bolster nonlinearity and convergence (Amanollah et al., 2023). In 2016, SqueezeNet was introduced by Forrest Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer, aiming for a streamlined CNN architecture with

high accuracy and minimal model size. This design allows for deployment on resource-constrained devices without compromising performance (Iandola et al., 2016c).

Classification problems involve mapping inputs to outputs, which is typically achieved through supervised learning. After training a classification model, its quality of learning is evaluated by testing it on unseen data and predicting the respective classes. Alternatively, pre-trained CNN models can be used to assess the similarity between generated and real images. This involves training the pre-trained model on an artificial image dataset and evaluating its performance on a real dataset (Alrashedy et al., 2022).. Various CNN architectures have been employed, including Residual Network 152V2, MobileNetV2, and AlexNet (Alrashedy et al., 2022). Evaluation of deep transfer learning models commonly utilises accuracy Equation (3.7), precision Equation (3.8), and recall Equation (3.9) metrics. These metrics rely on the correct classification of True Positives (TP) and True Negatives (TN) to accurately identify positive and negative instances. Additionally, they consider False Positives (FP) and False Negatives (FN), which refer to the incorrect classification of negative and positive instances, respectively (Nishat Toma and Kim, 2020).

$$\text{Accuracy} = (TP + TN)/(TP + FN + TN + FP) \quad \textbf{(3.7)}$$

$$Precision = TP/TP + FP \quad \textbf{(3.8)}$$

$$Recall = TP/TP + FN \quad \textbf{(3.9)}$$

However, in this section, accuracy was used as an evaluation metric, as shown in Equation (3.7), representing the number of times the model correctly classified fault type over the total number of predictions. Hence, True Positive (TP) represents correctly identified faults, and False Positive (FP) indicates incorrectly classified faults. Similarly, TN denotes correctly identified normal conditions, while incorrectly classifying a normal condition is referred to as False Negative (FN) (Nishat Toma and Kim, 2020).

Two CNN architectures, ResNet-18 and SqueezeNet, were tested for DL model training. Mainly, ResNet-18 and SqueezeNet were used to train, validate, and test the research models, starting with the baseline, which is the compromised thermal image quality dataset in models number 1 and 2, then generated CWT images in models 3 and 4, moving to GADF-generated images in 5 and 6. Finally, after resizing images, the Stitched Multimodal Image Dataset matched each CNN input requirement. Stitched Multimodal

Image Dataset was split into 60.00% for training, 20.00% for validation, and 20.00% for testing. DL network training and signal analysis were done using MATLAB. Microsoft Excel PowerQuery was also used to support the work. Eight different models were trained and validated. Then, the testing split was used to assess the models' performance, as shown in Table 3.3.

Table 3.3: Tested Model Performance.

Model No.	Inputs	Image Encoding	CNN	Test Accuracy (%)
1	Thermal images	None	SqueezeNet	87.96%
2	Thermal images	None	ResNet-18	85.19%
3	Vibration images	CWT	SqueezeNet	97.85%
4	Vibration images	CWT	ResNet-18	95.92%
5	Vibration images	GAF	SqueezeNet	99.14%
6	Vibration images	GAF	ResNet-18	97.64%
7	Thermal+vibration (Stitched Images)	GAF(vibration)	SqueezeNet	98.15%
8	Thermal+vibration (Stitched Images)	GAF(vibration)	ResNet-18	100%

The first two models focused on fault classification and condition monitoring performance on compromised thermal image quality modified using OpenCV to mimic critical conditions. Then, ResNet-18 and SqueezeNet were applied to assess each CNN's performance on the test dataset, as shown in Table 3.3. Using either ResNet-18 or SqueezeNet will result in low accuracy and needs further improvement, although, SqueezeNet performed better than ResNet-18 in accuracy by almost 2.77%. The trained model misclassified the IRF nearly 30.55% of the time using ResNet-18 and 16.66% using SqueezeNet, as shown in Figure 3.9 where Inner represents the IRF, Outer (ORF), and Normal (fault-free) or Healthy condition.

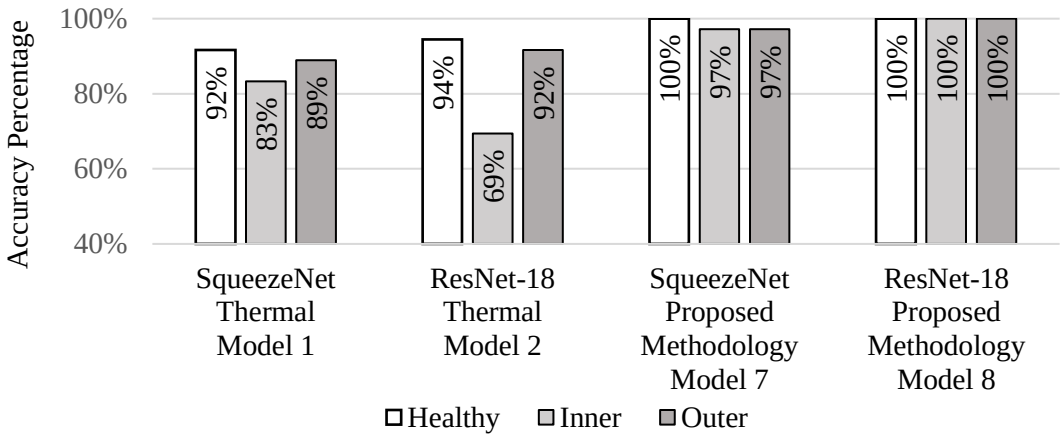


Figure 3.9: Thermal Images Vs. Proposed Methodology Accuracy Per Fault.

Consequently, fault classification with a single input in noisy conditions and limited data is unreliable and needs further enhancement. Hence, this chapter proposed a multimodal condition monitoring system, leading to a reliable fault classification system with a holistic learning experience using transfer learning capabilities in feature extraction.

Vibration signals were analysed individually to help choose the optimal model that would be used to elevate thermal image performance. Thus, the following four models analysed vibration signals by determining the performance of GAF and CWT encoding methodologies on subsamples using two CNNs, ResNet-18 and SqueezeNet, resulting in four different combinations. Then, the highest-performing model in terms of overall accuracy was selected.

However, the difference was slight between Model 3 and Model 6 vibration models, as shown in Table 3.3. The overall accuracy of using the GADF encoding methodology was better than the CWT encoding methodology in fault classification. Specifically, GADF signal encoding achieved a mean accuracy of 98.39% ± 1.07%, with Model 5 reaching 99.14% using SqueezeNet and Model 6 reaching 97.64% using ResNet-18. On the other hand, CWT signal encoding resulted in a slightly lower mean accuracy of 96.89% ± 1.38%, with Model 3 reaching 97.85% using SqueezeNet and Model 4 reaching 95.92% using ResNet-18.

However, Figure 3.10 shows the difference between vibration signal encoding methodologies in terms of fault classification accuracy by fault type on the test dataset. Hence, Model 5 is considered the most accurate compared to other models. Consequently, the GADF encoding methodology was adapted to the proposed methodology. Models could classify the IRF as higher than other fault types presented in Figure 3.10. Conversely, Model 1 and Model 2 struggled to classify the IRF type the most, as shown in Figure 3.9.

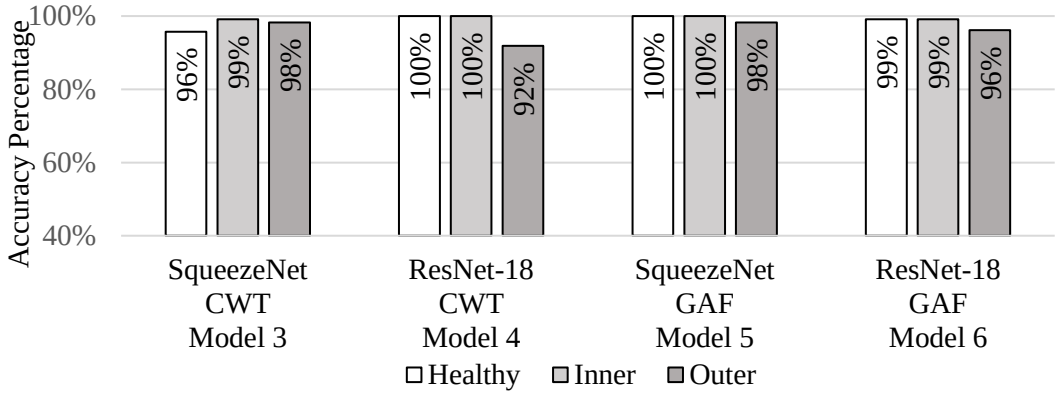


Figure 3.10: Vibration Signal Encoding Models Accuracy Per Fault.

The proposed methodology was implemented using SqueezeNet in Model 7 and ResNet-18 in Model 8. However, the selected vibration signal encoding technique was GADF. Also, the Stitched Multimodal Image Dataset Encoding Technique was used to merge encoded vibration images with the thermal image. As shown in Table 3.3, the proposed methodology, using both CNNs, performed better than Model 1 and Model 2. The model's accuracy using ResNet-18 jumped from 85.19% to 100%, leading to a 14.81% improvement. Also, it rose from 87.96% using SqueezeNet to 98.15.00%, making a 10.19% improvement in accuracy.

Furthermore, Figure 3.9 compares the first two models, depending on a single sensor fault classification system, with the proposed Novel Preprocessing of Multimodal Condition Monitoring Data methodology. It was clear from Figure 3.9 that the proposed methodology enhanced the fault classification accuracy, especially IRF type classification, by almost 31.00% using ResNet-18 and 14.00% using SqueezeNet. The accuracy enhancement of other conditions was also remarkable; ORF classification was improved by the same percentage,

8.00%, using both CNNs. Also, the Normal (fault-free) or Healthy condition classification was enhanced by 8.00% using SqueezeNet and 6.00% using ResNet-18.

3.4 Summary

Data-driven fault classification aims to improve detection in Induction Motor (IM) bearings. The study began by testing fault classification accuracy using compromised thermal image quality alone with pre-trained CNNs. The highest score achieved was 87.96% using SqueezeNet, indicating that fault classification with a single input in noisy conditions and limited data has scope for improvement.

To address this, the chapter proposes the Preprocessing of Multimodal Condition Monitoring Data for classifying induction motor faults using DL methods. This methodology investigates two signal-to-image encoding methods: CWT and GAF. The results revealed minimal differences between CWT and GAF, with GAF outperforming CWT by 1.50% on average. Specifically, GAF showed a 1.72% higher accuracy using ResNet-18 and a 1.29% higher accuracy using SqueezeNet, leading to an overall mean accuracy of 98.39% ± 1.07% for GAF compared to 96.89% ± 1.38% for CWT.

The proposed methodology employs a Stitched Multimodal Image Dataset Encoding Technique, combining GAF images with compromised thermal images. This process involves merging encoded vibration images (weighted at 66.66%) and thermal images (weighted at 33.33%) to emphasise the knowledge extracted from vibration data while leveraging thermal imaging insights. This approach significantly improved overall classification accuracy by 14.81% for ResNet-18 and 10.19% for SqueezeNet compared to using compromised thermal images as single-channel inputs. Consequently, the proposed approach improved classification accuracy by 12.50%, achieving 99.10% ± 0.50% when using both ResNet-18 and SqueezeNet compared to using compromised thermal images alone.

The main contribution of this chapter is the novel preprocessing methodology for multimodal condition monitoring data to classify IM faults using DL methods with compromised thermal image quality. Among the numerous noteworthy sub-contributions are the following:

1. Integration of multiple data types (sensor fusion): The methodology combines data from various sensors, specifically, thermal images and vibration signal data encoded

as images (using GADF). This integration is a classic example of feature fusion, whereby data from various sources or different types are merged to enhance decision-making.

2. Weighted combination (feature fusion): Assigning different weights to thermal and vibration images ensures that the most relevant features from each data type are emphasised. The approach weights the thermal and vibration images (66.66% vibration encoded and 33.33% thermal images), a key feature fusion aspect. This weighted combination ensures that the most relevant features from each data type are emphasised in the analysis.
3. Enhanced data representation: Fusing features from compromised-quality thermal images and vibration data creates a comprehensive signal representation, capturing more nuances of the fault conditions than either data type could achieve independently.
4. Use in ML models: The fused data are then used to train ML models (CNNs like SqueezeNet and ResNet-18). Feature fusion before model training is a strategic approach to improve the model's ability to learn from a richer set of features, thereby enhancing the overall accuracy and robustness of the fault classification.

While the proposed Preprocessing of Multimodal Condition Monitoring Data for classifying induction motor faults using DL methods improved the baseline accuracy using compromised thermal images as single-channel inputs, the small sample size of the thermal image dataset (180 images per health condition) remains a limitation. To overcome this, Chapter 4 (the following chapter in the thesis) aims to expand the dataset and employ data augmentation techniques using GANs.

Chapter 4: A Novel Approach Using Wasserstein Generative Adversarial Networks with Gradient Penalty (WGAN-GP) and Conditional WGAN-GP for Generating Artificial Thermal Images of Induction Motor Faults

4.1 Proposed Methodology

Training Generative Adversarial Networks (GANs) involves updating the parameters of the Generator (G) and Discriminator (D) using optimisation methods like Stochastic Gradient Descent (SGD), Adaptive Moment Estimation (Adam), or Root Mean Square Propagation (RMSProp). The goal is to reach a Nash equilibrium, where D is no longer able to distinguish between real images x and generated fake images ($x' = G(z)$) (Pang et al., 2022). GANs have two probability distributions, namely, P_g, the distribution from G's implicit distribution, and P_r, the probability distribution of real images. The D outputs a number between 0 and 1, representing the probability that the input image is real, with a score close to 1 indicating a real image. The G and D are continuously updated to improve the model's ability to generate data closer to real images and discriminate between real and fake data using the objective Equation (4.1) (Han et al., 2020):

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim P_r(x)}[\log(D(x))] + \mathbb{E}_{z \sim P_z(z)}[\log(1 - D(G(z)))] \qquad \textbf{(4.1)}$$

Equation (4.1) consists of two terms: G takes a noise vector z sampled from a prior distribution. P_z and generates a sample G(z) in the target data distribution. D takes a sample x from either the real data distribution P_r or the generated data distribution P_z (i.e., D(G(z))), and outputs a probability score indicating whether the input is a real or fake sample. The first term in the equation is the expected value of the logarithm of D's output on real samples x, while the second term is the expected value of the logarithm of D's output on fake samples G(z) (Han et al., 2020).

The proposed methodology is structured into two main parts to investigate the generation of artificial thermal images for induction motor fault detection using GANs. In the first part, the study begins with a foundational exploration of GANs, focusing on understanding their behaviour and evaluating the impact of various parameters on their performance. Initially, the basic Deep Convolutional Generative Adversarial Network (DCGAN) architecture is employed with an original image size of 224 x 224 x 3 pixels. Subsequently, the study introduces Wasserstein Generative Adversarial Networks with Gradient Penalty (WGAN-GP), leveraging advanced training parameters and GPU resources to specifically analyse the Inner Race Fault (IRF) at a resolution of 32 x 32 x 3 pixels.

The second part of the study expands to include four separate WGAN-GP models trained for IRF, Outer Race Fault (ORF), 8-bars, and Normal (fault-free) or Healthy condition. Following this, all health conditions are collectively trained using the conditional Wasserstein Generative Adversarial Network with Gradient Penalty (cWGAN-GP) model, generating high-quality artificial thermal images at a resolution of 128 x 128 pixels.

The proposed frameworks of WGAN-GP and cWGAN-GP are detailed in the subsequent sections. They emphasise their applications and outcomes in generating artificial thermal images that closely resemble real images from the dataset, specifically for detecting faults in the bearings of Induction Motors (IMs).

4.1.1 Foundational Study of Generative Adversarial Networks (GANs)

This section explores the behaviour of GANs and assesses how various parameters affect their performance. It includes experiments with the DCGAN architecture to generate artificial thermal images that closely resemble real images from the dataset, specifically targeting IRF, ORF, and Normal (fault-free) or Healthy condition. Additionally, it introduces the use of WGAN-GP to generate artificial thermal images that accurately replicate real images from the dataset, focusing on IRF.

4.1.1.1 Basic Deep Convolutional Generative Adversarial Networks (DCGAN)

The Basic DCGAN is a type of GAN discovered by Radford, Metz, and Chintala in 2014; they proved it could generate realistic images not added during the training stage (Kusiak, 2020). It is considered one of the most recent significant improvements in GAN architecture in vision modelling. Its deep architecture can stabilise the training, generating high-quality images. DCGAN utilises Convolutional Neural Network (CNN) architecture with GANs (Niu et al., 2020).

One of the critical innovations of DCGANs is the replacement of pooling layers with convolutions that use strides and convolutions with fractional strides. ‘Strides’ refers to the step size with which the filter moves across the input image, while fractional strides are used to increase the spatial dimensions of the input. This allows the Generator (G) and Discriminator (D) to learn convolutional operations, spatial downsampling, and upsampling individually. By doing this, DCGAN ensures that the G and D networks can learn independently, which can help to stabilise the training process.

The second innovation in DCGAN is Batch Normalisation (BN), which stabilises learning. BN is a technique used to normalise the input to a layer, which helps solve the vanishing gradient problem and prevents the deep G from collapsing all samples to the same points. Finally, DCGANs use Rectified Linear Unit (ReLU) and LeakyReLU activation functions to allow the model to learn quickly and perform well. The ReLU activation function is used in all G layers except for the last layer, which uses the Tanh activation function to produce image pixel values between -1 and 1. LeakyReLU activation functions are used in all D layers to prevent the problem of "dying ReLU" and enable the model to learn from small gradients. (Alotaibi, 2020; Wang et al., 2021c). On the other hand, DCGANs produce high-quality images but require a long training duration (Al-Qerem et al., 2019).

DCGAN proved its efficiency (Du et al., 2019) for image generation in solving imbalanced datasets in the chemical industry fault diagnosis field. It was also used by He et al. (2021) for axial piston pump bearing fault diagnosis to mitigate data availability and missing fault labelling challenges. DCGAN was also used in IM fault classification using Case Western Reserve University's (CWRU) famous dataset in which Continuous Wavelet Transform (CWT) images were synthesised (Zhong et al., 2023).

Consequently, the first architecture in this section is the DCGAN model, which was trained on Google Colab using a Tesla T4 GPU. Training and testing images of size 224 x 224 x 3 were stored in a Google Drive directory. The G model takes a random noise vector of size 100 as input and generates an image, while the D model predicts whether the image is real or fake. Both models were trained alternately using the binary cross-entropy loss function and the Adam optimiser. The complete code of DCGAN is shown in *(APPENDIX 3).*

Images were loaded and processed using the ImageDataGenerator function from the Keras API, which also handled data augmentation. Augmenting the data can help improve the performance of Machine Learning (ML) models by increasing the size of the training set. This process allows the model to learn more robust features and reduces overfitting (Yousuf et al., 2024). Pixel values of the images were rescaled to range between 0 and 1. The training dataset contained 288 images, and the batch size was set to 32. Images in the training dataset were labelled 'inner' with the class_mode set to 'binary'. The validation dataset, stored in the

same Google Drive directory, consisted of 72 images, resized to 224 224 pixels and similarly labelled.

The G model consists of several dense and reshaping layers, followed by transposed convolutional layers with BN and ReLU activation. The output layer uses a Tanh activation function to generate values between -1 and 1. In contrast, the D model includes several convolutional layers with BN and LeakyReLU activation, culminating in a flattened layer and multiple dense layers. The output layer employs a sigmoid activation function to produce values between 0 and 1, indicating the probability that the input image is real.

Both the G and D models are compiled with binary cross-entropy loss, and the adversarial model is compiled with the Adam optimiser using LRs of 0.0001 and 0.0002 and beta_1 of 0.5. During training, the models are trained alternately—the G aims to create images that fool the D. In contrast, the D learns to distinguish real images from fakes generated by the G. All experiments were conducted on a Tesla T4 GPU using the Google Colab Pro platform.

4.1.1.2 Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP) on Inner Race Fault (IRF)

A Wasserstein Generative Adversarial Network (WGAN) differs from traditional GANs by not using a sigmoid activation function at the end of the critic model and by employing the Wasserstein Distance (Earth Mover's Distance, or EMD) as its loss function, instead of the Jensen-Shannon Divergence (JSD) used in standard GANs. In WGAN, the discriminator is called the "critic" because it evaluates the quality of generated samples by assigning them a continuous score rather than classifying inputs as real or fake (Gulrajani et al., 2017). Equation (4.2) consists of two parts. In the first part, the critic applies the function f to a real image x from the real probability distribution. In the second part, x is taken from the output of Generator (G), generated from a latent noise vector, and then the critic is applied to the generated image. The critic is constrained with $\max_{\|f\|_L \leq 1}$, ensuring the function is Lipchitz continuous. This constraint is important for the critic to differentiate between real and generated samples. The critic estimates the Wasserstein distance between the real and generated data distributions, guiding G to generate more realistic samples. The critic aims to maximise the expression, while G aims to minimise this distance (Arjovsky et al., 2017a).

$$\max_{\|f\|_L \leq 1} \mathbb{E}_{x \sim P_r}[\mathrm{F}(x)] - \mathbb{E}_{x \sim P_g}[F(x)] \quad \textbf{(4.2)}$$

Consequently, the second architecture is the WGAN-GP, a type of GAN architecture that uses a gradient penalty (GP) to enforce the Lipschitz continuity of the D. The D is trained to output a scalar value instead of a probability. The G is trained to minimise the Wasserstein distance between the distribution of real and generated samples. The G loss is the negative Wasserstein distance, and the D loss is the difference between the average D output on the real samples and the generated samples, plus a GP term. The GP term is added to ensure the D satisfies the Lipschitz continuity condition with a 0.0001 Learning Rate (LR), batch size of 64 and Adam optimiser (Gulrajani et al., 2017).

As mentioned, the dataset consists of 288 training images for the IRF class and 72 test images, with an original image size of 224 x 224. Since the training dataset size is relatively small and the dataset resolution of 224 x 224 is relatively big, the images in the dataset were resized to 32 x 32, and the WGAN-GP model was trained for 10,000 iterations with a batch size of 64 on a GPU in Colab. This second architecture is a proof of concept which will represent the baseline for Part 2, aiming for further improvement in image resolution, including other faults, and similarity assessment compared to real datasets.

4.1.2 Advanced GANs Framework

4.1.2.1 Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP)

WGAN-GP addresses the limitations of weight clipping in regular WGAN. Instead of weight clipping, the GP is used to enforce the Lipschitz constraint on the critic. WGAN-GP, introduced in 2017 by Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville (Gulrajani et al., 2017), improves stability, resolves mode collapse, and optimises hyperparameters in training. The Wasserstein distance metric measures the difference between generated and real images. Additionally, the algorithm includes a GP term in the critic for smoothness. The number of Generator (G) and critic iterations, as well as the strength of the gradient penalty, can be adjusted using a lambda term (Wang and Wang, 2019).

4.1.2.2 Proposed Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP) Methodology

Figure 4.1 illustrates the overall WGAN-GP methodology for generating thermal images of IMs under various health conditions. The framework involves training the Discriminator (D) and Generator (G) to produce realistic images. D distinguishes between real and fake images, while G generates images to fool D. The loss function is based on the Wasserstein distance between the distributions of real and fake images with a GP to control D's power. Training alternates between D and G until convergence. The G is a neural network that takes a 100-dimensional latent vector as input and uses transpose convolutional layers to generate images of size C x 128 x 128, where C is the number of channels. The model uses a main module consisting of several convolutional layers followed by a Tanh activation function to generate the image. The output of the main module is then passed through the Tanh function to normalise the pixel values between -1 and 1. D is a neural network with three layers of filters (256, 512, 1024), taking images of size 128 x 128 with C channels and outputting a single value indicating real or fake. The input image is passed through a sequential module and then flattened to be fed into a fully connected layer. The full code is available in the thesis supplementary file.

The proposed methodology aims to comprehensively assess the performance of the WGAN-GP model on the IM dataset (refer to section 4.1.3) with varying resolutions. Experiment results demonstrated that the WGAN-GP model effectively generated IM images that closely resembled real images in the dataset. However, challenges arose with the small sample sizes and lower image quality in certain classes, such as the 'ball' and '1bar' classes, which impacted the model's performance. To address these issues, the following section will explore methods to augment the dataset and enhance the model's performance for classes with smaller sample sizes and lower image quality. Specifically, the cWGAN-GP will be used, where the author plans to incorporate bearing fault types as a condition in the cWGAN-GP and train all fault types simultaneously to reduce training time.

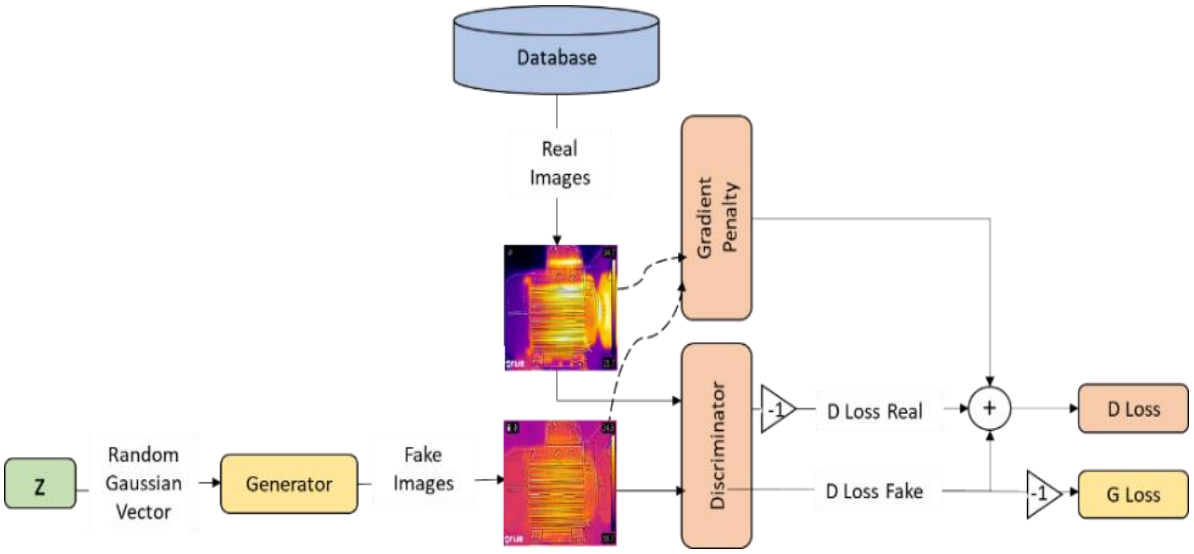


Figure 4.1: Proposed Wasserstein GAN with Gradient Penalty (WGAN-GP) Methodology.

4.1.2.1 Conditional Wasserstein Generative Adversarial Network with Gradient Penalty (cWGAN-GP)

The cWGAN-GP is an extension of traditional GANs called conditional Generative Adversarial Networks (cGANs). It introduces an additional input, denoted as y, to the network, representing additional information such as class names, data from another model, vectors, or images. This conditional factor adds a new dimension to the min-max game between the Generator (G) and Discriminator (D) (He et al., 2022).

The objective function of cWGAN-GP, as shown in Equation (4.3), involves D outputting a high value when given real data point x conditioned on a label y drawn from the true distribution P_r (He et al., 2022; Pang et al., 2022). The second part of the objective function calculates the expected value of the logarithm of D's output when given a fake data point generated by G using a noise vector z drawn from a prior distribution P(z) conditioned on the same label y (He et al., 2022). The goal is to optimise G and D to minimise this objective function, generating high-quality conditional samples.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim P_r(x)}[\log(D(x|y))] + \mathbb{E}_{z \sim P_z(z)}[\log(1 - G(z|y))] \quad \textbf{(4.3)}$$

4.1.2.2 Proposed Conditional Wasserstein Generative Adversarial Network with Gradient Penalty (cWGAN-GP) Methodology

To enhance the generation of Induction Motor (IM) thermal images under different health conditions, we introduce the cWGAN-GP. This approach incorporates label information into the model inputs, enabling Generator (G) and Discriminator (D) networks to generate class-specific images that benefit from patterns of other classes, leading to faster convergence. Figure 4.2 illustrates the cWGAN-GP methodology, which represents the image class and includes a condition vector as input for both G and D networks. The loss function incorporates the Wasserstein distance and a GP term for smoothness in D. The G in the cWGAN-GP methodology is similar to the WGAN-GP, with the addition of a one-hot encoded condition vector representing all health state classes. It takes a concatenated input of the latent vector and condition vector, using transpose convolutional layers to generate C x 128 x 128 images.

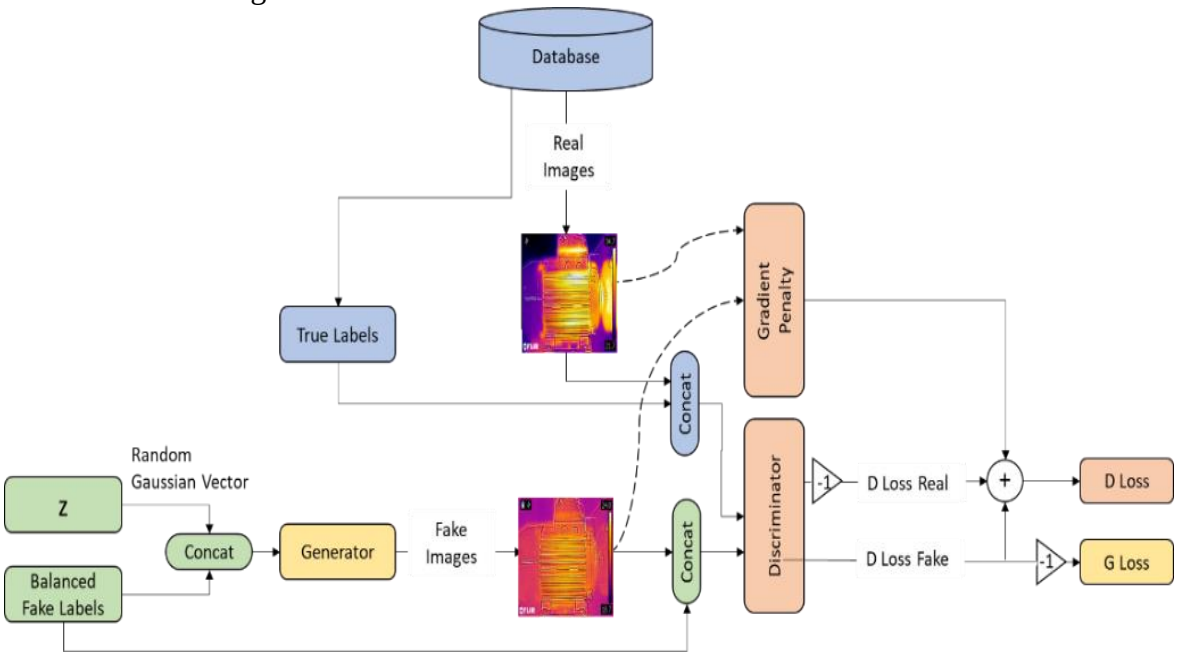


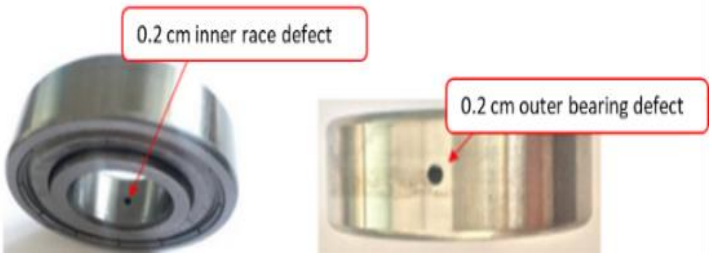
Figure 4.2: Proposed Conditional Wasserstein GAN with Gradient Penalty (cWGAN-GP) Methodology.

The model includes a main module with convolutional layers and a Tanh activation function. The D in cWGAN-GP is similar to WGAN-GP, with the addition of a condition vector concatenated with the input image, resulting in an input shape of (10, 128, 128) after

combining the label information. The output of D is a single value indicating real or fake. The complete code is available in the thesis supplementary file.

4.1.3 Dataset

Wolfson Magnetics Laboratory is located at Cardiff University School of Engineering. The laboratory conducted experiments to simulate six different failure modes and Normal (fault-free) or Healthy condition. To create these failure modes, a 2mm diameter drill was used to make holes in both the inner and outer parts of the bearing, as illustrated in Figure 4.3 (Al-Musawi et al. 2020)..



(a) (b)

Figure 4.3: Bearing Faults (a) IRF, (b) ORF (Al-Musawi et al. 2020).

This chapter explicitly investigates three conditions: IRF, ORF, and Normal (fault-free) or Healthy condition.

To achieve this, thermal images of bearing faults were collected using a forward-looking infrared (FLIR) thermal camera positioned 30cm from the centre of the housing. The camera was connected to a computer to capture images of six artificially induced faults and one health condition, Figure 4.4. One hundred twenty images were captured under three load types, with 360 images per condition. The data were split into 80.00% for training, equivalent to 288 RGB images, and 20.00% for testing, equal to 72 RGB images (Al-Musawi et al., 2020; McGhan, 2020b).

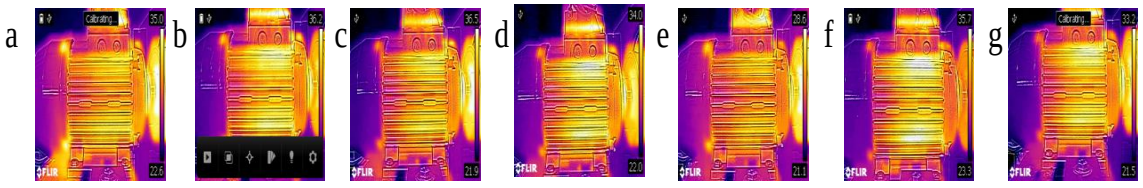


Figure 4.4: Thermal Images for all the Faults and Normal Health Conditions: (a) 8Bars; (b)IRF; (c) ORF; (d) Ball; (e) 4Bars; (f) Normal (fault-free) or Healthy condition; and (g) 1Bar.

4.2 Results and Discussion

4.2.1 Basic Deep Convolutional Generative Adversarial Network (DCGAN) and WGAN-GP

The Basic DCGAN was tested on three health condition datasets: IRF, ORF, and Normal (fault-free) or Healthy condition. The experiments varied in terms of hyperparameters, such as LR (0.0001, 0.0002), batch size (16,32), and the number of epochs (50, 500). The training and test dataset was in the Google Drive directory and contained 288 images. The images were resized to 224 x 224 pixels, and the batch size was 32. On the other hand, the WGAN-GP was tested on IRF only with advanced training parameters. The IRF dataset consists of 288 training images for the "IRF" class and 72 test images, with an original image size of 224 x 224. Since the training dataset size is relatively small, the author resized the images to 32 x 32 and trained the model for 10,000 iterations with a batch size of 64 on a GPU in Colab, as shown in Table 4.1.

To evaluate the performance of each experiment, the Generator (G) and Discriminator (D) losses were assessed. A lower D loss relative to the G indicates superior D performance. Conversely, a lower G loss suggests a better performance of the G. These findings align with the results presented in Table 4.1, where the performance of each model can be observed based on their respective losses. The experiments were conducted in two stages: the first involved using basic DCGAN with simple parameters (Experiments 1-12) to test various health condition datasets. In contrast, the second stage focused on utilising WGAN-GP with advanced parameters for the IRF.

- DCGAN Architecture: Three conditions were tested, starting with the IRF (Experiments 1-4): longer training time led to better performance, with the best result observed in Experiment 2. It used 500 epochs, LR of 0.0001, a batch size of 32, and achieved a G loss of 4.627 and a D loss of 0.0002. ORF (Experiments 5-8): increasing the batch size led to better performance, with the best result obtained in experiment 7, which used 50 epochs, LR of 0.0002, a batch size of 16, and achieved a G loss of 0.0021 and D loss of 3.4129. Normal (fault-free) or Healthy condition (Experiments 9-12): longer training time resulted in improved performance, with the best outcome observed in Experiment 12, which utilised 1,000 epochs, LR of 0.0002, a batch size

of 32, and achieved a G loss of 8.3201 and a D loss of 5.0697. However, visual inspection results were less promising than the project objective.

- WGAN-GP Architecture: the IRF was exclusively tested using WGAN-GP (Experiment 13) with more epochs and longer training times. The experiment required 11 hours of training time. The G loss was a high negative number, while the D loss was -1897.3366 for fake samples and -1548.9419 for real samples. These findings suggest that the D network distinguished between real and fake samples, minimising the loss during training. Similarly, the G loss was also a high negative number, specifically -1,896.8599. This indicates that the G successfully generates samples that D classifies as real samples with high confidence.

Table 4.1: GAN Performance for Fault Detection Experiments.

No.	Model	Dataset	LR	Batch Size	Epochs	Training time (min)	(G) Loss	(D) Loss
1	DCGAN	IRF	0.0001	16	500	48	3.2172	5.537
2	DCGAN	IRF	0.0001	32	500	41	4.627	0.0002
3	DCGAN	IRF	0.0002	16	50	5	0.0019	3.5658
4	DCGAN	IRF	0.0002	32	500	40	5.0668	7.6786
5	DCGAN	ORF	0.0001	16	50	4	0.0045	3.0554
6	DCGAN	ORF	0.0001	32	50	4	0.0079	2.782
7	DCGAN	ORF	0.0002	16	50	4	0.0021	3.4129
8	DCGAN	ORF	0.0002	32	50	4	0.00308	3.2324
9	DCGAN	Normal	0.0001	16	50	4	0.00681	2.8419
10	DCGAN	Normal	0.0001	32	50	4	0.0068	2.8419
11	DCGAN	Normal	0.0002	16	50	5	0.00681	2.84186
12	DCGAN	Normal	0.0002	32	500	50	8.32011	5.0697
13	WGAN-GP	IRF	0.0001	64	10,000	660 (11 hours)	-1896.8600	Loss Fake: -1897.3366 Loss Real: -1548.9419

Consequently, the choice of hyperparameters significantly impacts model performance. Generating artificial fault images of large size and high complexity posed challenges requiring significant time and GPU capabilities. Table 4.1 has provided insights into the performance of GANs using DCGAN. Implementing WGAN-GP with advanced

hyperparameters (10,000 epochs, batch size 64) on a GPU yielded an efficient generation of motor images closely resembling real images over time, as shown in Figures 4.5, 4.6, and 4.7.

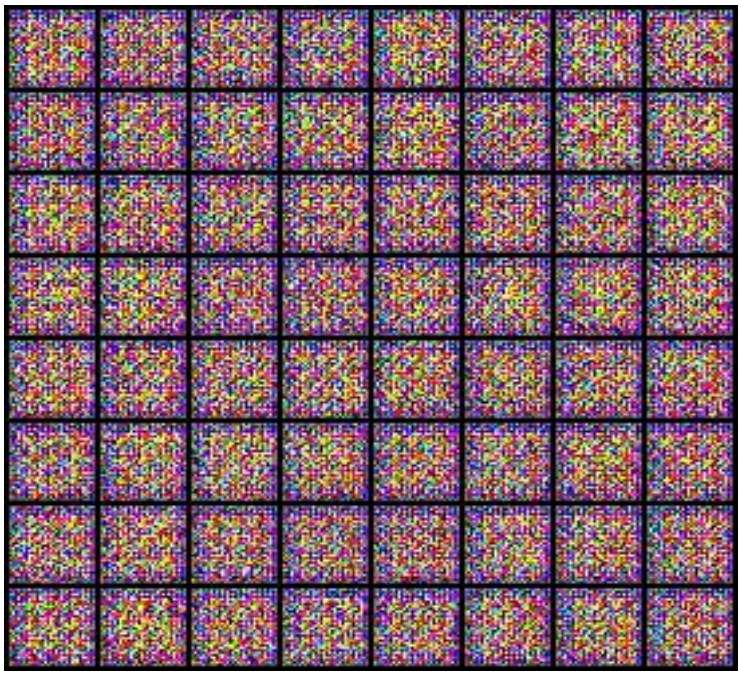


Figure 4.5: WGAN-GP Generated images at Epoch 0.

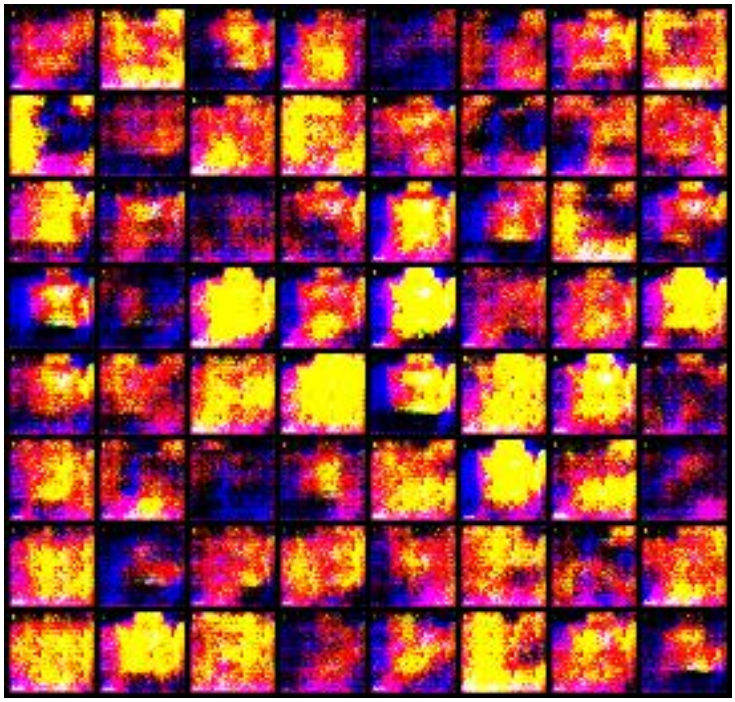


Figure 4.6: WGAN-GP Generated Images at Epoch 100.

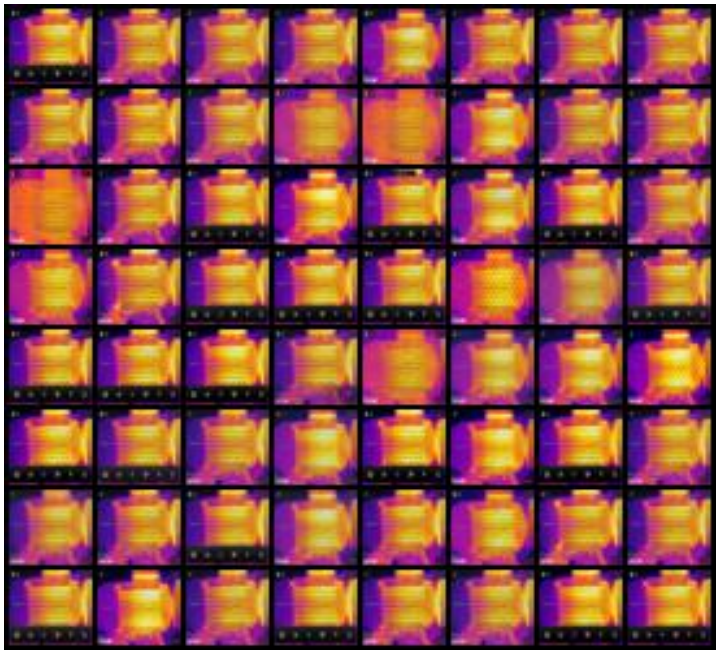


Figure 4.7: WGAN-GP Generated Images at Epoch 10000.

4.2.2 Advanced WGAN-GP and cWGAN-GP

4.2.2.1 Generated Images Similarity Assessment: Visual Quality Assessment

The WGAN-GP and cWGAN-GP approaches yielded promising results in generating realistic thermal images. The cWGAN-GP approach, which incorporates class information (using health condition classes as cWGAN-GP conditions), demonstrated further improvements in image generation, allowing for better control over the generated images. Figure 4.8 showcases generated images using the cWGAN-GP approach, exhibiting a resolution of 128 x 128 and belonging to seven different health condition classes. Each row represents a different health condition class from row one to row seven, representing 8 bars, IRF, ORF, Ball, 4 bars, Normal (fault-free) or Healthy condition, and 1 bar, respectively. Meanwhile, Figure 4.9 presents images generated using the WGAN-GP approach, with a resolution of 128 x 128 and belonging to the Normal (fault-free) or Healthy condition class. Both types of generated images visually demonstrate high variability and closely resemble real motor thermal images. However, additional quantitative assessment is needed.

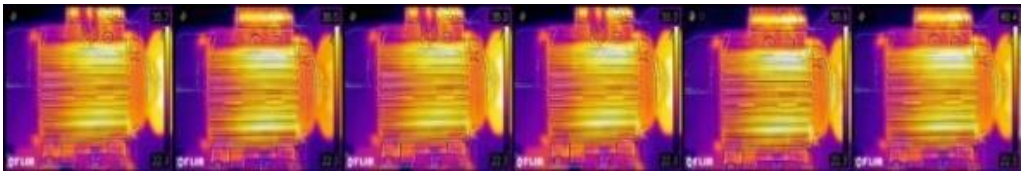


Figure 4.8: Generated Images Class: (Normal (fault-free) or Healthy condition) with Resolution 128 x 128 Using WGAN-GP.

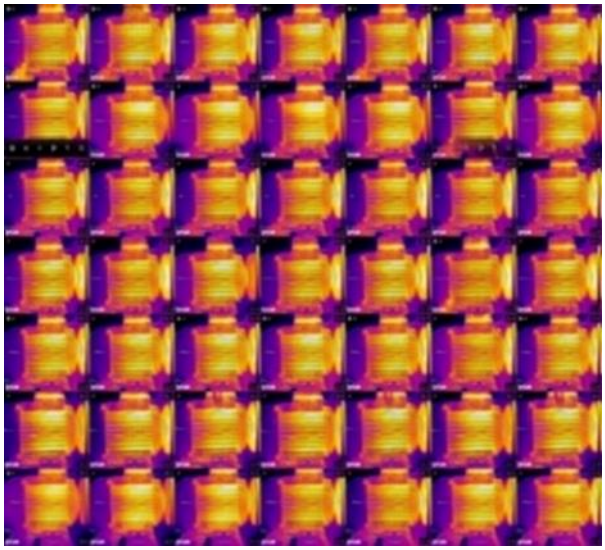


Figure 4.9: Generated Images with Resolution 128 x 128 using cWGAN-GP. Each Row Represents a Different Health Condition Class from Row One to Row Seven, Representing 8 Bars, IRF, ORF, Ball, 4 Bars, Normal (fault-free) or Healthy, and 1 Bar, Respectively.

4.2.2.2 Generated Images Similarity Assessment: GAN Similarity Metrics (MMD, EMD)

Evaluating GAN performance is challenging due to the lack of standardised metrics and the subjectivity of human visual evaluation. However, visual similarity assessment is used with other quantitative metrics (Niu et al., 2020; Shao et al., 2023). This section uses non-visual quantitative metrics to assess image similarity, providing insights into the quality, diversity, and similarity of generated images using GAN compared to real images. The following metrics are used to evaluate the quality and similarity of generated images in the context of GANs:

- Fréchet Inception Distance (FID)

Fréchet Inception Distance (FID) was introduced by Heusel et al. in 2017. It measures the distance between the real distribution and the distribution generated by the trained model (Niu et al., 2020). It is computed using Equation (4.4), where, μ_r and μ_g are the mean value

for real images and generated images, respectively, and C_r and C_g are the covariance of the image features (Niu et al., 2020). A lower FID indicates a better model with images closer to real ones. Hence, FID fits a Gaussian distribution to the hidden activation of InceptionNet for each image set and computes the Fréchet Distance (also known as Wasserstein-2 distance) between the Gaussians (Chen et al., 2020a).

$$FID(P_r, P_g) = \|\mu_r - \mu_g\| + Tr(C_r + C_g - 2(C_r C_g))^{1/2} \quad \textbf{(4.4)}$$

- Maximum Mean Discrepancy (MMD)

Measures the dissimilarities between generated and real images by capturing independent samples from each distribution. It quantifies the distance between the actual and generated distribution, with a lower score indicating better model performance (Pan et al., 2019). Equation (4.5) presents the MMD score using the Gaussian kernel. P_r and P_g represent the real and generated image distribution, respectively, while *x* and *y* are samples drawn from these distributions. The first term captures the similarity of samples within the real distribution P_r, the second term measures the similarity between samples from the real and generated distributions v and P_g, and the third term assesses the similarity of samples within the generated distribution P_g (Borji, 2019).

$$MMD(P_r, P_g) = \mathbb{E}_{x,x' \sim P_r}[k(x, x')] - 2\mathbb{E}_{x,x' \sim P_r, y \sim P_g}[k(x, y)] + \mathbb{E}_{y,y' \sim P_g}[k(y, y')] \quad \textbf{(4.5)}$$

- Earth Mover's Distance (EMD)

Also known as the Wasserstein distance, EMD measures the distance between two probability distributions (Borji, 2019). It represents the minimum amount of work or effort needed to transform one distribution into another (Gao et al., 2020). Equation (4.6) defines EMD, where γ is a transport plan specifying the amount of mass to be transported from each point in P_r to each point in P_g. $\prod(P_r, P_g)$ represents the set of all joint distributions, and $\gamma(x, y)$ indicates the amount of work needed to transform the distributions Pr into Pg from point x to point y. The Wasserstein distance is calculated as the infimum (greatest lower bound) of the expected distance ‖x-y‖ between randomly sampled pairs of points (x, y) from γ (Arjovsky et al., 2017b).

$$W(P_r, P_g) = inf_{\gamma \in \prod(P_r, P_g)} \mathbb{E}_{(x,y) \sim \gamma}[\|x - y\|] \quad \textbf{(4.6)}$$

FID measures image similarities using the Inception network to extract and compare features. However, FID scores can be misleading if the Inception network is biased or mismatched, mainly when working with datasets lacking visual diversity or exhibiting high similarity. In contrast, EMD and MMD are robust metrics that focus on comparing distributions. They allow for evaluating the similarity between generated samples and the real data distribution, even when visual appearances may appear similar to the human eye. EMD measures the distance between probability distributions, while MMD quantifies the distance between data sets, capturing their statistical properties (Heusel et al., 2017; Alqahtani et al., 2019; Borji, 2019; Chen et al., 2020b).

Thermal images present unique challenges for human visual perception, necessitating a comprehensive evaluation. This section compares the WGAN-GP and cWGAN-GP approaches for generating thermal images of IMs under various health conditions. Two evaluation metrics, EMD and MMD, are used to assess the similarity between generated and real images. The experiments were conducted using NVIDIA T4 and NVIDIA V100 GPUs with different training times and epochs. The NVIDIA V100 GPU demonstrated superior performance, processing nearly three times faster than the NVIDIA T4 GPU. All experiments were performed on Google Colab Pro, utilising the allocated GPUs.

Table 4.2 provides a detailed performance comparison of the WGAN-GP and cWGAN-GP approaches. Initially, WGAN-GP trained models for each health condition, starting with a 32 x 32 resolution for the IRF as a baseline. Subsequently, the scope expanded to include four health condition classes. Training duration and epochs varied for each condition, ranging from 18.5 hours for IRF and ORF to 36 hours for 8-bar faults. The training was terminated based on plateaued evaluation metrics and visually acceptable generated images. While training individual models resulted in high-quality 128 x 128 images, it required substantial time, which varies per health condition; for instance, the ORF condition model took more than one day.

In contrast, the cWGAN-GP approach trained all fault types together, reducing overall training time and increasing efficiency. WGAN-GP evaluation indices represent the average of four conditions, while cWGAN-GP evaluation indices represent the average of all conditions. The EMD metric quantifies dissimilarity in terms of spatial alignment and intensity variations. WGAN-GP achieved a lower average EMD score of 4.663 for four

conditions than cWGAN-GP's score of 4.816, indicating a slightly higher similarity between the generated and real images regarding spatial alignment and intensity characteristics.

Table 4.2: Comparison of GPU types, Training Time, Epochs, FID, MMD, EMD, Resolution, Class Name, and Method Used for Generating Synthetic Images.

Method	**Class Name**	**Resolution**	**EMD**	**MMD**	**Epochs**	**TrainingTime (Hours)**	**GPU Type**
WGAN-GP	IRF	32x32	0.32	0.24	5000	4.5	NVIDIA T4
	IRF	128x128	4.64	1.10	10000	12	NVIDIA V100
	Normal		4.70	1.07	5000	18.5	NVIDIA T4
	ORF		4.72	1.10	5000	18.5	NVIDIA T4
	8 bars		4.59	1.04	10000	36	NVIDIA T4
Average (8bars, IRF, ORF, Normal)(128x128)			**4.663**	**1.078**			
cWGAN-GP	IRF	32x32	0.29	0.21	10000	11	NVIDIA T4
	8 bars		0.18	0.59			
	ORF		0.13	0.81			
	Ball		0.18	0.25			
	4 bars		0.15	0.66			
	Normal		0.21	0.30			
	1bar		0.21	0.31			
	All Classes		0.12	0.09			
	IRF	128x128	4.83	1.07	10000	7.25	NVIDIA V100
	8 bars		4.78	1.02			
	ORF		4.74	1.01			
	Ball		4.80	1.26			
	4 bars		4.75	1.08			
	Normal		4.88	0.99			
	1 bar		5.08	1.23			
	All Classes		4.70	1.13			
Average (8bars, IRF, ORF, Normal)(128x128)			**4.816**	**1.023**			

The table includes results for four classes using WGAN-GP and all seven classes using c*WGAN-GP.*

The MMD metric compares the mean feature representations of real and generated image distributions. It was found that the cWGAN-GP obtained a lower MMD score of 1.023, suggesting a better capture of real image characteristics, while WGAN-GP had a slightly higher score of 1.078. Thus, the cWGAN-GP approach outperforms the WGAN-GP approach in capturing the distribution and characteristics of real images. Additionally, the cWGAN-GP approach's advantage lies in training all fault types together, reducing the overall training time and increasing methodology efficiency. Considering the better MMD

similarity scores achieved by the cWGAN-GP approach and its reduced training time, it can be concluded that the cWGAN-GP approach is superior to the WGAN-GP approach in generating thermal images that closely resemble real images of IMs under various health conditions while also being more efficient in terms of training time.

4.2.2.3 Generated Images Similarity Assessment: Pre-Trained AlexNet Classification

This section discusses using pre-trained CNN models for non-visual image similarity assessment. A pre-trained CNN model called AlexNet was used to enhance the evaluation process further, using the Stochastic Gradient Descent with Momentum (SGDM) optimiser and 0.0001 LR with seven classes (health conditions) and 56.8M total learnable parameters. These parameters include weights and biases associated with the layers in the network. By leveraging the knowledge and features learned by AlexNet from large-scale image classification tasks, we can evaluate the generated samples based on their classification accuracy or other relevant metrics. This approach enables us to assess the discriminative capabilities of the generated samples and their alignment with the real data distribution. An artificial dataset was generated using cWGAN-GP, which had 288 images per health condition. The images were first divided randomly into 80.00% training and 20.00% validation and then tested on unseen original lab-collected images. This resulted in 98.41% overall classification accuracy, 98.41% precision, and 98.49% recall. However, the following Table 4.3 shows the accuracy per health condition type as follows:

Table 4.3: Accuracy Per Health Condition Using AlexNet.

8 bars	IRF	ORF	Ball	4 bars	Normal	1 bar
100%	95.83%	100%	100%	93.06 %	100%	100%

4.3 Summary

Part 1 (Section 4.1.1: Foundational Study of Generative Adversarial Networks (GANs)) highlighted the significant impact of hyperparameters on the performance of GANs and successfully demonstrates the remarkable ability of GANs to create thermal images of IMs. This study confirmed the applicability of the WGAN-GP for generating artificial thermal images; however, it requires considerable computational power. The research in Part 2 aims to optimise the dataset creation process by training all fault types simultaneously using conditional GANs at higher image resolutions. The results showed that the WGAN-GP model effectively generated motor images with advanced training parameters and GPUs

that closely resembled the real images in the dataset. However, there remains scope for improvement, particularly in testing other bearing faults.

This part investigated the feasibility of using GANs to create realistic IM thermal RGB image datasets for multimodal condition-monitoring systems. Generating high-quality thermal images presents computational challenges. The current study used two GAN frameworks, DCGAN and WGAN-GP, under different health conditions. Initially, DCGAN was applied to three conditions using various hyperparameters, but the results required further improvement. Subsequently, WGAN-GP was utilised with an extensive training duration of 11 hours, using 10,000 epochs and a batch size 64, targeting the IRF dataset. This resulted in artificial images being generated which closely resembled real images. This study highlights the effect that hyperparameters have on GAN performance. It demonstrates the capability of GANs in creating artificial thermal image datasets, paving the way for further improvement with WGAN-GP in Part 2.

Part 2 (Section 4.1.2: Advanced GANs Framework) explored and compared two frameworks (WGAN-GP and cWGAN-GP) for generating artificial thermal images of IMs with different health conditions. The evaluation process for comparing the similarity between the real images and the artificially created images included visual quality assessment, evaluation using GAN similarity metrics (MMD and EMD), and classification using a pre-trained AlexNet model. Both approaches produced high-quality thermal images resembling real IM faults and thermal images of Normal (fault-free) or Healthy condition when evaluated visually and qualitatively. Quantitatively, the generated images were evaluated using two similarity metrics: EMD and MMD. While WGAN-GP achieved a marginally better EMD score of 4.663 for four conditions compared to cWGAN-GP's score of 4.816 for all conditions, cWGAN-GP recorded a lower MMD score of 1.023, thereby indicating a closer resemblance to real images in terms of statistical properties.

This suggests that the generated images from cWGAN-GP exhibit similar texture, shape and overall distribution as observed in the real images. To further validate the generated images, a pre-trained AlexNet model was utilised for classification on the cWGAN-GP dataset, which achieved an overall classification accuracy of 98.41% and higher accuracy rates for some health conditions.

In conclusion, previous studies have primarily focused on creating artificial image models for each fault type separately, presenting a significant gap in the research. This chapter sought to address this gap using cGANs to simultaneously create artificial thermal images for various health conditions in IMs. By incorporating health conditions as a new input for the cWGAN-GP model, representing the network condition, the cWGAN-GP approach proved superior in generating thermal images that closely resemble real images of IMs under various health conditions. Its ability to integrate class information facilitated faster convergence, enhanced pattern recognition, and greater diversity in image generation. The proposed approach achieved a higher similarity with the MMD score and reduced training time, and demonstrated high classification accuracy on real datasets, thus highlighting its effectiveness and efficiency. These findings contribute significantly to thermal image generation and demonstrate potential applications in motor condition monitoring and fault classification.

Thus, this chapter's main contribution is the generation of a novel dataset of artificial thermal images representing various health conditions of IM and evaluating the effectiveness of GANs in enhancing the accuracy of CNN-based condition monitoring systems. This was achieved through two key segments, each containing specific sub-contributions:

Part 1: Explored the use of DCGAN and WGAN-GP for generating IM fault images from thermal data.

Part 2: Introduced a novel approach using WGAN-GP and cWGAN-GP to generate artificial thermal images of IM faults. Significant contributions of Part 2 in this chapter to the field of IM condition monitoring include the following:

1. Generating synthetic thermal images that represent various health conditions using WGAN-GP.
2. Examining the effectiveness of training individual WGAN-GP models for each health state.
3. Enhancing the quality of the generated images and reducing the training time required by incorporating health state labels through cWGAN-GP.
4. Comparing and contrasting the WGAN-GP and cWGAN-GP approaches through a combined assessment method.

Chapter 5: A Novel Customised Load Adaptive Framework (CLAF) for Induction Motor Fault Classification Utilising the MFPT Bearing Dataset

5.1 Proposed Methodology

The Customised Load Adaptive Framework (CLAF) proposed in this research is a two-phase approach designed to enhance our understanding of how radial loads influence system behaviour, particularly in the presence of faults and varying load conditions. The term ‘Customised’ is used because this framework can be tailored to any dataset; in this study, it is specifically customised for the Machinery Failure Prevention Technology (MFPT) bearing dataset. The term ‘Load Adaptive’ is used because it emphasises and deepens the understanding of how load variations impact Induction Motor (IM) faults, leading to changes in Time and Frequency Domain (TFD) patterns and the identification of load-dependent fault subclasses—'Mild,' 'Moderate,' 'Severe,' and 'Normal (fault-free) or Healthy condition’—through Continuous Wavelet Transform (CWT) energy analysis. This approach focuses on a tailored assessment of load effects and is implemented using MATLAB R2023a.

5.1.1 Time and Frequency Domain (TFD) Load-Dependent Pattern Analysis

Phase 1 unveils load-dependent patterns in varying load conditions, as depicted in Figure 5.1, shedding light on the intricate interplay between load dynamics and bearing fault behaviour through the following steps:

1. Data preprocessing and general load-dependent feature extraction: the MFPT bearing dataset is segmented into smaller, manageable portions, involving the division of the continuous signal into smaller segments stored as separate CSV files.

2. Data segmentation and load-dependent subfile creation: TFD features are extracted from the segmented data, focusing on assessing feature variations during faults and their sensitivity to load changes.

3. TFD feature extraction from data segmentation: generate a load-dependent time and frequency feature set, where an initial load-dependent feature set is created for use in the following step.

4. Significant load-dependent feature selection and validation: Identify and validate the most significant load-dependent features from the time domain, frequency domain, and spectral features using an iterative one-way Analysis of Variance (ANOVA) approach. The selected features are then validated by assessing the accuracy of

various classifiers. This step examines how load variations affect these features. One-way ANOVA, a reliable method for feature reduction, is employed to streamline the analysis by focusing on the most relevant features. (Suresh and Naidu, 2022; Alharbi et al., 2023; Zhang et al., 2023d).

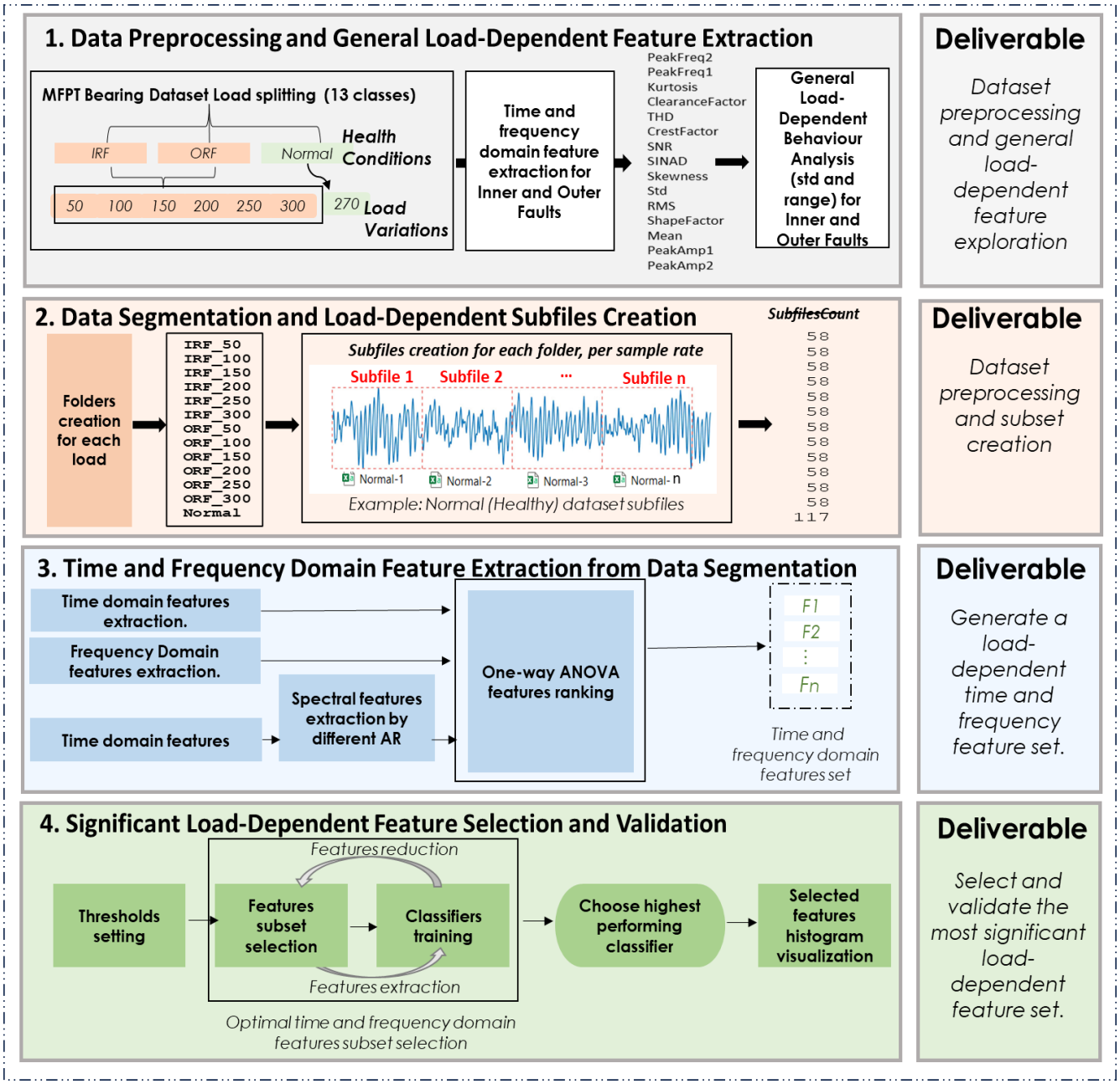


Figure 5.1: Time and Frequency Domain Load-Dependent Pattern Analysis Methodology .

5.1.1.1 ***. Feature Extraction***

Feature extraction operates within three primary domains: temporal, spectral, and time-frequency. These distinct domains serve as tools to capture distinctive aspects of signal behaviour. The section starts with TFD feature extraction and moves to the 2D time-frequency domain features.

1) Time Domain Analysis

Traditional Statistical Features (TSFs) are fundamental measures in the time domain derived from vibration or time series data. The formulas and descriptions of TSFs are presented in Table 5.1 (Liu and Weng, 2019; Pinedo-Sánchez et al., 2020; Shi et al., 2020; Jain and Bhosle, 2021). These features collectively capture the temporal characteristics of signals, enabling the examination of behaviour over time. Analysing vibration signals in the time domain is crucial for understanding signal dynamics and detecting anomalies or faults (Shi et al., 2020).

T**able 5.1:** Traditional Statistical Features (TSFs).

Parameter	Formula	Description
Peak or Max	X_{max}	The highest amplitude value is observed within a given signal or dataset.
Root Mean Square (RMS)	$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i)^2}$	Gives a measure of the magnitude of the signal.
Skewness	$\frac{\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^3}{\left[\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2\right]^{\frac{3}{2}}}$	Measures the asymmetry of the distribution about the mean.
Standard deviation (std)	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i)^2}$	The square root of the variance represents the average deviation from the mean.
Kurtosis	$\frac{\frac{1}{N}\sum_{i=1}^{N}(x_i-\bar{x})^4}{\left[\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2\right]^2}$	Indicates the "tailedness" of the distribution. A high kurtosis might indicate the presence of outliers or impulses in the signal.
Crest Factor	$\frac{Peak}{RMS}$	The peak amplitude ratio to its RMS value indicates the relative sharpness of peaks.
Peak to Peak	$X_{min} - X_{max}$	Difference between the maximum and minimum values of the signal.
Impulse Factor	$\frac{max\lvert X_i\rvert}{\frac{1}{n}\sum_{i=1}^{n}\lvert X_i\rvert}$	Highlights the impulsive behaviours indicative of machinery faults.

In the table, N is the sample size, x_i represents individual data points and $\bar{x}$ is the average data point.

2) Frequency Domain Analysis

Extracting features from the frequency domain offers insights into data's periodic components and harmonic structures, as represented in Table 5.2 (Kumar et al., 2022; Tian et al., 2022; Granados-Lieberman et al., 2023).

Table 5.2: Frequency Domain Features.

Parameter		Formula	Description
Harmonic Features	THD	$\sqrt{\frac{\left(\sum_{i=2}^{N} A_i^2\right)}{A_I}}$	Frequency domain, measuring the distortion caused by harmonics in the signal.
	S/N	$10\,log_{10}\left(\frac{P_{signal}}{P_{noise}}\right)$	Compares the level of a desired signal to the level of background noise.
	SINAD	$10\,log_{10}\left(\frac{P_{signal}}{P_{noise}+P_{distortion}}\right)$	A measure of signal quality compares the level of desired signal to the level of background noise and harmonics.
Spectral Features	Peak amplitude	$\lvert x_{f-peak}\rvert$	Represents the highest point (or peak) of the signal's waveform when viewed in the frequency domain.
	Peak frequency	$f-peak$	Corresponds to the frequency component that is most prominent or dominant in the signal.
	Band power	$\sum_{f-start}^{f-end}\lvert x(f)\rvert^2$	Quantifies the total energy within a specific frequency range, providing insights into the distribution of signal energy across the spectrum.

In the context of frequency domain analysis, A_I is the amplitude of the fundamental frequency, and A_i is the amplitude of the i-th harmonic. For S/N and SINAD calculations, P_{signal} is the signal power, P_{noise} is the noise power, and $P_{distortion}$ is the power of harmonic distortion. Peak amplitude x_{f-peak} is the frequency domain's complex value at bin $f-peak$. $x(f)$ is the complex value at frequency bin 'f', and $|x(f)|^2$ represents its squared magnitude.

Frequency domain analysis of vibration signals involves examining amplitude variations across different frequencies, contributing to a better understanding of vibration behaviour (Ahmed and Nandi, 2018; Shi et al., 2020). Frequency domain features such as Root-Mean-Square Frequency (RMSF), Centre Frequency (CF), and Total Harmonic Distortion (THD) are vital in analysing a signal's power distribution and harmonics (Shi et al., 2020). The Signal-to-Noise Ratio (S/N) and Distortion Ratio (SINAD), expressed in decibels (dB), merge time, and frequency domain aspects, aid in gearbox fault analysis (Kumar et al., 2022).

On the other hand, spectral feature extraction transforms a signal from the time to the frequency domain, revealing its frequency content (MathWorks-7, 2024). In rotating machine fault diagnosis, the Autoregressive (AR) model, especially with the forward–backwards approach, improves classification over traditional methods (Hu and Zhang, 2019; Metwally et al., 2020). This model, effective in bearing diagnosis, isolates noise and fault impulses dependent on the optimal AR order (Djemili et al., 2023). The resulting spectral features from the AR model, such as peak amplitude, peak frequency, band power, and formulas and descriptions, are shown in Table 5.2. The AR model denoted as $AR(P)$, is formulated as in Equation (5.1) (Hu and Zhang, 2019):

$$x[n] = \sum_{p=1}^{p} a_p x[n-p] + e[n] \quad \textbf{(5.1)}$$

where $x[n]$ is the signal's current value, influenced by its past values $x[n-p]$ and AR coefficients a_p, with $e[n]$ as the random noise component (Hu and Zhang, 2019; Silik et al., 2021).

5.1.2 Customised Load Adaptive Framework for IM Bearings Fault Classification

In Phase 2, this research customises explicitly the methodology for the MFPT bearing dataset, focusing on Wavelet Transform and load-dependent fault subclasses, as shown in Figure 5.2. The chapter explored different CWT approaches to find the optimal wavelet function or mother of wavelets, which was determined using Wavelet Singular Entropy (WSE), followed by preprocessing and load effect assessment, resulting in the proposed CLAF. This framework introduced a new dimension to traditional fault classification by considering load variation dataset customisation, revealing load-dependent fault subclasses' signatures, which are absent in conventional approaches:

1. CWT signal encoding and optimal technique selection: Various CWT methods are explored to represent signals concerning fault types, leading to selecting the most appropriate approach (Amor, Bump, or Morse).
2. CWT energy assessment for each Load Factor (LF) involves preprocessing, health condition classification, and categorisation into thirteen classes corresponding to specific load levels. The research calculates WSE and mean energy, providing insights into fault severity and energy distribution.

3. CLAF: the research proposes load-dependent fault subclasses tailored to assess radial load impact under different conditions, incorporating insights from the analysis for a customised evaluation.
4. CLAF Validation: we train different classifiers on proposed load-dependent fault subclasses to examine the classification accuracy of the proposed classes.

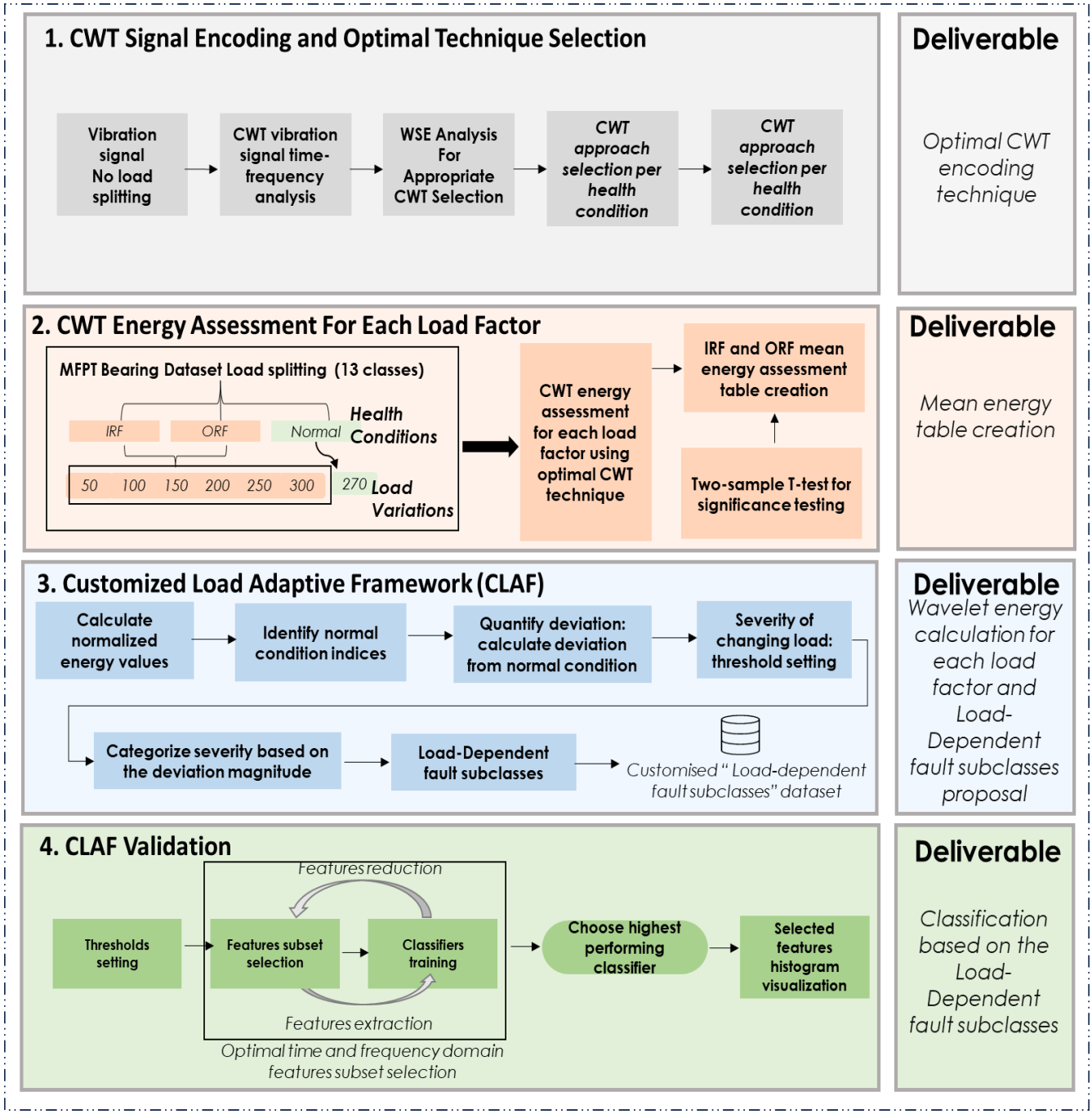


Figure 5.2: Customised Load Adaptive Framework (CLAF).

5.1.1 Dataset

This research comprises two phases, each dedicated to investigating the radial effects of loads under various operational conditions, encompassing both faulty and Normal (fault-free) or Healthy condition utilising the MFPT bearing dataset. The experimental setup for the MFPT bearing dataset involved a test rig equipped with a NICE bearing, including a roller diameter of 0.235 inches, a pitch diameter of 1.245 inches, and eight rolling elements positioned at a contact angle of zero degrees. This setup allowed vibration data to be collected under various loading conditions, accurately replicating both bearings with faults and those without faults for comprehensive fault analysis research. The Normal (fault-free) or Healthy condition (formerly called 'baseline') data were collected under a 270 lb load, with a sampling rate of 97,656 samples per second (SPS) over 6 s. Simultaneously, fault signals originating from Inner Race Defect (IRD) or Inner Race Fault (IRF) and Outer Race Defect (ORD) or Outer Race Fault (ORF) were acquired from the bearing test rig under six different load conditions, 50, 100, 150, 200, 250, and 300 lbs, all while maintaining a constant speed of 25 Hz (Bechhoefer, 2013; Bechhoefer, 2016).

An essential aspect of this study involves categorising the severity of load-dependent fault subclasses within the MFPT bearing dataset. This categorisation is based on changes in wavelet energy compared to Normal (fault-free) or Healthy condition, with a 20.00% increase classified as Mild severity, 20.00% to 50.00% as Moderate severity, and anything exceeding 50.00% as Severe. While acknowledged as an assumption, this categorisation is a fundamental component of the methodology, ensuring a structured and systematic approach to assessing fault severity under varying load scenarios. The following section will present the results obtained from this framework, covering Phase 1 and Phase 2.

5.2 Results and Discussion

5.2.1 Time and Frequency Domain (TFD) Load-Dependent Pattern Analysis

This phase (Phase 1) involves data preprocessing for data preprocessing, general feature extraction, and segmentation and data segmentation for LF subset creation.

5.2.1.1 Step1: Data Preprocessing and General Load-Dependent Feature Extraction

The dataset was categorised for separate analysis to assess the load-dependent impact in fault scenarios, explicitly focusing on IRF, as presented in Table 5.3, and ORF, as shown in Table 5.4. This study involved a comparison of six different LFs (50, 100, 150, 200, 250, and 300 lbs) against the Normal (fault-free) or Healthy condition at LF 270 lbs. The Normal (fault-free) or Healthy condition dataset served as a baseline for comparative analysis, aiding in identifying distinctive features that indicate the presence of a fault in both IRF and ORF datasets.

Table 5.3: IRF Dataset Splitting Per Load.

Inner Race Fault Dataset	Code	LF(lbs/kg)	Sampling Rate(Hz)	Duration (s)
baseline_2	data_normal	270/122.47	97,656	6
InnerRaceFault_vload_2	IRF_50	50/22.68	48,828	3
InnerRaceFault_vload_3	IRF_100	100/45.36	48,828	3
InnerRaceFault_vload_4	IRF_150	150/68.04	48,828	3
InnerRaceFault_vload_5	IRF_200	200/90.72	48,828	3
InnerRaceFault_vload_6	IRF_250	250/113.40	48,828	3
InnerRaceFault_vload_7	IRF_300	300/136.08	48,828	3

Table 5.4: ORF Dataset Splitting Per Load.

Outer Race Fault Dataset	Code	LF(lbs/kg)	Sampling Rate(Hz)	Duration (s)
baseline_2	data_normal	270/122.47	97,656	6
OuterRaceFault_vload_2	ORF_50	50/22.68	48,828	3
OuterRaceFault_vload_3	ORF_100	100/45.36	48,828	3
OuterRaceFault_vload_4	ORF_150	150/68.04	48,828	3
OuterRaceFault_vload_5	ORF_200	200/90.72	48,828	3
OuterRaceFault_vload_6	ORF_250	250/113.40	48,828	3
OuterRaceFault_vload_7	ORF_300	300/136.08	48,828	3

- General Load-Dependent Behaviour Analysis

This study conducted general TFD feature extraction, resulting in 13 features for IRF (Table 5.5) and ORF (Table 5.6). Additionally, spectral features were extracted using an AR model with an order of 15, focusing on two significant resonant peaks in the frequency spectrum and providing five additional load-dependent feature patterns, as detailed in Table 5.5. Key findings regarding the impact of changing the radial load on these extracted features are as follows: Firstly, the Clearance Factor (CF) exhibited a noticeable decrease with increasing radial loads for both IRF and ORF. Specifically, IRF decreased by 12.10% (from 40.04 at load 50 to 35.24 at load 300), while ORF experienced a decrease of about 68.00%

(from 10.26 at LF 50 to 27.18 at LF 300). Secondly, the Crest Factor consistently decreased with higher radial loads, showing a decrease of approximately 16.00% for IRF (from 15.46 at LF 50 to 12.99 at LF 300) and a comparable reduction of roughly 50.60% for ORF (from 6.39 at LF 50 to 12.92 at LF 300). Mean values significantly increased, with higher radial loads for IRF and ORF. IRF exhibited an increase of approximately 10.90% in its peak value (from 23.06 at LF 150 to 25.585 at LF 300), while ORF showed a substantial increase of about 294.90% in its peak value(from 4.93 at LF 100 to 19.43 at LF 300).

Table 5.5: General Time and Frequency Domain Features (IRF).

LF (lbs)	CF	Crest Factor	Impulse Factor	Kurtosis	Mean	Peak Value	RMS	Shape Factor	Skewness	Std	SINAD *	S/N *	THD *
50	40.04	15.46	28.69	27.97	−0.22	27.50	1.78	1.86	0.62	1.76	−21.32	−21.31	−5.36
100	37.30	14.49	26.96	30.53	−0.22	26.59	1.84	1.86	0.87	1.82	−21.05	−21.03	−0.53
150	33.30	13.25	24.31	33.13	−0.22	23.06	1.74	1.84	1.28	1.72	−19.05	−19.05	−10.06
200	38.15	13.54	26.92	37.28	−0.21	27.38	2.02	1.99	1.15	2.01	−18.22	−18.21	−6.31
250	37.52	13.02	26.18	37.49	−0.20	27.14	2.08	2.01	0.72	2.08	−17.70	−17.68	−5.46
300	35.24	12.99	25.17	35.30	−0.19	25.58	1.97	1.94	0.68	1.96	−17.35	−17.34	−8.41
270 **	7.75	5.230	6.56	3.02	−0.14	4.65	0.89	1.25	0.00	0.88	−23.60	−23.60	−11.39

** Frequency domain features. ** Normal (fault-free) or Healthy condition .*

Table 5.6: General Time and Frequency Domain Features (ORF).

LF (lbs)	CF	Crest Factor	Impulse Factor	Kurtosis	Mean	Peak Value	RMS	Shape Factor	Skewness	Std	SINAD *	S/N *	THD *
50	10.26	6.39	8.48	5.09	−0.19	6.35	0.99	1.33	0.04	0.98	−14.41	−14.40	−11.97
100	9.15	5.84	7.62	4.40	−0.18	4.93	0.84	1.31	−0.01	0.82	−13.15	−13.12	−9.06
150	9.54	6.10	7.94	4.04	−0.18	5.21	0.85	1.30	−0.04	0.83	−12.59	−12.56	−9.934
200	21.81	12.46	17.67	11.90	−0.17	12.28	0.99	1.42	0.31	0.97	−17.54	−17.52	−5.54
250	15.03	9.07	12.30	6.59	−0.16	8.66	0.96	1.36	0.12	0.94	−16.09	−16.06	−4.92
300	27.18	12.92	20.80	17.69	−0.16	19.43	1.50	1.61	0.27	1.50	−15.10	−15.10	−14.69
270 **	7.75	5.23	6.56	3.02	−0.14	4.65	0.89	1.25	0.01	0.88	−23.60	−23.60	−11.39

** Frequency domain features. ** Normal (fault-free) or Healthy condition.*

In Table 5.7, variations in peak amplitudes (PeakAmp1 and PeakAmp2), peak frequencies (PeakFreq1 and PeakFreq2), and Band Power for both IRF and ORF across a range of LFs (from 50 to 300 lbs) were observed. Notably, with increasing radial load, IRF exhibited higher peak amplitudes at 300lbs compared to ORF, while their peak frequencies tended to converge. Furthermore, Band Power showed a more pronounced rise as LF increased, especially for IRF, underscoring its sensitivity to LF variations. When compared to the reference condition at LF of 270, we observed significant differences in peak amplitudes and frequencies, highlighting the discernible impact of varying loads on fault characteristics.

Table 5.7: Spectral Features by AR Model (IRF and ORF).

LF	**PeakAmp1**		**PeakAmp2**		**PeakFreq1**		**PeakFreq2**		**BandPower**	
(lbs)	**IRF**	**ORF**	**IRF**	**ORF**	**IRF**	**ORF**	**IRF**	**ORF**	**IRF**	**ORF**
50	0.00034	0.000109	0.00031	0.000093	4363.937	1413.267	13,991.090	14,179.042	1.474	0.454
100	0.00046	0.000075	0.00012	0.000028	4256.059	1379.739	13,968.668	14,258.280	1.476	0.322
150	0.00046	0.000080	0.00005	0.000036	4191.394	1377.111	14,127.206	14,462.995	1.330	0.327
200	0.00031	0.000063	0.00011	0.000058	4025.383	4947.698	10,622.786	1391.188	1.663	0.461
250	0.00061	0.000058	0.00009	0.000049	4124.988	1621.552	10,365.553	5212.034	1.807	0.430
300	0.00077	0.000302	0.00058	0.000296	4081.332	2915.517	748.668	11,675.566	1.618	1.101
Normal 270	0.00003	0.000028	0.00003	0.000028	5490.855	5490.855	14,478.764	14,478.764	0.279	0.302

Further exploration is needed to fully understand the nuanced impact of each LF through detailed feature extraction, as represented in Table 5.8. Analysing standard deviation (Std) and range across various features revealed distinctions between IRF and ORF.

In the frequency domain, PeakFreq1 and PeakFreq2 show notable variability, with IRF having lower variability in PeakFreq1 (510.38 vs. 1788.4) compared to ORF. Regarding impulse characteristics, IRF exhibits higher variability in ImpulseFactor (7.6174 vs. 5.5733), indicating diverse impulse characteristics compared to ORF. ClearanceFactor exhibits more significant variability for IRF (11.237 vs. 7.4289), indicating significant changes in mechanical conditions. Vibration amplitudes also vary, with IRF showing higher variability in PeakValue (8.2942 vs. 5.4215). Additionally, IRF features display more pronounced changes in vibration characteristics compared to ORF, as seen in kurtosis (12.08 vs 5.3444), Skewness (0.41466 vs 0.13983), Std (0.40479 vs 0.23206), RMS (0.40468 vs 0.22898), and

ShapeFactor (0.25877 vs 0.11854). Signal quality parameters (S/N and SINAD) vary more in ORF, indicating alterations in signal-to-noise characteristics. These insights contribute to a comprehensive understanding of vibration signals' dynamic response to IRF and ORF conditions, aiding condition monitoring and load-dependent behaviour analysis for fault detection.

Table 5.8: Std and Range of Time And Frequency Domain Extracted Features for IRF and ORF.

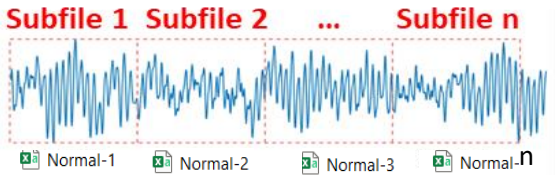
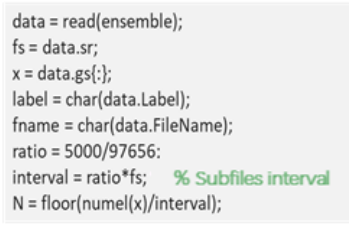
Feature	IRF		ORF	
	Std	**Range**	**Std**	**Range**
PeakFreq2	4916.5	13730	5336.1	13088
PeakFreq1	510.38	1465.5	1788.4	4113.7
Kurtosis	12.08	34.47	7.4289	19.431
ClearanceFactor	11.237	32.294	5.5733	14.24
PeakValue	8.2942	22.848	5.4215	14.783
ImpulseFactor	7.6174	22.127	5.3444	14.669
THD	3.5994	10.863	3.7308	11.036
CrestFactor	3.3591	10.232	3.7247	11.013
S/N	2.3024	6.2569	3.5022	9.7777
SINAD	2.2989	6.2471	3.2456	7.6881
Skewness	0.41466	1.2782	0.23206	0.67179
Std	0.40479	1.1962	0.22898	0.66026
RMS	0.40468	1.1953	0.13983	0.3453
ShapeFactor	0.25877	0.75648	0.11854	0.35602
Mean	3.00 x 10-2	8.59 x 10-2	1.73 x 10-2	5.11 x 10-2
PeakAmp1	2.36 x 10-4	7.44 x 10-4	9.63 x 10-5	2.69 x 10-4
PeakAmp2	1.99 x 10-4	5.57 x 10-4	9.15 x 10-5	2.74 x 10-4

5.2.1.2 Step2: Data Segmentation and Load-Dependent Subfile Creation

First, the dataset was categorised by Normal (fault-free) or Healthy condition and fault types, each corresponding to LF of 50, 100, 150, 200, 250, and 300 lbs. Then, based on different sampling rates, the Normal (fault-free) or Healthy condition baseline signals were differentiated from fault signals: IRF and ORF. The Normal (fault-free) or Healthy condition baseline signals were captured at 97,656 SPS for 6 s, while fault signals were sampled at 48,828 SPS for 3 s. Subfiles were created to enhance statistical robustness, each containing 2,500 vibration data points. This led to 117 subfiles for the Normal (fault-free) or Healthy condition baseline and 58 for each fault category (IRF and ORF), strengthening the sample size and signal integrity; see Table 5.9. Such meticulous preparation establishes a solid foundation for the subsequent one-way ANOVA analysis, enabling the identification of

significant variations in vibration signals linked to different LF levels and fault occurrences *(APPENDIX 2)*.

Table 5.9: Dataset Segmentation and Subfiles Creation Demonstration.

Dataset Segmentation	**CSV Files**	**Code**	**LF**	**Subfiles Count**
Example on baseline (Normal (fault-free) or Healthy condition) with MATLAB code. The segment is based on ratio, i.e., each segment in IRF and ORF contains 2500 samples, and each sample in Normal conditions contains 5000 data points.  	IRF_50	IRF_50	{'IRF−50'}	58
	IRF_100	IRF_100	{'IRF−100'}	58
	IRF_150	IRF_150	{'IRF−150'}	58
	IRF_200	IRF_200	{'IRF−200'}	58
	IRF_250	IRF_250	{'IRF−250'}	58
	IRF_300	IRF_300	{'IRF−300'}	58
	Normal	Normal	{'Normal'}	117
	ORF_50	ORF_50	{'ORF−50'}	58
	ORF_100	ORF_100	{'ORF−100'}	58
	ORF_150	ORF_150	{'ORF−150'}	58
	ORF_200	ORF_200	{'ORF−200'}	58
	ORF_250	ORF_250	{'ORF−250'}	58
	ORF_300	ORF_300	{'ORF−300'}	58

5.2.1.3 Step3: Time and Frequency Domain Feature Extraction from Data Segmentation

Section 5.1.1 discussed the impact of LF variations on features. In this stage, we generate load-dependent time and frequency features from Table 5.9 subfiles for IRF, ORF, and Normal (fault-free) or Healthy condition. This allows for detailed analysis and subsequent one-way ANOVA feature ranking.

First, ten time-domain features, namely, Shape Factor, Peak Value, Clearance Factor, Impulse Factor, Mean, Crest Factor, Kurtosis, RMS, standard deviation (Std), and Skewness, were extracted. Second, there were three general frequency domain features: SINAD, S/N, and THD. Third, AR model estimation was applied to transform the time domain signal into the frequency domain to extract specific spectral features: peak amplitude, peak frequency, and Band Power.

This research explored two AR models for spectral feature extraction: one of order two with a single peak (Figure 5.3a) and another of order fifteen with five peaks (Figure 5.3b). This strategic approach aimed to unravel how the complexity of modelling influences the representation of frequency components in the signal. The order-two model, being

simpler, offers a foundational perspective, capturing fundamental frequency components. These features are extracted within a smaller frequency band of 600–18,000 Hz, excluding peaks beyond 18,000 Hz. On the other hand, the order-fifteen model, with its higher complexity, aspires to provide a more detailed and nuanced representation of intricate frequency variations. Here, feature extraction focuses on a smaller band of frequencies between 10,000–25,000 Hz, excluding peaks after 25,000 Hz. Five spectral peaks were extracted for each signal, generating five frequency features for each peak.

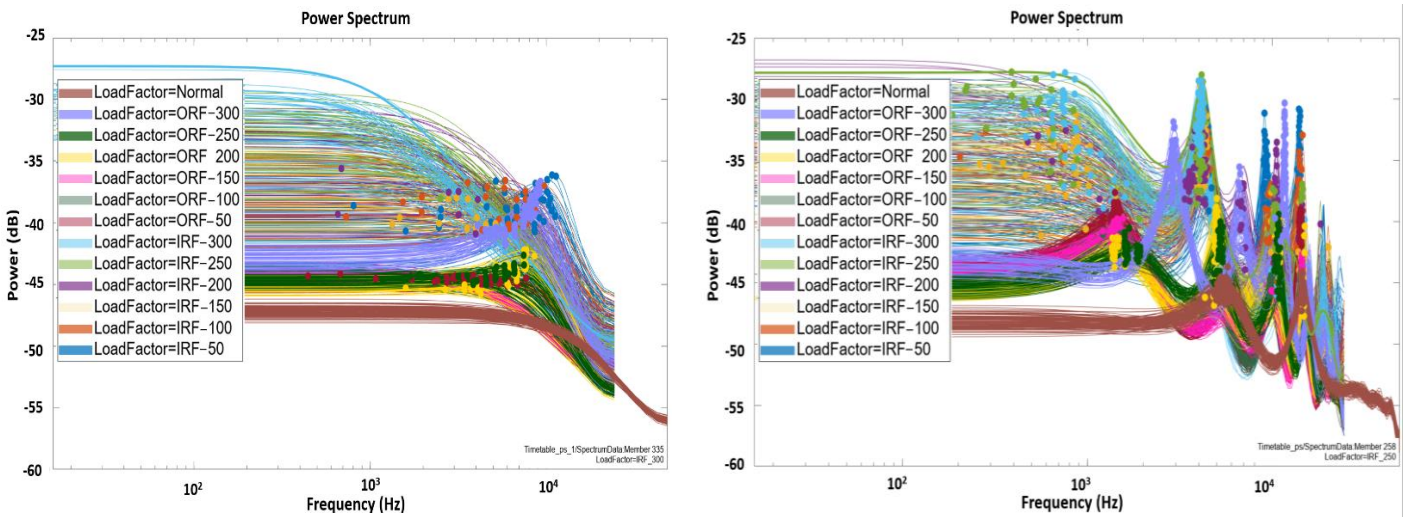


Figure 5.3: AR Model: (a) Order Two and Peak = 1; (b) Order Fifteen and Peak = 5.

The first AR model added three extra features to the 13 TFD features. Conversely, the second AR model generated a more extensive set of 24 features, including general TFD features and 11 features derived explicitly from the AR model. The disparity in feature count resulted primarily from variations in the extracted frequency domain features. When testing different AR models, the decision to calculate peak amplitude and peak frequency for each peak aimed to achieve a more detailed and adaptable analysis of the signal's spectral characteristics. This approach acknowledges variations in frequency modes captured by different models, facilitating the identification and individual analysis of each peak.

This exploration assesses the trade-off between model simplicity and accuracy, a crucial consideration for fault classification. Furthermore, testing different peak configurations allows for a nuanced understanding of how the chosen models identify and distinguish peaks in the frequency spectrum. In essence, this approach yields valuable

insights into the suitability of various model configurations for capturing the diverse characteristics of the signal under investigation.

5.2.1.4 Step 4: Significant Load-Dependent Feature Selection and Validation

Diverse classifier algorithms were systematically examined, focusing on optimal accuracy and minimal confusion. AR models with different peak counts were explored, with the first model (order two, peak one) and the second model (order fifteen, peak five) achieving the highest performance. Subsequently, the dataset was split into testing (20.00%), validation (20.00%), and training (60.00%) subsets, with five-fold cross-validation for test accuracy comparison. Feature richness varied with peak counts, where the first model showcased robust performance with a single peak, emphasising the power of a strategically selected minimal feature. The second model, with five peaks, offered a more detailed representation of spectral characteristics. Features scoring below 20 one-way ANOVA scores were excluded, refining the selection based on substantial impact. This step highlighted load-changing trends on extracted features, providing valuable insights into load impact during faults. Key steps include feature subset selection, classifier training, and selecting the highest-performing classifier with the optimal feature set. One-way ANOVA was employed to determine statistically significant variations in feature values across LF, aligning with the project's aim to analyse LF influences comprehensively. ANOVA ranking was used to systematically rank features based on their significance in distinguishing fault types. The values associated with ANOVA ranking represent the effectiveness of each feature in differentiating between groups in vibration signal data.

(a) Autoregressive Model: Order Two, Peak = 1

Features in the first AR model were reduced based on their ANOVA scores, with lower-scoring features removed first, as shown in Table 5.10. The initial set of the top 13 features had ANOVA scores higher than 20, forming the baseline for further reduction. From this set, the top eight features with ANOVA scores greater than 350 were retained, followed by a subset of the top seven features, each with a score exceeding 370. Additionally, the two highest-scoring features, with ANOVA scores above 600, were selected. These feature sets were designed to investigate the impact of various combinations on classification accuracy, providing insights into the relationship between feature selection and model performance. However, Table 5.11 explores classifier performance across these different feature selection

thresholds, offering notable insights. With the top 13 features (ANOVA scores > 20), the Boosted Trees classifier demonstrated superior adaptability, achieving the highest accuracy at 74.10%, highlighting the discriminative power of the selected features. Reducing the feature set to the top eight (ANOVA scores > 350) and top seven (ANOVA scores > 370) resulted in a trade-off between feature reduction and accuracy, with Boosted Trees still maintaining a competitive edge. However, the drastic reduction to only two features (ANOVA scores > 600) significantly affected accuracy across all classifiers, particularly impacting the Fine Gaussian Support Vector Machine (SVM). Interestingly, increasing the feature count to 629 did not proportionally improve performance, suggesting a saturation point beyond which additional features may introduce noise. These findings highlight the nuanced relationship between feature selection and classifier performance, with Boosted Trees exhibiting robustness across various feature sets.

Table 5.10: One-way ANOVA Ranking Including Spectral Features Extracted By AR Model (a) Order Two, Peak = 1.

Feature Rank	One-way ANOVA Score	Feature Rank	One-way ANOVA Score
1. ShapeFactor	638.7770	9. Std	344.1456
2. PeakValue	629.7172	10. BandPower	215.4163
3. ClearanceFactor	583.7172	11. Skewness	61.1082
4. ImpulseFactor	539.9968	12. PeakAmp1	50.9148
5. Mean	451.5449	13. PeakFrequency1	43.5724
6. CrestFactor	380.0333	14. SINAD	19.2070
7. Kurtosis	373.6953	15. S/N	19.1580
8. RMS	345.8699	16. THD	0

Table 5.11: Classifier Performance Across Feature Selection Thresholds For AR Model (a) and Peak = 1.

No. of Features Used in Classifier Training	Classifier	Test Accuracy
Top 13 (ANOVA scores > 20)	Boosted Trees	74.10%
Top 8 (ANOVA scores > 345)	Narrow Neural Network	72.80%
Top 7 (ANOVA scores > 373)	Bi-layered Neural Network	73.50%
Top 2 (ANOVA scores > 629)	Fine Gaussian SVM	59.90%

(b) Autoregressive Model: Order Fifteen, Peak = 5

In the context of the second AR model, applying the one-way ANOVA Rank generated 24 spectral features, a notable increase from the initial 16; see Table 5.12. These spectral features, which include time domain features like SINAD and S/N, alongside the frequency domain feature peakfrequency2, contribute to a comprehensive feature set. The top 19

features (ANOVA scores > 20) chosen for classifier training were exported to the classification learner, reserving 20.00% of the data for testing.

Table 5.12: One-way ANOVA Ranking Including Spectral Features Extracted by AR Model (b) Order Fifteen, Peak = 5.

Feature Rank	One-way ANOVA Score	Feature Rank	One-way ANOVA Score
1.ShapeFactor	638.7770	13. PeakAmp2	129.8349
2. PeakValue	629.7172	14. PeakFreq3	72.5228
3. ClearanceFactor	583.5995	15. PeakAmp3	68.3653
4. ImpulseFactor	539.9968	16. Skewness	61.1082
5. Mean	451.5449	17. PeakAmp5	49.5351
6. CrestFactor	380.0333	18. PeakFreq1	45.4170
7. Kurtosis	373.6953	19. PeakFreq5	38.6599
8. RMS	345.8699	20. SINAD	19.2070
9. Std	344.1456	21. S/N	19.1580
10. BandPower	263.0314	22. PeakFreq2	18.4450
11. PeakAmp1	171.6077	23. PeakAmp4	14.1606
12. PeakFreq4	162.5469	24. THD	0

The second AR model (Order 15) and peak five features exhibit compelling insights into classifier performance across distinct feature selection thresholds; see Table 5.13.

Table 5.13: Classifier Performance Across Feature Selection Thresholds for AR Model (b) Order Fifteen, Peak = 5.

Number of Selected Features from ANOVA Ranking	Classifier	Test Accuracy
Top 19 (ANOVA scores > 20)	Bagged Trees	86.40%
Top 14 (ANOVA scores > 72)	CubicSVM	86.40%
Top 13 (ANOVA scores > 129)	Quadratic SVM	83.30%
Top 11 (ANOVA scores > 171)	Quadratic Discriminant	84.60%
Top 8 (ANOVA scores > 345)	Quadratic SVM	76.50%

Utilising the top 19 features, Bagged Trees and Cubic Support Vector Machine (CubicSVM) achieved remarkable accuracy scores of 86.40%, underlining the efficacy of these classifiers in leveraging a relatively more extensive set of features. The reduction to the top 14 features (ANOVA scores > 72) maintained high accuracy across all classifiers, emphasising their robustness. Notably, even with a more stringent selection of 14 features, all classifiers sustained accuracy levels above 80.00%, indicating resilience to feature reduction. The decrease to the top 13, 11, and 8 features demonstrated a nuanced trade-off between feature reduction and accuracy, with Bagged Trees consistently leading in performance. The findings reinforce the adaptability of the classifiers to varying feature sets,

providing valuable insights for future considerations in feature selection strategies for this AR model and peak feature combination.

The effectiveness of a classifier depends heavily on the chosen features, showing a delicate balance between feature quantity and classification accuracy. Simply adding more features can sometimes reduce performance because of overfitting. Therefore, features with high ANOVA scores are preferable for training a Machine Learning (ML) model, as they are more likely to enhance accuracy. Moreover, different classifiers exhibit varied sensitivities to feature selection, with some performing well with a concise set of informative features while others benefit from a more extensive feature set. In the context of the AR model, considering the number of peaks proves crucial. Utilising multiple peaks enhances sensitivity to changes in spectral composition, accommodates the potential introduction of new peaks, and furnishes a fine-grained feature set that adeptly captures the distinct contribution of each frequency component.

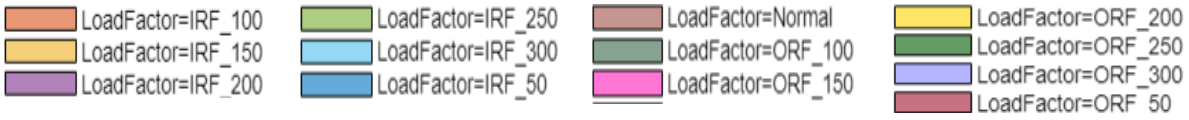
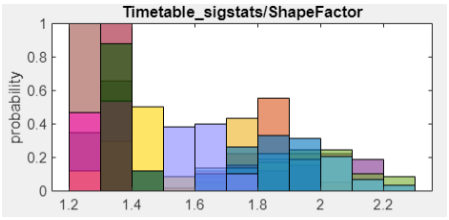
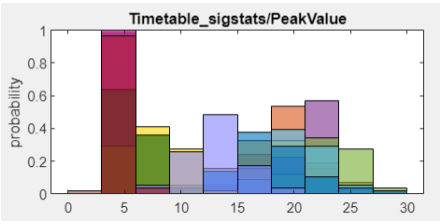
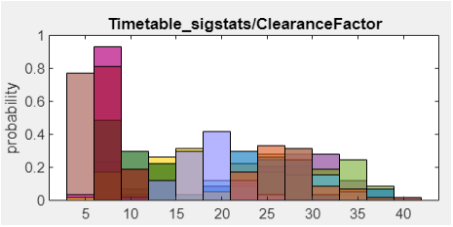
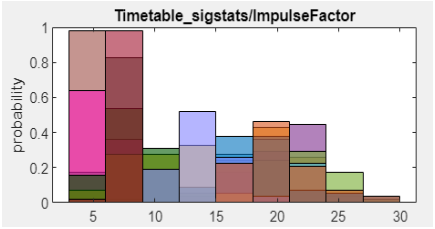
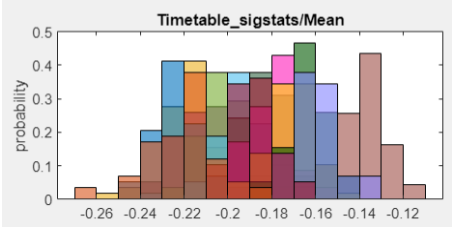
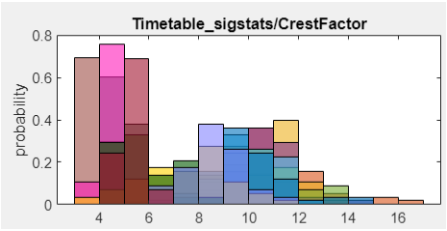
- Summary of Selected Features

The 86.40% accuracy of the test dataset is credited to 14 features derived from an AR model (order 15, peak = 5), covering the time domain, frequency domain, and spectral categories. These features, such as shape factor, peak value, clearance factor, impulse factor, mean, crest factor, kurtosis, RMS, standard deviation (Std), band power, and various peak amplitudes and frequencies, are distinctly represented through a histogram colour scheme (Table 5.14). The LF Colour Code Legend aids in differentiating LF associated with IRF, ORF, and Normal (fault-free) or Healthy condition. Out of 24 features, these 14 were selected for their superior class discrimination ability.

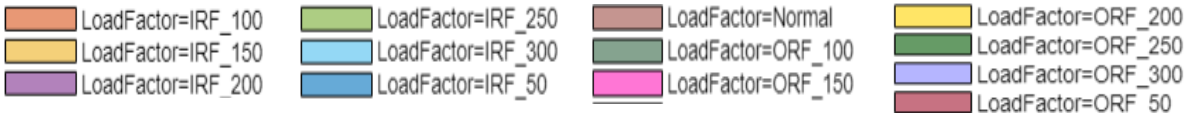
The colour coding in the histograms is crucial for demonstrating the distribution of these features and their impact on the Bagged Trees classifier's accuracy. Specific colours indicate intense feature discrimination for certain LFs. For example, the shape factor histogram separates the IRF_300 LF (purple colour), the peak value excels in distinguishing the IRF_250 class (light green), the clearance factor is more effective for the Normal (fault-free) or Healthy condition, and the impulse factor better identifies the ORF_150 class. This indicates the necessity of a collection of features with varied segregation capabilities for adequate classification.

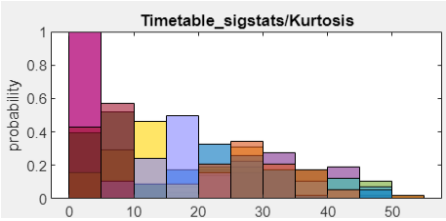
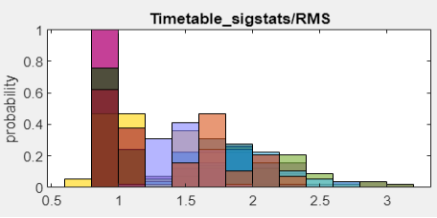
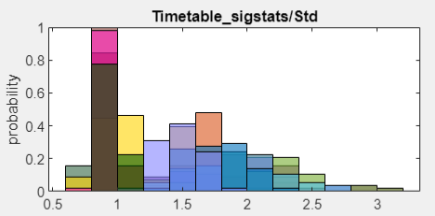
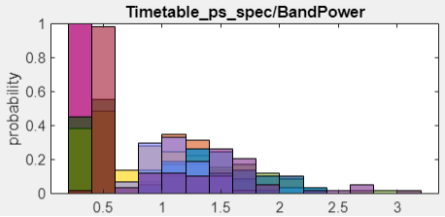
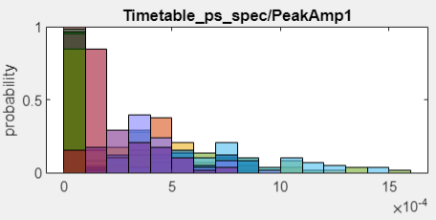
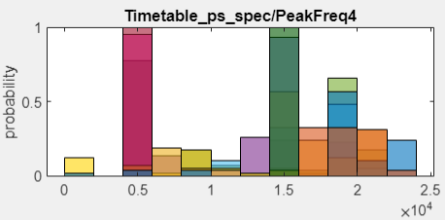
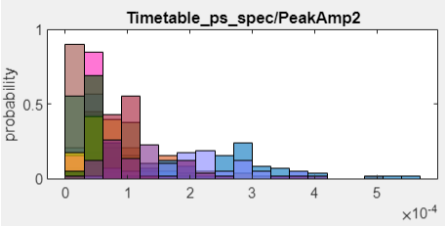
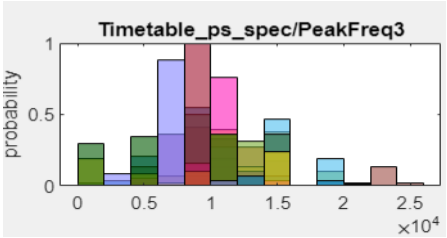
Table 5.12's one-way ANOVA ranking is essential in this context, pinpointing features that accurately differentiate between LFs and assisting in selecting an optimal feature subset for classifier training. This systematic approach is validated by classification accuracy, confirming the chosen features' ability to identify specific LFs precisely under various conditions.

Table 5.14: Top 14 Selected Features Distinguishing Load-Dependent Fault Types: A Histogram Visualisation.

Load Factor Colour Code Legend for the Top 14 Features Ranked by One-Way ANOVA			
			
Feature (ANOVA Rank)	**Feature Histogram**	**Feature (ANOVA Rank)**	**Feature Histogram**
1. Shape Factor		2. Peak Value	
3. Clearance Factor		4. Impulse Factor	
5.Mean		6.Crest Factor	

Load Factor Colour Code Legend for the Top 14 Features Ranked by One-Way ANOVA



Feature (ANOVA Rank)	Feature Histogram	Feature (ANOVA Rank)	Feature Histogram
7. Kurtosis		8.RMS	
9.Standard deviation		10.Band Power	
11.Peak Amplitude1		12.Peak Frequency4	
13.Peak Amplitude 2		14.Peak Frequency3	

5.2.2 Customised Load Adaptive Framework for IM Bearings Fault Classification

This phase (Phase 2) delves into time-frequency feature analysis for different fault types, focusing on the CWT applied to vibration signals with various mother wavelets. The best wavelet function or mother of wavelets was identified using WSE, which aided in developing the CLAF for the MFPT bearing dataset.

5.2.2.1 Step1: CWT Signal Encoding and Optimal Technique Selection

This step involved determining the optimal CWT mother wavelet approach for the MFPT bearing dataset using CWT Time–Frequency Diagrams and WSE, enabling effective feature extraction, denoising, and pattern recognition.

Time-frequency domain analysis, crucial for understanding non-stationary data, merges time and frequency data to examine signal frequency over time intervals (He et al., 2010). Techniques like the Wavelet Transform (WT), using mother wavelets like Amor, Bump, and Morse, are vital in localising frequency information in time (Zhang et al., 2022c). The CWT and WSE are especially effective in damage detection. (Silik et al., 2021). CWT offers a two-dimensional (2D) view of the signal across time and frequency (Kaji et al., 2020). Meanwhile, WSE, derived from wavelet singular values, quantifies signal complexity (He et al., 2010; Tian et al., 2022). CWT is mathematically expressed as in Equation (5.2), with coefficients indicating the wavelet's scale and position as represented in Equation (5.3) (Amanollah et al., 2023):

$$WT_f(a,\tau) = \left(\frac{1}{\sqrt{a}}\right) \int f(t)\bar{\varphi}\left(\frac{t-\tau}{a}\right) dt, \quad \textbf{(5.2)}$$

$$WT_f(a,\tau)f(t) = f(t) * \varphi_{(a,\tau)}(t), \quad \textbf{(5.3)}$$

where $WT_f(a,\tau)$ denotes the wavelet coefficient at a specific scale, a, and position, τ. The term a is the scaling factor that instead stretches or compresses the wavelet, while τ is the translation factor that shifts the wavelet along the signal's time axis. The function φ represents the scaled and translated versions of the mother wavelet. Different mother wavelets yield distinct wavelet coefficients, highlighting varied facets of the signal (Amanollah et al., 2023).

On the other hand, WSE is calculated based on the singular values obtained from the WT of the signal. It reflects the uncertainty of the energy distribution of the characteristic

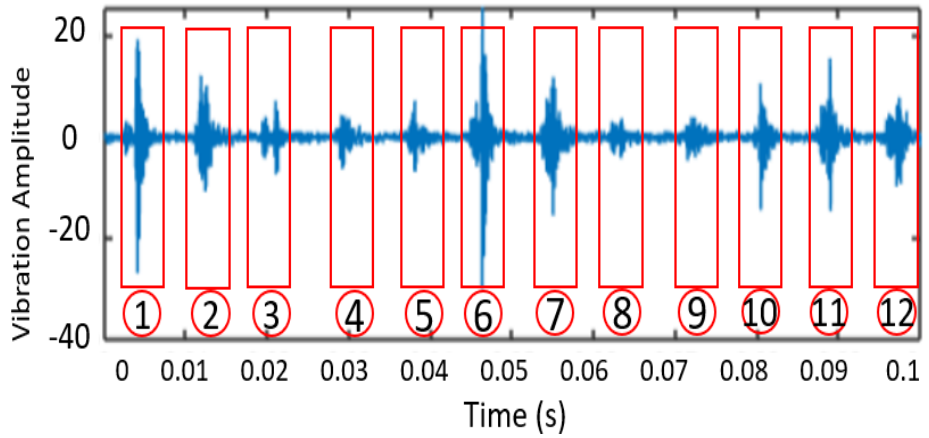
mode of the analysed signal. A smaller WSE indicates a more straightforward and concentrated energy distribution, while a higher WSE suggests a more complex and dispersed energy distribution. The singular values are non-negative and arranged in descending order. The WSE can be defined as represented in Equation (5.4) (He et al., 2010):

$$WSEk = -\sum(\lambda i/\Sigma\lambda i)\log(\lambda i/\Sigma\lambda i), \quad \textbf{(5.4)}$$

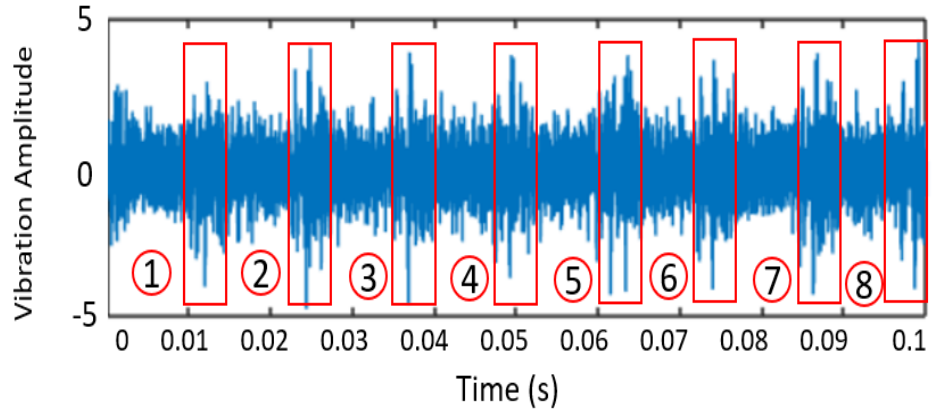
where λi denotes the i-th singular value from the WT, representing the magnitude of coefficients in the analysis. The sum $\Sigma\lambda i$ is the total of all singular values, providing a normalisation factor. The logarithmic component, $\log(\lambda i/\Sigma\lambda i)$, calculates the entropy, thus capturing the distribution complexity of the signal's energy (Zhang et al., 2022c).

- CWT Vibration Signal Time-Frequency Analysis

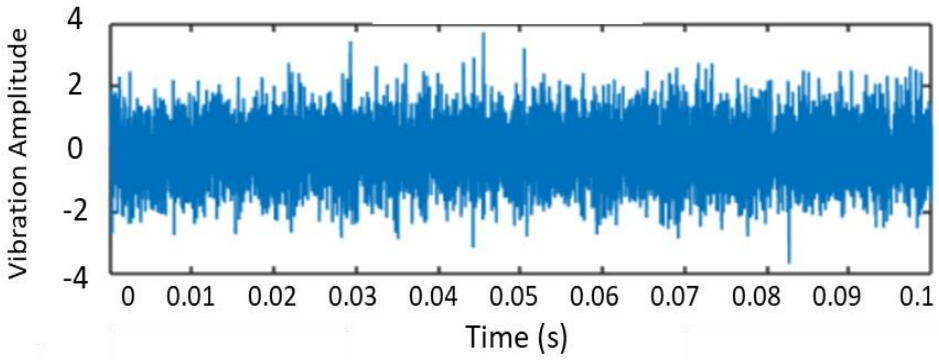
The analysis was initiated with the original MFPT bearing dataset and categorised into IRF, ORF, and Normal (fault-free) or Healthy condition. The objective was to evaluate the capability of CWT in fault recognition, given its suitability for time-frequency analysis. CWT generates wavelet scalograms and 2D representations that illustrate the local energy density across time and frequency, offering insights into system behaviour over time. Scalograms present time on the x-axis and scale on the y-axis, providing a comprehensive view of time-frequency domain characteristics compared to one-dimensional (1D) signals. The CWT filters transient and non-smooth signal segments, as shown in Table 5.15. In Figure 5.4a, 12 impulses in the IRF vibration signal, corresponding to the bearing's IRF frequency, are observed. This results in 12 distinct peaks in the 2D time-frequency diagram in Table 5.15, with more apparent patterns produced by the Amor and Morse wavelets. Similarly, in Figure 5.4b, eight peaks for ORF faults are observed, with the most distinct pattern generated by Amor wavelets in Table 5.15. In contrast, in Figure 5.4c, a lack of clear patterns or features is observed in the Normal (fault-free) or Healthy condition signal, regardless of the wavelet used; refer to Table 5.15. The count of distinct peaks is valuable for distinguishing between IRF, ORF, and Normal (fault-free) or Healthy condition. The following section will employ WSE to validate the selection of the optimal mother wavelet quantitatively *(APPENDIX 2).*



(a)



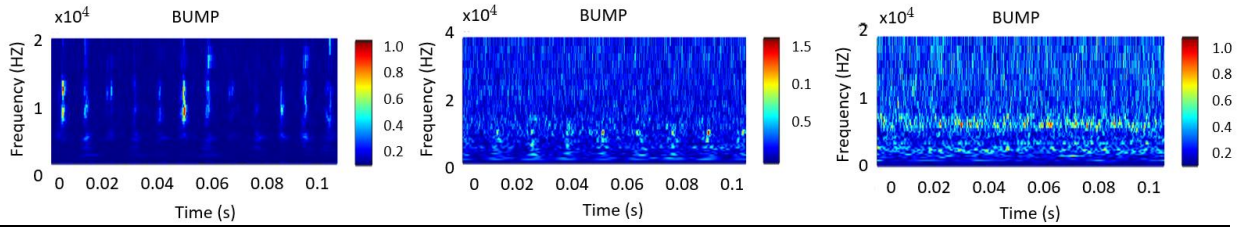
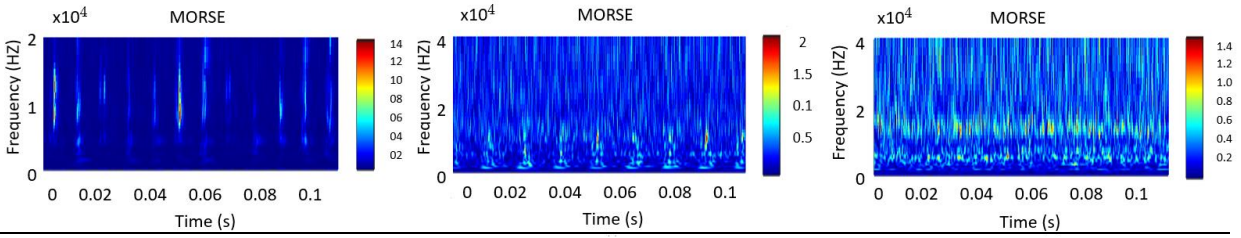
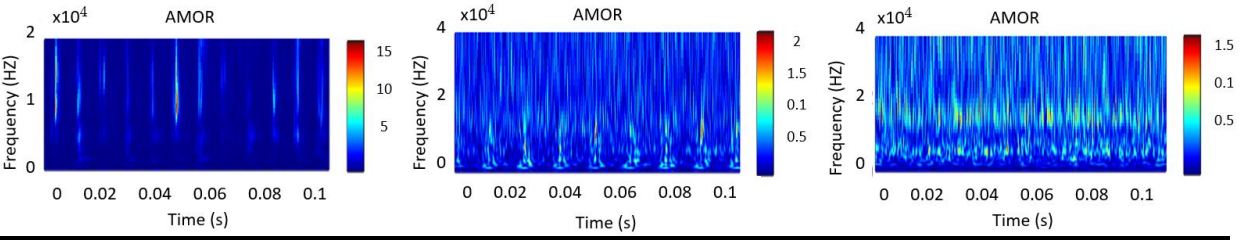
(b)



(c)

Figure 5.4: (a) IRF Signal Trace Peak Count (Represented By Red Boxes) For Innerracefault_Vload_1 Dataset, (b) ORF Signal Trace Peak Count For Outerracefault_3 Dataset (Represented By Red Boxes) and (c) Normal (fault-free) or Healthy Condition Signal Trace for Baseline_1 Dataset, No Peaks.

Table 5.15: Comparative Visualisation of Health Condition Signals: 2D Time–Frequency Diagrams Using Three Types of Mother Wavelet Functions.

Health State	**IRF**	**ORF**	**Normal**
Dataset	InnerRaceFault_vload_1	'OuterRaceFault_3.mat'	'baseline_1.mat'
	2D time-frequency diagrams		
Bump			
Morse			
Amor			

The colour bar adjacent to the scalogram represents the magnitude of the wavelet coefficients. Colours closer to red indicate higher values, which may correspond to signal peaks or areas of high power, while colours closer to blue represent lower values.

- WSE Analysis for Appropriate CWT Selection

A meticulous comparison of WSE scores identified the most suitable mother wavelet function for fault scenarios. The highest WSE score indicates a more scattered signal with a less noticeable pattern, likely representing the Normal (fault-free) or Healthy condition; see Figure 5.4c. WSE is a crucial quantitative measure for CWT, guiding the selection of suitable wavelet foundations in wavelet analysis. The chosen mother wavelet significantly influences denoising, signal preservation, and feature extraction, enhancing the frequency spectrum of the denoised signal (Silik et al., 2021; Guo et al., 2022). Average WSE was subsequently calculated by selecting the optimal mother wavelet function by comparing WSE scores

across different wavelet types (Zhang et al., 2022c). The selection process involves evaluating (WSE_j) scores across various mother wavelet functions in Equation (5.5):

$$WSE_j = \sum_{t=1}^{n} \left|C_{fs}(t,j)\right|^2 . log\left(\left|C_{fs}(t,j)\right|^2\right), \quad \textbf{(5.5)}$$

where C_{fs} is the WT coefficient obtained from W, and fs (Hz) is the sampling frequency determining the number of samples taken per second. The summation range depends on the number of wavelet coefficients obtained from the transform and the chosen wavelet scale. Each coefficient corresponds to a specific scale, *j*, and time, *t*, capturing information about the signal's frequency content and time location (He et al., 2010; Zhang et al., 2022c).

Afterwards, Mean WSE(W) is calculated in Equation (5.6), where *D* represents the dataset (e.g., Normal (fault-free) or Healthy condition, IRF, or ORF), W represents the wavelet type (e.g., 'Bump', 'Morse', or 'Amor'), and N is the total number of datasets. Subsequently, the average mean WSE score (AvgMean WSE(W)) across all datasets for

$$Mean\ WSE(W,D) = \frac{1}{n}\sum_{j=1}^{n} WSE_j, \quad \textbf{(5.6)}$$

$$AvgMean\ WSE(W) = \frac{1}{N}\sum_{D} Mean\ WSE(W,D), \quad \textbf{(5.7)}$$

specific wavelets is determined in Equation (5.7):

Table 5.16 scores provide valuable insights into energy distribution patterns in signals under different fault conditions, with two randomly chosen datasets assessed using WSE *(APPENDIX 2):*

1) Bump:

   The IRF's WSE scores are low (0.017424 and 0.039571), indicating a more concentrated energy distribution and simpler signals. In contrast, the ORF exhibits higher scores (2.0282 and 1.7431), suggesting a more complex energy distribution. The scores in Normal (fault-free) or Healthy condition are relatively low (1.4832 and 1.5995), indicating a simpler energy distribution.

2) Morse:

   In the case of the IRF, low scores (0.011188 and 0.022887) suggest simpler signals. Conversely, the ORF displays higher scores (2.311 and 2.2253), indicating a more

complex energy distribution. The Normal (fault-free) or Healthy condition scores are relatively low (2.2357 and 2.836), suggesting a simpler energy distribution.

3) Amor:

Low scores (0.0090466 and 0.019031) indicate simpler signals for the IRF. The ORF, however, shows a positive score (0.61065), suggesting a more dispersed energy distribution. In the Normal (fault-free) or Healthy condition, higher scores (2.6529, 5.3807, and 15.826) indicate more complex energy distributions.

Table 5.16: WSE Scores Comparison with Three Types Mother of Wavelet Functions.

Health State	Training Set	Code	Morse	Bump	Amor
Normal	baseline_1	data_normal	2.236	1.483	5.381
	baseline_2	data_normal_2	2.836	1.600	15.830
	WSE Avg. for 0.1 s		2.536	1.541	10.603
IRF	InnerRaceFault_vload_1	data_inner	0.011	0.017	0.009
	InnerRaceFault_vload_2	data_inner_2	0.023	0.040	0.019
	WSE Avg. for 0.1 s		0.017	0.028	0.014
ORF	OuterRaceFault_3	data_outer	2.311	2.028	0.611
	OuterRaceFault_1	data_outer_2	2.225	1.743	2.653
	WSE Avg. for 0.1 s		2.268	1.886	1.632

The mother of wavelet analysis can be summarised in Figure 5.5, where it shows the visual comparison; the “Amor” wavelet type shows relatively better discrimination between the Normal (fault-free) or Healthy condition and faulty conditions, as it exhibits lower WSE scores for the faulty conditions compared to the Normal (fault-free) or Healthy condition. However, based on the analysis of the WSE scores, three wavelet coefficients were evaluated: Morse, Bump, and Amor. For the Normal (fault-free) or Healthy condition dataset, the Morse coefficient had an average WSE score of 2.53585, the Bump coefficient had a score of 1.54135, and the Amor coefficient had the highest score of 10.60335, indicating a more dispersed energy distribution. When considering the IRF dataset, the Morse, Bump, and Amor coefficients had average WSE scores of 0.0170375, 0.0284975, and 0.0140388, respectively. For the ORF dataset, the average WSE scores were 2.26815, 1.88565, and 1.631775 for the Morse, Bump, and Amor coefficients, respectively. The results show that the Amor coefficient exhibited the highest average WSE score for the Normal (fault-free) or Healthy condition dataset, suggesting a distinct energy distribution. Consequently, the Amor coefficient emerges as a potential candidate for identifying Normal (fault-free) or Healthy condition in contrast to faulty ones.

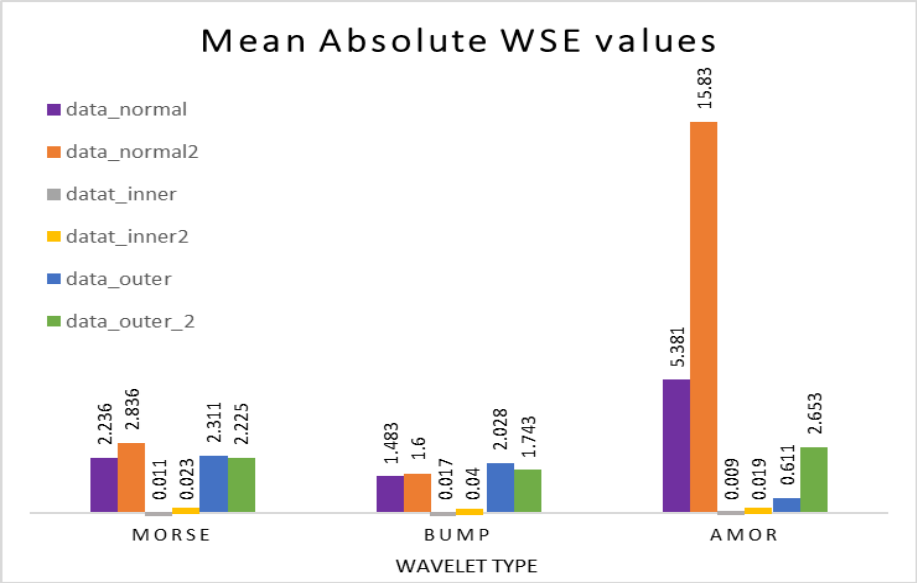


Figure 5.5: Mean Absolute WSE Values for Different Mothers of Wavelets.

5.2.2.2 Step 2: CWT Energy Assessment for Each Load Factor

This section uses the data segmentation subfiles in Table 5.9 in Section 5.2.1.2. For further mean energy analysis per LF for IRF and ORF, types per LFi, where 'i' indexes the different LFss.calculate the wavelet energy values using the CWT technique. Let $x_i(t)$ represent the vibration signal for LF$_i$ at time t. The CWT coefficients are denoted as $C_{i,j}(t)$, where j represents the selected wavelet scale (Jayamaha et al., 2019; Silik et al., 2021). Following these steps:

- Extract the vibration signal for LFi: $x_i(t)$.
- Perform the CWT on the vibration signal: $C_{i,j}(t)$; see Equation (5.8). The scale used in this study was 5.
- Calculate the wavelet energy $E^j_{wavelet,i}$ for each scale j, $E^j_{wavelet,i}$, in Equation (5.9):

$$C_{i,j}(t) = CWT(x_i(t), wavelet_{type}, j), \quad \textbf{(5.8)}$$

$$E^j_{wavelet,i} = \sum_t \left|C_{i,j}(t)\right|^2 \quad \textbf{(5.9)}$$

Hence, the concept of "scale" j is crucial in understanding the CWT technique in wavelet analysis. The CWT is a method used to examine signals at various scales, allowing the detection of different frequency components in a signal with varying levels of detail. Each scale j corresponds to a specific width of the analysing wavelet, a mathematical function

used in the transformation process. Smaller scales represent narrower wavelets sensitive to high-frequency details, enabling the capture of rapid signal variations. Conversely, larger scales correspond to wider wavelets, capturing lower-frequency signal components with broader coverage but less fine detail. In equations involving wavelet analysis, such as $\left|C_{i,j}(t)\right|^2$, the squared absolute value of wavelet coefficients at a particular scale j and for a specific LF *i* is calculated. This squared magnitude is summed across time t, resulting in the computation of the wavelet energy at that scale j. This energy measure provides valuable insights into the contribution of different frequency components to the overall energy content of the signal (Jayamaha et al., 2019).

Subsequently, the mean energy tables for each LF*i*, covering IRF, ORF and Normal (fault-free) or Healthy condition, were created by aggregating the calculated wavelet energy values. Let $E_{wavelet,i} = \left[E^1_{wavelet,i}, E^2_{wavelet,i}, \ldots, E^{N_{scales}}_{wavelet,i}\right]$ be the vector of wavelet energy values for LF *i*. Then, calculate the mean wavelet energy wavelet, $\bar{E}_{wavelet,i}$ for each LF *i* by taking the average of the wavelet energy values across all scales, shown in Equation (5.10):

$$\bar{E}_{wavelet,i} = \frac{1}{N_{scales}} \sum_{j=1}^{N_{scales}} E^j_{wavelet,i} \qquad \textbf{(5.10)}$$

Here, building upon the foundation of wavelet energy, the mean wavelet energy $\bar{E}_{wavelet,i}$ is computed by averaging energy values over all scales. This metric provides a concise yet powerful representation of the energy behaviour post-fault for each LF.

- CWT Energy Assessment for Each LF Using Optimal CWT Technique

In the assessment of mean energy values for IRF and ORF with LF 270 as Normal (fault-free) or Healthy condition shown in Table 5.17, the following observations were made: For IRF, LF 270 (Normal (fault-free) or Healthy condition) exhibited a mean energy value of 5.7012, indicating a lower energy content. LF 50, 100, and 150 had mean energy values ranging from 24.915 to 27.547, indicating a relatively lower energy content. In contrast, LF 200, 250, and 300 showed mean energy values ranging from 32.199 to 36.147, suggesting a higher energy content and a more pronounced presence of IRFs. Similarly, LF 50, 100, 150, 200, and 250 for ORF bearings had mean energy values ranging from 5.4309 to 7.6992, indicating a relatively lower energy content than LF 270. LF 270 (Normal (fault-free) or Healthy condition) had a mean energy value of 5.7012, representing the Normal (fault-free)

or Healthy condition with a lower energy content. LF 300 exhibited a mean energy value of 18.612, indicating a substantial 226.88% increase compared to Normal (fault-free) or Healthy condition.

In summary, ORF and IRF showed notable increases in mean energy with distinct patterns. ORF exhibited the highest increase at LF 300 (226.88%), while IRF showed higher increases, with the highest at LF 250 (533.49%). The increased variability ranged from 2.08% to 226.88% for ORF and 337.68% to 533.49% for IRF. IRF generally displayed higher percentage increases than ORF, providing insights for effective fault detection and system management. The complete code is in *(APPENDIX 2).*

Table 5.17: IRF and ORF CWT Mean Energy.

LF	**IRF Type**		**ORF Type**	
(lbs)	**Mean Energy**	**Mean Energy Increase (%)**	**Mean Energy**	**Mean Energy Increase (%)**
50	25.549	347.70%	7.699	35.16%
100	27.547	383.65%	5.431	4.76%
150	24.915	337.68%	5.573	2.08%
200	33.742	491.88%	7.604	33.35%
250	36.147	533.49%	7.178	25.90%
270	5.7012	0% (baseline)	5.701	0% (baseline)
300	32.199	464.25%	18.612	226.88%

- Two-Sample *t*-Test for Significance Testing

In this study, a two-sample *t*-test was conducted using MATLAB R2023a to assess differences in mean CWT energy between the Normal (fault-free) or Healthy LF condition (LF 270 lbs) and other LFs (50, 100, 150, 200, 250, and 300 lbs) for IRF in Figure 5.6 and ORF in Figure 5.7. Individual *t*-tests for each LF determined whether the mean energy of the Normal (fault-free) or Healthy condition load differed significantly from other LFs, with a significance level of 0.05. Results consistently demonstrated a clear and significant distinction in mean CWT energy between the Normal (fault-free) or Healthy condition and various loads. The null hypothesis (H0), suggesting no significant difference in CWT mean energy between LF 270 and other LFs, was rejected in favour of the alternative hypothesis (H1), indicating a substantial distinction. This finding held for IRF and ORF LFs, with low *p*-values, large sample sizes, substantial t-values, and confidence intervals, all supporting the robustness and reliability of these results.

LoadFactor	Mean	StdDev	SEMean	MeanDiff	CI_Lower	CI_Upper	tValue	DF	pValue	Significant
50	25.549	80.211	0.20958	19.848	-20.259	-19.437	-94.663	2.923 x105	0	true
100	27.547	80.949	0.2115	21.846	-22.261	-21.431	-103.25	2.923 $x10^5$	0	true
150	24.915	74.066	0.19352	19.214	-19.594	-18.835	-99.238	2.923 $x10^5$	0	true
200	33.742	109.49	0.28607	28.04	-28.601	-27.48	-97.997	2.923 $x10^5$	0	true
250	36.147	113.29	0.29599	30.445	-31.026	-29.865	-102.84	2.923 $x10^5$	0	true
300	32.199	94.74	0.24753	26.498	-26.983	-26.013	-107.01	2.923 $x10^5$	0	true

Figure 5.6: Two Samples' t-Test Results Compare IRF Load Factors (50, 100, 150, 200, 250, 300) With Normal (fault-free) or Healthy Load Factor Condition (Load Factor 270).

LoadFactor	Mean	StdDev	SEMean	MeanDiff	CI_Lower	CI_Upper	tValue	DF	pValue	Significant
50	7.6992	7.3528	0.019211	1.9981	-2.0376	-1.9585	-99.051	2.923 $x10^5$	0	true
100	5.4309	4.0811	0.010663	-0.27028	0.24616	0.29441	21.955	2.923 $x10^5$	9.364 $x10^{107}$	true
150	5.5728	4.0481	0.010577	-0.12837	0.10439	0.15235	10.491	2.923 $x10^5$	9.573 $x10^{26}$	true
200	7.6036	14.487	0.037852	1.9024	-1.9776	-1.8273	-49.609	2.923 $x10^5$	0	true
250	7.1779	9.5466	0.024943	1.4767	-1.5271	-1.4264	-57.48	2.923 $x10^5$	0	true
300	18.612	51.481	0.13451	12.911	-13.175	-12.647	-95.882	2.923 $x10^5$	0	true

Figure 5.7: Two Samples' t-Test Results Compare ORF Load Factors (50, 100, 150, 200, 250, 300) With Normal (fault-free) or Healthy Load Factor Condition (Load Factor 270).

5.2.2.3 Step 3: Customised Load Adaptive Framework (CLAF)

The Load Index, developed based on optimal CWT energy to capture the influence of LF variations during fault occurrences, is a qualitative representation of the effects of varying LFs on bearing behaviour. Subsequently, bearing faults were categorised into load-dependent fault subclasses, displaying distinct severity levels: Mild, Moderate, and Severe, using the CLAF. This comprehensive classification helps explain how varying LFs contribute to the manifestation and progression of bearing faults by following these steps; see the full code in *(APPENDIX 2)*:

1. Calculate normalised energy values:

   For each LF *i*, the normalised CWT energy values $E^j_{normalized,i}$ were calculated using min–max scaling. This process ensures that the wavelet energy values$E^j_{wavelet,i}$ range between 0 and 1. The normalisation is expressed by Equation (5.11):

$$E^j_{normalized,i} = \frac{E^j_{wavelet,i} - min\left(E^j_{wavelet}\right)}{max\left(E^j_{wavelet,i}\right) - min\left(E^j_{wavelet,i}\right)}, \quad \textbf{(5.11)}$$

   In this normalised range, 0 represents the minimum energy value in the dataset, and 1 represents the maximum energy value in the dataset. Hence, data normalisation helps improve the performance of ML models by ensuring that all features are on a similar

scale. This can prevent some features from dominating others and improve the model's accuracy (Jang and Cho, 2021). All other energy values are linearly scaled within this range. Here, min $(E^j_{wavelet})$ represents the minimum wavelet energy value across all LFs and scales, and max$\left(E^j_{wavelet,i}\right)$ represents the maximum wavelet energy value across all LFs and scales.

2. Identify Normal (fault-free) or Healthy condition indices:

   I_{normal} represents the indices corresponding to the Normal (fault-free) or Healthy condition. In the analysis context, it refers to the index where the LF is 270, which is considered the Normal (fault-free) or Healthy condition or baseline. These indices are used to calculate the deviation from the Normal (fault-free) or Healthy condition for each LF and wavelet energy value.

   In the mathematical notation, I_{normal} is a set of indices i for which the LF is equal to 270; see Equation (5.12):

$$I_{normal} = \{i | load\ factor = 270\} \quad \textbf{(5.12)}$$

3. Quantify deviation: calculate deviation from Normal (fault-free) or Healthy condition; see Equation (5.13):

$$D_i^j = \begin{cases} E^j_{normalized,i}, & if\ i \notin I_{normal} \\ 0, & otherwise \end{cases} \quad \textbf{(5.13)}$$

   where deviations D_i^j from the Normal (fault-free) or Healthy condition are calculated, highlighting differences between the normalised energy values and the baseline. When an LF is not within I_{normal}, the corresponding normalised energy value $E^j_{normalized,i}$ is considered. Otherwise, the deviation is set to zero.

4. Severity of changing LF: threshold setting

   4.1 Define adjustable severity thresholds

   4.2 Categorise the severity S_i^j based on the deviation magnitude D_i^j.and threshold; see Equation (5.14):

$$S_i^j = \begin{cases} 'Mild', & if\ D_i^j \le mild\_threshold \\ 'Moderate', & if\ mild\_threshold < D_i^j \le moderate\_threshold \\ 'Severe', & if\ otherwise \end{cases} \quad \textbf{(5.14)}$$

Hence, the severity of deviations D_i^j is categorised to assess the impact post-fault. Adjustable severity thresholds differentiate between 'Mild', 'Moderate' and 'Severe' conditions and store severity as a cell array value. This step is vital in determining the gravity of the machinery's response to various fault scenarios, enabling efficient resource allocation and timely interventions, and preventing potential escalations. In this chapter, the author chose the following thresholds, which can be adjusted according to the application: mild_threshold = 0.2; moderate_threshold = 0.5.

5. Categorise severity S_i^j based on the deviation magnitude D_i^j

The normalised energy values allow us to effectively compare the energy levels of different LFs, as they are all scaled within the same range. However, it is essential to note that the normalised energy values are not directly related to the severity categorisation ('Mild', 'Moderate', or 'Severe'). The severity categorisation is based on the 'Deviation' column, which represents the deviation of each LF's mean energy from the mean energy of the Normal (fault-free) or Healthy condition. Following are the IRF and ORF types after the assessment, as shown in Table 5.18:

Table 5.18: IRF and ORF Load-Dependent Fault Subclasses Through CLAF.

LF	**Mean Energy**		**NormalisedEnergy**		**Deviation**		**Load-Dependent Subclasses**	
(lbs)	**IRF**	**ORF**	**IRF**	**ORF**	**IRF**	**ORF**	**IRF**	**ORF**
50	25.549	7.6992	0.14035	0.05758	0.1403	0.05758	{'Mild'}	{'Mild'}
100	27.547	5.4309	0.15053	0.023062	0.15053	0.023062	{'Mild'}	{'Mild'}
150	24.915	5.5728	0.14063	0.031372	0.14063	0.031372	{'Mild'}	{'Mild'}
200	33.742	7.6036	0.28444	0.092816	0.28444	0.092816	{'Moderate'}	{'Mild'}
250	36.147	7.1779	0.29911	0.061822	0.29911	0.061822	{'Moderate'}	{'Mild'}
270	5.7012	5.7012	0.00930	0.027659	0	0	{'Normal'}	{'Normal'}
300	32.199	18.612	0.23412	0.89814	0.23412	0.89814	{'Moderate'}	{'Severe'}

- IRF Customised LF Assessment:

  Min-max scaling was employed to normalise the energy values, transforming the original energy values into a range of [0, 1]. In this normalised range, 0 signifies the minimum energy value in the dataset, while 1 represents the maximum energy value. All other energy values are linearly scaled within this range. The 'Normalised Energy' column in the provided table reflects the energy values post min-max scaling, where one corresponds to the maximum energy value. For instance, the energy value of 'LF' 250

is relatively the highest compared to other LFs in the dataset, as evidenced by its proximity to 1 in the normalised range.

Conversely, 'LF' 50, 'LF' 100, and 'LF' 150 had normalised energy values around 0.14, indicating that their energy values were closer to the lower end of the normalised range (0). These LFs exhibited lower energy values compared to others in the dataset. Notably, the normalised energy values did not directly correspond to the severity categorisation ('Mild,' 'Moderate,' or 'Severe'). The severity categorisation was based on the 'Deviation' column, which represents the deviation of each LF's mean energy from the mean energy of the Normal (fault-free) or Healthy condition.

- ORF Customised LF Assessment:

  Long-duration operation at higher LFs for the ORF significantly influences degradation. Across LFs 50, 100, 150, 200, and 250, the mean energy values ranged from 5.4309 to 7.6992, indicating relatively lower energy content in the vibration signals compared to LF 270, which represents the Normal (fault-free) or Healthy condition with a mean energy value of 6.0981. The Normal (fault-free) or Healthy condition exhibited relatively lower energy levels, as expected. However, LF 300 stood out with a higher mean energy value of 18.612, suggesting that the associated ORF condition had a notably higher energy content in the vibration signals than the other LFs. This detailed energy analysis provides valuable insights into the variations related to different LFs and fault conditions, enhancing the understanding of the degradation process.

5.2.2.4 Step 4: CLAF Validation

The proposed CLAF is a fault condition monitoring system designed to identify issues based on load-dependent fault subclasses. In this step, the efficiency of the CLAF is validated by evaluating time, frequency, and spectral features, which have been ranked using one-way ANOVA.

In real-world scenarios, not all extracted features are equally important. Some features are more critical for classification, while others can negatively impact accuracy by hindering the algorithm's ability to generalise patterns. One-way ANOVA is used to select the most robust subset of features, addressing the challenge of extracting key components for damage detection in structural health monitoring (Zhang et al., 2023d). Structural

dynamic measurements often exhibit complex, time-varying behaviour, making them sensitive to changes in time-frequency characteristics (Silik et al., 2021).

Time, frequency, and spectral features were carefully selected and ranked within this context using one-way ANOVA. Following this, classifiers were trained to determine the optimal feature set based on accuracy, as detailed in Section 5.2.1.4 (see Table 5.12). The study generated 24 features within the 2,500–25,000 Hz frequency band, with each signal contributing five spectral peaks, resulting in five frequency features per peak. Subsequently, a one-way ANOVA test was conducted on the CLAF load-dependent fault subclasses to refine feature selection further (see Table 5.19).

Table 5.19: CLAF Load-Dependent Fault Subclasses (One-Way ANOVA Ranking, AR Model Order Fifteen, Peaks = 5).

Feature Rank	One-way ANOVA Score	Feature Rank	One-way ANOVA Score
1. Mean	316.4447	13. PeakAmp5	84.3280
2. ShapeFactor	288.4202	14. Skewness	73.1278
3. PeakValue	245.4272	15. PeakAmp2	70.5038
4. RMS	240.9294	16. PeakFreq1	69.1381
5. Std	240.2707	17. SINAD	58.7164
6. ClearanceFactor	235.2321	18. S/N	58.6086
7. ImpulseFactor	225.2555	19. PeakAmp4	51.3935
8. Kurtosis	211.9440	20. PeakAm3	38.7712
9. CrestFactor	198.2645	21. PeakFreq4	25.1821
10. PeakAmp1	161.2217	22. PeakFreq2	17.6449
11. BandPower	126.8539	23. PeakFreq5	13.9307
12. PeakFreq3	116.7983	24. THD	0

Features with ANOVA scores below 26 were excluded from further study. This step aimed to enhance the selection process by concentrating on features that had a more significant impact. Observing the initial trial's high accuracy, the author systematically reduced the number of features, utilising accuracy as a metric for efficient feature reduction. This reduction process was carried out gradually, guided by accuracy measures. Subsequently, several classifiers were evaluated in the study, and their performance was meticulously documented across various feature subsets. The training dataset, comprising 813 subfolders, was divided into 60.00% for training, 20.00% for validation, and 20.00% for testing.

A five-fold cross-validation was implemented to ensure robust performance assessment. The feature selection process, guided by one-way ANOVA scores, began with

the top 20 features (ANOVA scores > 26) and these were systematically narrowed down to the top 5 features (ANOVA scores > 240), allowing for refined classifier selection based on accuracy and efficiency. Table 5.20 shows that the RUSBoostedTrees model, tested with the top 20 features, achieved an accuracy of 93.80% with a training time of 11.539 s.

Table 5.20: CLAF Load-Dependent Fault Subclasses Classifiers Training on Various Feature Subsets.

Classifier	**ANOVA Ranking**	**TTime [1]**	**Test Dataset**				
		(s)	**VA [2]**	**NA [3]**	**MA [4]**	**MoA [5]**	**SA [6]**	**Overall Accuracy**
RUSBoostedTrees	Top 20 >26	11.539	92.60%	100%	92.40%	91.20%	100%	93.80%
Fine Tree	Top 17 >58.6	4.393	92.60%	100%	95.70%	82.40%	100%	93.80%
WNN	Top 10 >161	18.155	91.20%	100%	97.80%	88.20%	100%	96.30%
CubicSVM	Top7 (a) >215	8.1055	93.10%	100%	96.70%	82.40%	100%	94.40%
Medium Gaussian SVM	Top 7 (b) >215	5.8059	91.60%	100%	96.70%	82.40%	100%	94.40%
Fine Gaussian SVM	Top 5 >240	12.711	92.90%	100%	97.80%	82.40%	100%	95.10%

[1] TTime is the training time, [2] VA is the validation accuracy, [3] NA is the Normal (fault-free) or Healthy condition accuracy, [4] MA is the Mild state accuracy, [5] MoA is the Moderate state accuracy, and [6] SA is the Severe state accuracy.

The Fine Tree model, using 17 features, matched this accuracy but with a shorter training time of 4.393 s. The Wide Neural Network (WNN), which used the top 10 features (with ANOVA scores > 161), achieved an accuracy of 96.30% with a standard deviation of ± 0.50% in 18.155 s during five-fold cross-validation. This performance is attributed to the WNN's single-layer architecture with 100 neurons and a Rectified Linear Unit (ReLU) activation function without regularisation (Lambda set to 0). The accuracy was determined by dividing the dataset into five parts, training the model on four parts, and testing the remaining part. This process was repeated five times, with the ± 0.50% standard deviation reflecting the variability across the different folds. Cross-validation helps ensure the model performs consistently on unseen data, reducing the risk of overfitting. The careful selection of the top 10 features maintained interpretability while achieving a solid testing performance. Iteration refers to the repeated updates to the model's parameters during training to minimise the loss function. The network's validation accuracy of 91.00% over 57 iterations further illustrates its effectiveness. While cross-validation evaluates how well the model generalises

to new data, iteration focuses on improving the model's performance during training. The WNN, known for handling datasets with many input features, is well-suited for tabular or structured data (MathWorks-3, 2024). All these experiments were conducted using the Classification Learner application in MATLAB 2023a.

Such a high level of accuracy demonstrates the CLAF's nuanced understanding of fault patterns and its capability to effectively distinguish between 'Mild,' 'Moderate,' and 'Severe' fault categories under different LFs.

5.3 Summary

This research proposes the CLAF for classifying faults in Induction Motors (IMs) into load-dependent fault subclasses, namely 'Mild,' 'Moderate,' 'Severe,' and 'Normal (fault-free) or Healthy condition' categories. The framework provides a comprehensive understanding of fault severity under varying LFs, offering an insightful method for fault analysis. Tailored to the MFPT bearing dataset, this research highlights patterns in TFD features under six different LFs. It demonstrates how fault severity varies across various LF conditions using an optimal CWT energy approach selected by WSE.

In this research, the CLAF was developed in two phases:

1) Phase 1: Load-dependent patterns in TFD features were explored using one-way ANOVA ranking, and validation was carried out with bagged tree classifiers. The findings revealed consistent deviations in key features for both fault types, with IRF showing more pronounced alterations. The one-way ANOVA test ranked the shape factor feature as the most significant, followed by peak value, whereas THD was insignificant. Two AR models were employed in the frequency domain feature extraction. Subclassification based on these extracted features for each LF revealed distinct patterns, which helped to identify load-induced patterns and improve understanding of the relationship between LFs and feature expression in bearing health assessment. The approach using Bagged Tree classifiers with the top 19 features, as determined by ANOVA scores, achieved an accuracy of 86.40%.
2) Phase 2: WSE determined 'Amor' as the optimal CWT method, surpassing alternatives such as 'Bump' and 'Morse' in the Normal (fault-free) or Healthy condition dataset. This phase highlighted a significant correlation between fault severity and LF, significantly when loads exceeded 300 lb. Severe ORF faults demonstrated a notable

226.88% increase in CWT energy compared to the Normal (fault-free) or Healthy condition. Similarly, IRFs exhibited significant energy increases at different LF levels, rising by 491.00%, 533.49%, and 464.25% at 200 lb, 250 lb, and 300 lb, respectively. A two-sample t-test confirmed the significance of these results. The study defined load-dependent fault subclasses within the MFPT bearing dataset, establishing specific thresholds for Mild, Moderate, and Severe fault levels based on the energy deviation from Normal (fault-free) or Healthy condition. The CLAF framework was validated for its load-dependent fault subclass creation using TFD features. It achieved 96.30% ± 0.50% standard deviation classification accuracy, reflecting the variability across the five different folds with a WNN and the top 10 features ranked by ANOVA. It was particularly effective at classifying Severe faults, achieving 100% accuracy, Moderate faults at 88.30%, and Mild faults at 97.80%, thereby demonstrating its ability to detect nuanced fault variations under different LF conditions in IMs. These results underscore the practical benefits of the CLAF in enhancing fault classification for IMs and its potential in advancing condition monitoring.

This chapter's main contributions are as follows:

1. Comprehensive Time and Frequency Analysis: This study conducted a detailed TFD analysis under six LF conditions, highlighting patterns and variations in fault severity and providing valuable insights into IM behaviour.
2. Optimal CWT Approach: Selecting an optimal CWT approach using WSE improves signal processing for TFD feature extraction, denoising, and pattern recognition.
3. Revealing Load-Dependent Fault Subclasses: This research identified and classified load-dependent fault subclasses, including Mild, Moderate, and Severe, enhancing the understanding of fault severity in different LF scenarios.
4. Proposing a CLAF: The research introduces a novel CLAF, extending traditional fault classification methodologies by considering LF variations and dataset customisation.

Chapter 6: A Novel Load-Dependent Multimodal Vibration Signal Enhancement and Fusion (LD-MVSEF) for Load-Specific Condition Monitoring

6.1 Proposed Methodology

This section outlines the systematic approach of the proposed novel Load-Dependent Multimodal Vibration Signal Enhancement and Fusion (LD-MVSEF) for Load-Specific Condition Monitoring, building upon the Customised Load Adaptive Framework (CLAF) load-dependent fault subclasses introduced in Chapter 5. The methodology was applied to the Machinery Failure Prevention Technology (MFPT) bearing dataset. It involves the independent extraction of features from different data representations within this single data source, implemented across three separate feature extraction channels. Features extracted from each channel are then directed to their respective classification modules, where individual classification decisions are made. Subsequently, a fusion module consolidates these individual decisions into a unified classification result. The processing for the proposed methodology was conducted using MATLAB R2023a software. This section provides an overview of the methodology framework and details of the data used.

6.1.1 Load-Dependent Multimodal Vibration Signal Enhancement and Fusion

The LD-MVSEF framework incorporates multiple data channels and decision fusion approaches, complemented by the CLAF for creating load-dependent fault subclasses. Various data sources are integrated, including Gradient Angular Difference Field (GADF) images, Continuous Wavelet Transform (CWT) images, and features from the time and frequency domains. These inputs enhance the efficacy of condition monitoring by leveraging complementary patterns across different modalities for improved fault classification. The outputs from multiple classifiers are consolidated using decision fusion techniques, ensuring robust and accurate classification. The methodology includes six detailed steps, as presented in Figure 6.1:

1. Data Preprocessing with the CLAF:

   In this stage, the input data are segmented and prepared for further analysis. The process begins with splitting the raw MFPT bearing data vibration signals according to their respective classes— Normal (fault-free) or Healthy condition and fault (Inner Race Fault (IRF) and Outer Race Fault (ORF). After splitting, the raw vibration signals are encoded into equivalent image formats.

- This multichannel approach is critical for identifying differences in vibration signals associated with varying Load Factor (LF) conditions and fault types. The structured preparation of the dataset into load classes (IRF50, IRF100, IRF150, IRF200, IRF250 and IRF300, ORF50, ORF100, ORF150, ORF200, ORF250 and ORF300, Normal270)is essential for its subsequent application in the CLAF, ensuring that the framework receives correctly segmented and analysed data for optimal performance.

2. Multichannel Input Preparations:

In this stage, The first phase establishes three distinct data channels for comprehensive analysis. In Channel 1, raw vibration signals are processed. Channel 2 generates two-dimensional (2D) CWT images from these signals, and Channel 3 produces 2D encoded GADF images. After splitting, the raw vibration signals are encoded into equivalent image formats:

- Channel 1: Raw vibration signal.
- Channel 2: The class-specific raw vibration signals are encoded into CWT images using the Amor technique.
- Channel 3: The class-specific raw vibration signals are encoded into 2D GADF images.
- Applying the CLAF to create load-dependent fault subclasses—'Mild,' 'Moderate,' 'Severe,' and 'Normal (fault-free) or Healthy condition'—tailored to specific datasets, forming the foundation for subsequent analysis.

3. Feature Extraction and Classifier Selection for Channel 1 (Raw Vibration Signal):

In the third step, the raw vibration signal data (Channel 1) are subject to feature extraction and subsequent selection through the one-way Analysis of Variance (ANOVA) ranking method. Various classifiers are employed, and the highest classifier accuracy for the raw vibration signal subfiles is determined where Channel 1 inputs are switched from raw vibration signal to (Channel 1): Optimal Time and Frequency Domain (TFD) feature subset selection.

After that, a data preprocessing step was added to address the data imbalance issue and link the TFD features with the equivalent GADF and CWT images into a single data store. This adjustment enhances the dataset by ensuring all classes are balanced before

classification. This involved oversampling the minority classes to ensure that each class had an equal number of samples, matching the class with the highest number of samples.

4. Channels Classification Approaches and Training Methods: Training and Selection of Classifiers for TFD features, including spectral features using Autoregression (Channel 1) and CNN Architectures for Channels 2 and 3 (CWT and GADF images): Step four focuses on the training and selection of classifiers for the different data channels:
   - For Channel 1 (TFD features, including spectral features using Autoregression), classifiers such as Cubic Support Vector Machine (CubicSVM) and Wide Neural Network (WNN) are trained on the extracted features. The best-performing model is selected for further analysis.
   - For Channels 2 and 3 (CWT and GADF images), pre-trained Convolutional Neural Networks (CNNs), such as AlexNet and ResNet-18, originally trained on the ImageNet dataset, are fine-tuned on the 2D encoded images. The final fully connected layer of each network is replaced with a new layer containing four output neurons, where each neuron corresponds to one of the four categories: 'Mild,' 'Moderate,' 'Severe,' and 'Normal (fault-free) or Healthy condition'. The images, including CWT spectrograms and 2D GADF-encoded images, are resized to match the input dimensions of the CNN architectures: 227 x 227 x 3 for AlexNet and 224 x 224 x 3 for ResNet-18.

5. Single Channel Performance Analysis:

In this step, the performance of classifiers for each of the three channels—raw vibration signals, CWT images, and GADF images—is analysed. The classifiers are evaluated on their ability to classify the data into four classes ('Mild,' 'Moderate,' 'Severe,' and 'Normal (fault-free) or Healthy condition'). For each channel, the classifier with the highest overall accuracy is selected.

6. Weighted Decision Fusion:

In the final step, two weighting systems are used for decision fusion: Weighting System 1 (Adaptive Weighting), where weights are dynamically assigned based on the classifier's performance for specific conditions, and Weighting System 2 (Equal Weighting), where all channels receive equal weights regardless of their individual

performance. Two- and three-channel configurations are evaluated to determine the most effective fusion model for optimising classification accuracy and robustness.

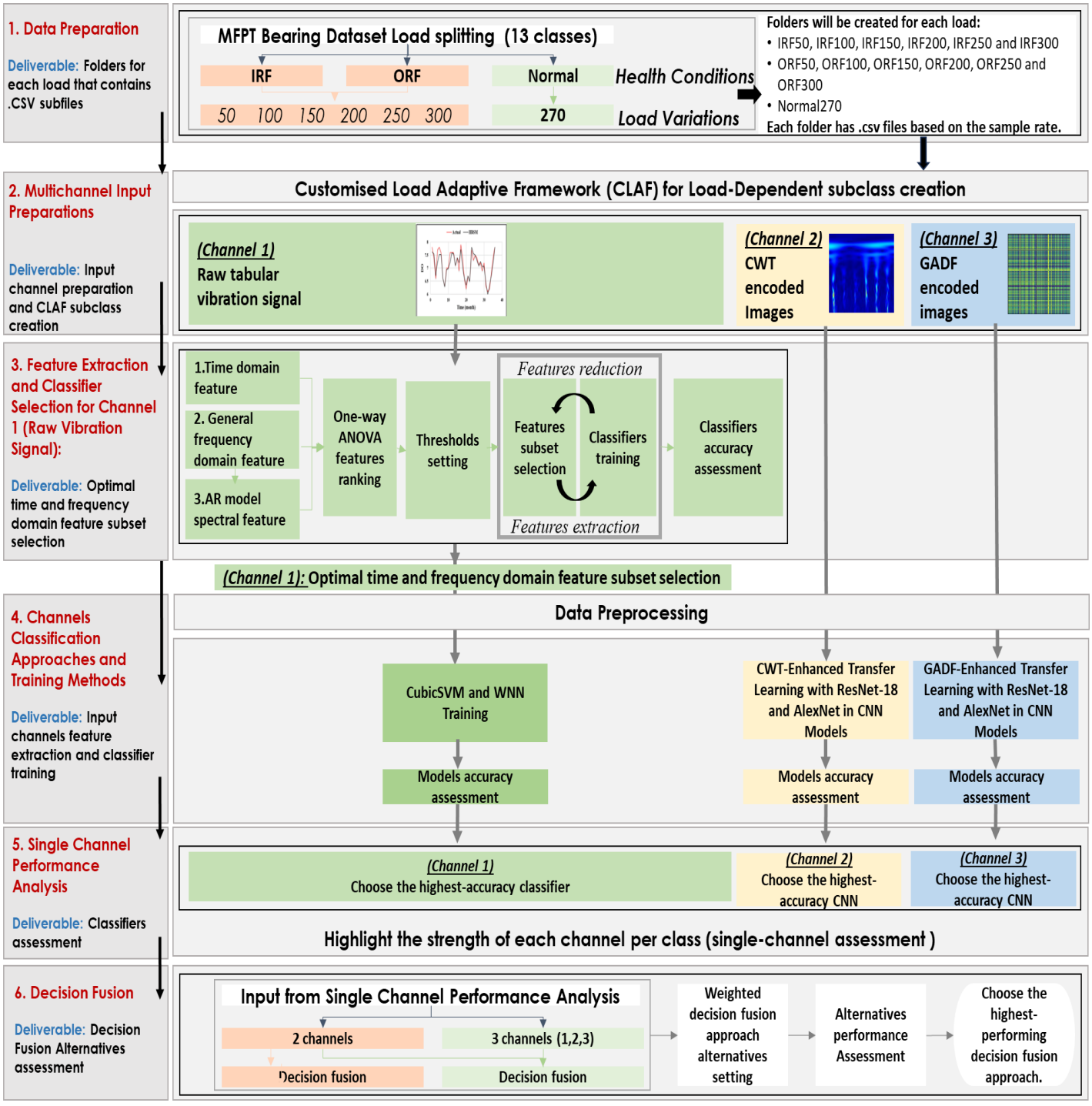
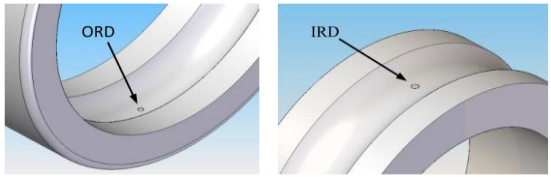


Figure 6.1: Load-Dependent Multimodal Vibration Signal Enhancement and Fusion (LD-MVSEF) Methodology.

6.1.2 Dataset

The current study utilises the MFPT bearing dataset obtained from a NICE bearing test rig. The dataset includes samples representing the Normal (fault-free) or Healthy condition, ORF, and IRF, as illustrated in Figure 6.2. Key details of the dataset include three Normal (fault-free) or Healthy condition samples recorded at 97,656 Hz for 6 s each, three ORF samples at the same frequency (with LF ranging from 25 to 300 lbs), and an additional seven ORF samples at 48,828 Hz (with LFs varying from 0 to 300 lbs). Furthermore, the dataset contains IRF samples at 48,828 Hz, with LFs ranging from 0 to 300 lbs. This dataset serves as a standardised benchmark, providing essential information such as radial LF, shaft speed, and signal characteristics while maintaining a consistent shaft speed of 1,500 rpm (25 Hz).

This thesis recommends splitting the dataset using a 5,000/97,566 ratio to ensure consistent and efficient data segmentation while preserving fault classification features. This interval captures essential characteristics across samples. Consistency ensures reproducibility for Machine Learning (ML). Each subfile represents the signal's behaviour for accurate fault detection within the specified timeframe. Consistency is maintained across the three channels to avoid bias in training.



(**a**) (**b**)

Figure 6.2: Computer-Aided Drawings of Defects Made on (a) ORF; (b) IRF (Jain and Bhosle, 2022).

6.2 Results and Discussion

This section presents a comprehensive analysis and interpretation of the outcomes obtained from the experimental study. The focal point of the analysis revolves around the performance evaluation of various fusion techniques utilised for the classification of LF conditions. These techniques encompass diverse feature representations and models. The overarching goal of the current study is to discern effective strategies for enhancing the accuracy of load index prediction, thereby improving the reliability and robustness of

machinery fault classification. Steps 3 and 4 show the three approaches used in the single channel, starting with TFD extraction features on the original vibration signal, then with pre-trained CNNs (AlexNet and Residual Network-18 (ResNet-18)) on encoded vibration signals into two forms: CWT vibration-encoded images and GADF vibration-encoded images.

6.2.1 Data Preparation

This section presents the preprocessing performed on the MFPT bearing dataset, detailing the process of dividing the data based on LF conditions. This division is crucial for applying the CLAF to identify LF-dependent patterns that differ from conventional fault classification methods used in Induction Motor (IM) bearings. The dataset has been systematically split per the CLAF approach, which marks a significant departure from traditional fault classification by factoring in LF conditions and adapting the dataset for specialised analysis, as indicated in Chapter 5. The research includes a comparative study across six distinct LF conditions (50, 100, 150, 200, 250, and 300 lbs) against a Normal (fault-free) or Healthy condition LF condition set at 270 lbs. This results in 13 categories: six each for IRF and ORF under varying LFs and one for Normal (fault-free) or Healthy condition, as illustrated in Figure 6.3.

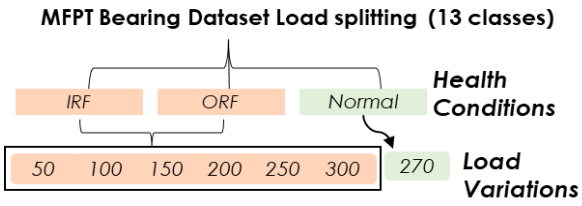


Figure 6.3: MFPT Bearing Dataset Load Factor Splitting.

This division evaluates the effect of LF on fault scenarios, emphasising IRF and ORF, which are detailed in Tables 6.1 and 6.2, respectively.

Table 6.1: IRF Dataset Splitting Per Load Factor.

Inner Fault Dataset	Code	LF (lbs/kg)	Sampling Rate (Hz)	Duration (s)	Subfile Count
baseline_2	data_normal	270/122.47	97,656	6	117
InnerRaceFault_vload_2	IRF_50	50/22.68	48,828	3	58
InnerRaceFault_vload_3	IRF_100	100/45.36	48,828	3	58
InnerRaceFault_vload_4	IRF_150	150/68.04	48,828	3	58
InnerRaceFault_vload_5	IRF_200	200/90.72	48,828	3	58
InnerRaceFault_vload_6	IRF_250	250/113.40	48,828	3	58
InnerRaceFault_vload_7	IRF_300	300/136.08	48,828	3	58

Table 6.2: ORF Dataset Splitting Per Load Factor.

Outer Fault Dataset	Code	LF (lbs/kg)	Sampling Rate (Hz)	Duration (s)	Subfile Count
baseline_2	data_normal	270/122.47	97,656	6	117
OuterRaceFault_vload_2	ORF_50	50/22.68	48,828	3	58
OuterRaceFault_vload_3	ORF_100	100/45.36	48,828	3	58
OuterRaceFault_vload_4	ORF_150	150/68.04	48,828	3	58
OuterRaceFault_vload_5	ORF_200	200/90.72	48,828	3	58
OuterRaceFault_vload_6	ORF_250	250/113.40	48,828	3	58
OuterRaceFault_vload_7	ORF_300	300/136.08	48,828	3	58

As illustrated in Figure 6.4, this segmentation process produced 117 subfiles for the Normal (fault-free) or Healthy condition baseline and 58 subfiles for each fault category (IRF and ORF). It also illustrates data segmentation for the Normal (fault-free) or Healthy condition associated with the MATLAB code. This preparatory phase sets the foundation for CLAF load-dependent fault subclass division.

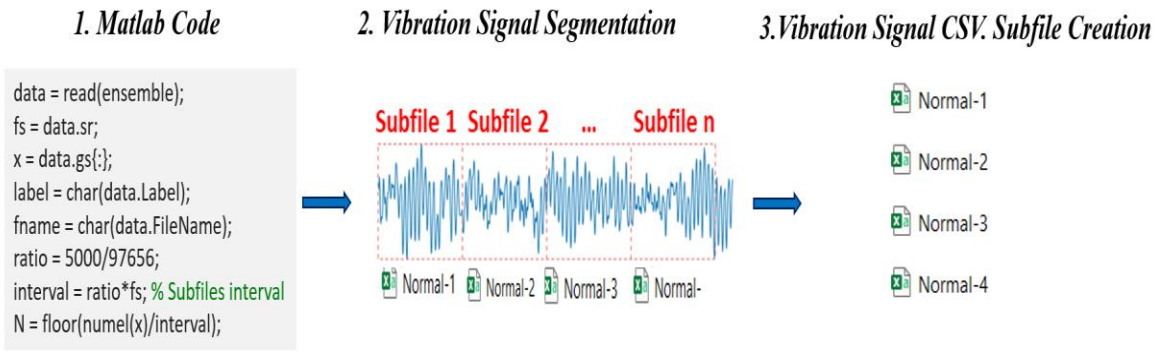


Figure 6.4: Dataset Segmentation Example on the Normal (fault-free) or Healthy Condition.

6.2.2 Multichannel Input Preparations

This section outlines the creation of three distinct data channels from the raw vibration signal for analysis. Channel 1 contains the raw segmented vibration signals, Channel 2 encodes the signals into CWT images, and Channel 3 encodes them into GADF images. The dataset was analysed using the CLAF, focusing on load-dependent subclasses: Mild, Moderate, Severe, and Normal (fault-free) or Healthy condition.

To ensure consistency and fairness in classifier performance evaluation, the datasets for Channels 1, 2, and 3—derived from the same pool of 813 subfiles—were divided in a uniform manner. Figure 6.5 illustrates the structure of the MATLAB datastore. In this

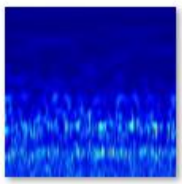
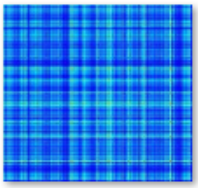
datastore, the Index column, as shown in the attached image, represents the unique subfolder name used to encode both the CWT images (stored in ImagePath_cwt) and the GADF images (stored in ImagePath_GADF). This ensures that the files corresponding to each load condition (e.g., IRF_50) are consistently linked across all three channels.

	1 Subfiles	2 LoadFactor	3 CLAF	4 Index	5 ImagePath_cwt	6 ImagePath_GADF
154	2500x1 time...	'IRF_50'	"Mild"	154	'C:\Users\Shahd\Docu...	'C:\Users\Shahd\Docu...
155	2500x1 time...	'IRF_50'	"Mild"	155	'C:\Users\Shahd\Docu...	'C:\Users\Shahd\Docu...
156	2500x1 time...	'IRF_50'	"Mild"	156	'C:\Users\Shahd\Docu...	'C:\Users\Shahd\Docu...
157	2500x1 time...	'IRF_50'	"Mild"	157	'C:\Users\Shahd\Docu...	'C:\Users\Shahd\Docu...

Figure 6.5: Datastore Structure Linking Raw Vibration Signals with CWT and GADF Images.

As illustrated in Table 6.3, this uniform approach is critical for a thorough and unbiased evaluation across all three channels. It ensures that the performance of the CNN models, which are trained on various types of encoded image data such as CWT and GADF in Channels 2 and 3, and tabular features extracted for each segment in Channel 1, is evaluated under similar conditions.

Table 6.3: Multichannel Input Preparations.

Subfiles	**Channel 1** Tabular features extracted from the raw vibration signal	**Channel 2** CWT 2D encoded image	**Channel 3** GADF 2D encoded image	**CLAF** Load-dependent fault subclasses
Normal-1	The time and frequency domain features.	 Normal-1	 Normal-1	Normal (fault-free) or Healthy condition.

6.2.2.1 Channel 1: Raw Tabular Vibration Signal

Accordingly, the dimensions of each vibration image are set to 227 x 227 x 3 and 224 x 224 x 3. These size specifications align with the input requirements of the AlexNet and ResNet-18 architectures, respectively. Figure 6.6 visually details the connection between each channel and outlines the process of creating each channel, starting with the raw vibration signal. The following subsection provides a comprehensive analysis of each channel.

TFD features, including spectral features using Autoregression will be extracted from this channel and used as the Channel 1 input in the proposed methodology. The extracted features are detailed in Section 6.2.3, as shown in Table 6.4.

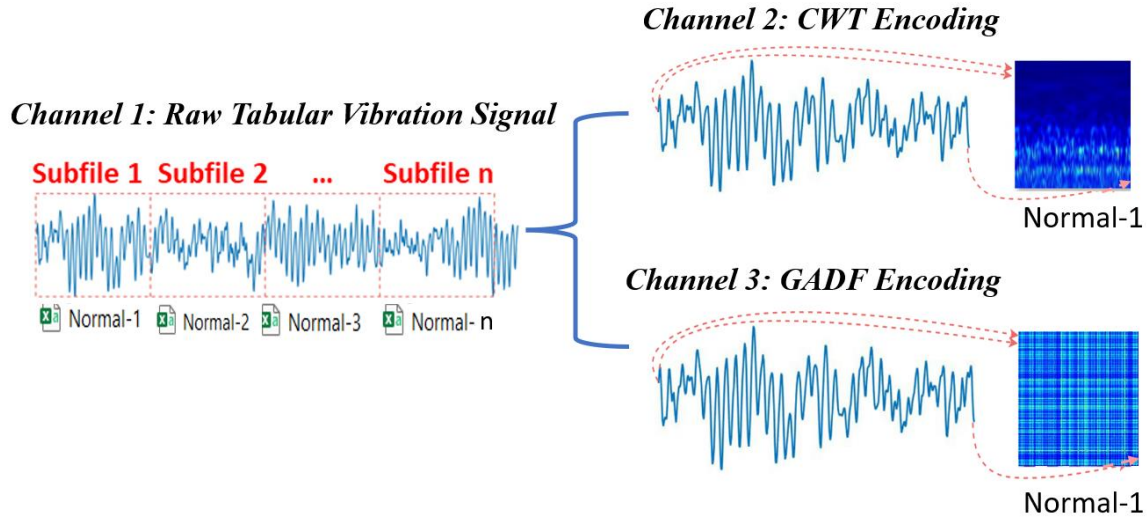


Figure 6.6: Input Channels General Overview.

6.2.2.2 *Channel 2: Continuous Wavelet Transform*

Converting vibration signals to scalogram images in MATLAB involves several systematic steps. The dataset comprises Normal (fault-free) or Healthy condition and IRF and ORF types and is first partitioned based on distinct LF conditions. Each signal subset is then processed through a Wavelet Transform (WT) using the CWT method with the 'Amor' wavelet. The transformed signals are converted into scalogram images by taking the absolute values of the CWT coefficients, flipping and scaling them. These images are then colour-mapped using the 'jet' colour map and resized to a uniform size of 224 x 224 pixels for consistency. The images of some of the generated samples are presented in Figure 6.7.

Each processed signal subset and its corresponding scalogram image are saved as an image file and a CSV file, categorically organised in folders named after the ensemble types and indices. This meticulous process is repeated for each subset of the signal, ensuring that every part of the signal is represented as a distinct image. This approach visualises the time-frequency information of vibration signals and prepares the data for further analysis, such as fault detection or ML applications.

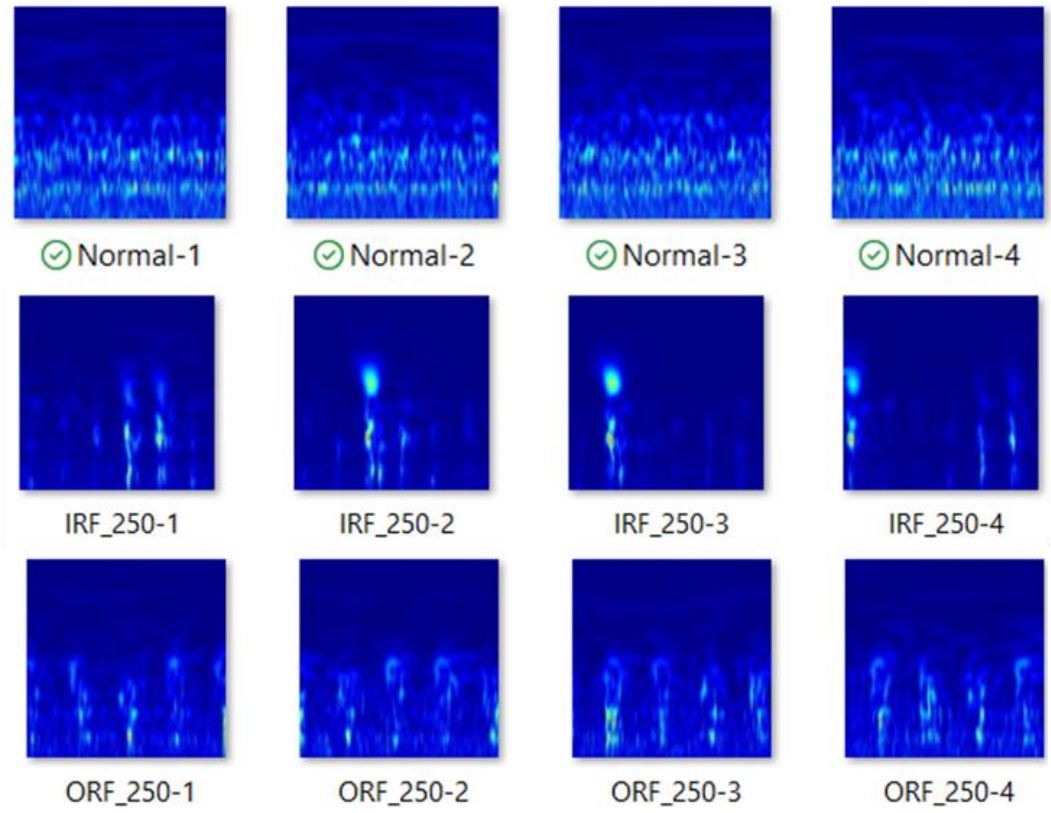


Figure 6.7: CWT 2D Encoded Image Examples From IRF, ORF and The Normal (fault-free) or Healthy Condition.

6.2.2.3 *Channel 3: Gramian Angular Difference Field (GADF)*

Creating GADF images from vibration signals involves several key steps. Initially, the time series signal is segmented into smaller subsets. For each subset, a Gramian matrix is computed using the GADF algorithm, which involves calculating the pairwise dot product of the signal and then manipulating the resulting sine and cosine matrices. The Gramian matrix is then transformed into a GADF image. This transformation includes scaling the matrix values to a range between 0 and 1, inverting this scaled matrix and resizing the image to a specified size. This process is iteratively applied to the entire signal, converting each segment into a GADF image representing the underlying time series data. This method offers an alternative way to analyse and interpret vibration signals, facilitating more profound insights into their characteristics.

GADF encoding produces distinct patterns for various health conditions, which need further analysis to validate their ability to differentiate between health conditions, as illustrated in Figure 6.8.

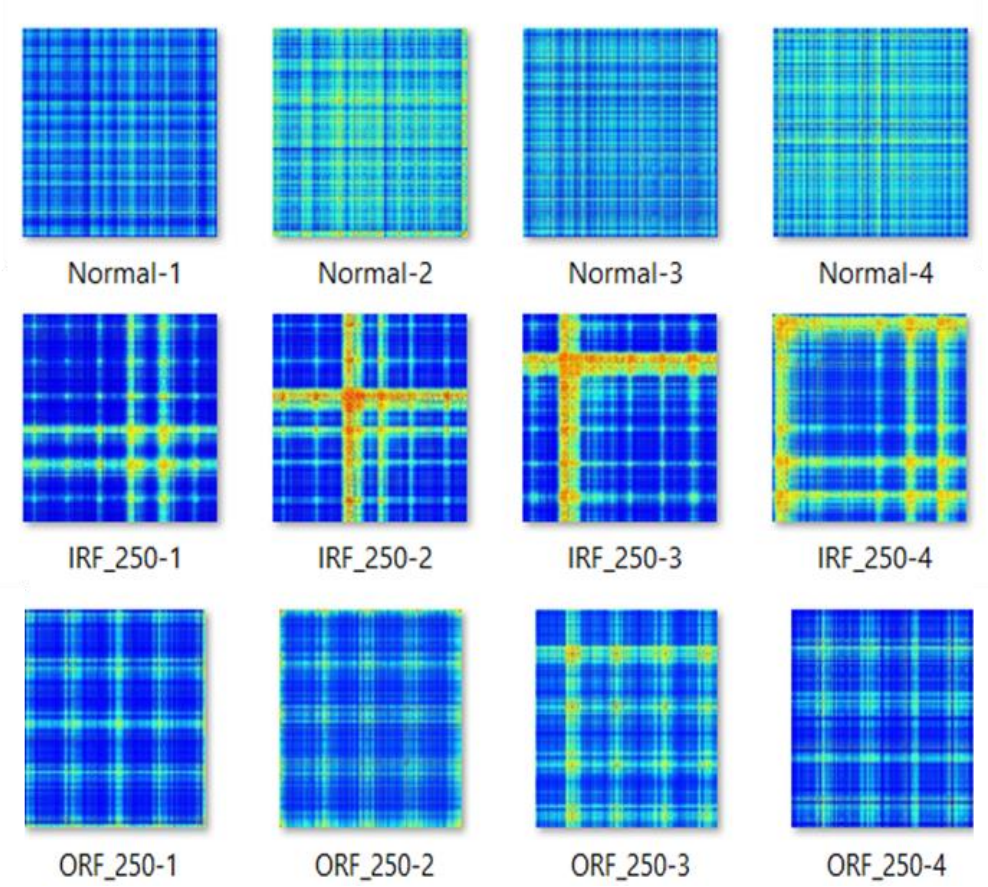


Figure 6.8: GADF 2D Encoded Image Examples From IRF, ORF and the Normal (fault-free) or Healthy Condition.

6.2.3 Feature Extraction and Classifier Selection for Channel 1 (Raw Vibration Signal):

This section conducts a one-way ANOVA test to rank the extracted general TFD features. Additionally, spectral features are extracted using an Autoregressive (AR) model of order 15 and a maximum of 5 peaks, creating 24 features. The selection of features for data representation plays a crucial role in the model's performance. Choosing the most relevant features is essential to ensure the model effectively captures the critical information related to the fault. On the other hand, including irrelevant features can sometimes result in overfitting or decreased performance (Kareem and Hur, 2022). One-way ANOVA feature selection involves comparing the means of each feature across different target classes to determine if there is a statistically significant difference. Features are ranked based on their p-values from the ANOVA test; the lower the p-value, the more likely the feature is to be influential in distinguishing between classes. These p-values are often transformed into

scores by taking the negative logarithm, with higher scores indicating more significant features for classification. In Table 6.4**,** the first column represents the extracted features, and the second column represents the one-way ANOVA scores, ranked from highest to lowest significance. Features that scored less than 26 (Peak frequency 4, Peak frequency 2, Peak frequency 5, and Total Harmonic Distortion (THD)) were not included in the current study due to their low scores, which could lead to confused training.

Table 6.4: One-way ANOVA Ranking Including Spectral Features Extracted by Autoregressive (AR) Model (b) Order Fifteen, Peak = 5.

Feature Rank	One-way ANOVA Score	Feature Rank	One-way ANOVA Score
1. Mean	316.44	13. PeakAmplitude 5	84.33
2. ShapeFactor	288.42	14. Skewness	73.13
3. PeakValue	245.43	15. PeakAmplitude 2	70.50
4. RMS	240.93	16. PeakFreq1	69.14
5. Std	240.27	17. SINAD	58.72
6. ClearanceFactor	235.23	18. S/N	58.61
7. ImpulseFactor	225.26	19. PeakAmplitude 4	51.39
8. Kurtosis	211.94	20. PeakAmplitude 3	38.77
9. CrestFactor	198.26	21. PeakFreq4	25.18
10. PeakAmplitude 1	161.22	22. PeakFreq2	17.64
11. BandPower	126.85	23. PeakFreq5	13.9307
12. PeakFrequency 3	116.80	24. THD	0

A critical analysis of classifier performance, based on the top 20 feature sets ranked by the one-way ANOVA score, provides diverse insights, as presented in Table 6.4. Various classifiers, including Support Vector Machines (SVMs), Neural Networks (NN), and Ensembles, were employed using MATLAB 2023a (MathWorks-3, 2024). The efficacy of SVM hinges on how effectively the input data are represented in this new space, a determination often made through the utilisation of diverse kernels like Linear, Polynomial (including quadratic and cubic), Gaussian, and others (Khanjani and Ezoji, 2021). The CubicSVM is a classifier that falls under the umbrella of supervised learning. SVMs are effective for high-dimensional data and are versatile in handling various structured datasets. Hence, they are widely used for classification and regression tasks (MathWorks-3, 2024)

The objective was to identify the classifier with the highest accuracy, making it a strong candidate for the proposed LD-MVSEF for Load-Specific Condition Monitoring. The training dataset, comprising 813 subfolders, was divided as follows: 60.00% for training, 20.00% for validation, and 20.00% for testing. Five-fold cross-validation was implemented

to ensure a robust performance assessment (see Table 6.5), divided by the load-dependent fault subclasses.

The Ensemble: The Boosted Trees classifier recorded a notable 94.40% accuracy, demonstrating its ability to effectively harness a larger feature set. Reducing the feature set to the top 17 had a minimal effect on accuracy, which consistently remained above 90.00%, demonstrating the classifiers' robustness and efficiency with a minor feature set. A further reduction in the feature set to the top 10, 7, and 5 revealed a nuanced interplay between feature count and accuracy. The WNN, using the top 10 features, outperformed its counterparts with a peak accuracy of 92.02%, suggesting its superior capability in working with a more compact yet pertinent feature set.

Classifier performance exhibited considerable variation in the Mild class, with the following Ensemble: Boosted Trees classifier's accuracy ranging from 89.20% with the top 17 features to 95.40% with the top 20 features. The CubicSVM classifier recorded the lowest accuracy in the Moderate class, scoring 85.70% and 91.40% with the top 7 and 5 feature subsets, respectively. In contrast, WNN achieved 91.40% with the top 10 feature subset. Remarkably, the Normal (fault-free) or Healthy condition class maintained a stable 100% accuracy across all classifiers and feature subsets, underscoring the classifiers' consistent ability to identify Normal (fault-free) or Healthy condition accurately. This consistency indicates a shared strength among the classifiers. At the same time, the variability in the load-dependent fault subclasses (Mild and Moderate classes) underscores the critical importance of appropriate feature subset selection for optimal classifier performance.

In a direct comparison, the accuracy of the CubicSVM and the WNN was closely matched. However, the selection of the top 10 features by one-way ANOVA demonstrated a well-calibrated compromise between training feature quantity and test dataset accuracy. The CubicSVM and WNN achieved 94.60% and 93.40% overall testing accuracies, respectively. Breaking this down further, the CubicSVM recorded 92.50% in the Mild class, 85.70% in the Moderate class, and 100% in the Severe class. Meanwhile, the WNN scored 90.3% in the Mild, 91.40% in the Moderate, and 91.70% in the Severe class. Notably, the CubicSVM outperformed the WNN in the Severe class by 8.30% and the Mild class by 2.20%.

Conversely, the WNN outperformed the CubicSVM in the Moderate class by 5.70%. As a result, these two classifiers were selected as the top performers for Channel 1. CubicSVM is designated as Channel 1a, while WNN is designated as Channel 1b.

Table 6.5: Classifier Performance on Channel 1 Across Distinct Feature Sets Ranked by One-Way ANOVA Feature Significance.

Classifier	ANOVA ranking	TTime[1]		Test Dataset				
		(s)	VA[2]	NA[3]	MA[4]	MoA[5]	SA [6]	Overall Accuracy
Ensemble: Boosted Trees	Top 20 >26	114.4	94.50%	100%	95.40%	88.50%	93.50%	94.40%
Ensemble: Boosted Trees	Top 17 >58.6	16.8	95.10%	100%	89.20%	85.70%	91.70%	91.70%
CubicSVM	Top 10 (a) >161	5.9	94.30%	100%	92.50%	85.70%	100%	94.60%
WNN	Top 10 (b) >161	15.7	92.20%	100%	90.30%	91.40%	91.70%	93.40%
CubicSVM	Top 7 >215	7.1	93.20%	100%	90.30%	85.70%	100%	94.00%
CubicSVM	Top 5>240	9.4	93.70%	100%	90.30%	85.70%	100%	94.00%

[1] TTime is the training time, [2] VA is validation accuracy, [3] NA is Normal (fault-free) or Healthy conditioncondition accuracy, [4]MA is Mild state accuracy, [5] MoA is Moderate state accuracy, [6] SA is Severe state accuracy

6.2.4 . Channels Classification Approaches and Training Methods

This section focuses on balancing the dataset and training each channel separately. The classifiers were evaluated on the CLAF load-dependent fault subclasses—'Mild,' 'Moderate,' 'Severe,' and ‘Normal (fault-free) or Healthy condition.' A total of 60.00% of the data was allocated for training, 20.00% for validation, and the remaining 20.00% for testing, with a Random Seed (S) of 1 set for reproducibility. The results are shown in Table 6.6:

Table 6.6: Dataset Information.

Total Dataset Description		Dataset Splitting Count Per Class
Class distribution:		Training set class distribution: Counter ({1: 278, 2: 278, 0: 278, 3:
Healthy: 0	464	278})
Mild: 1	464	Validation set class distribution: Counter ({1: 93, 2: 93, 0: 93, 3:
Moderate: 2	464	92})
Severe : 3	464	Test set class distribution: Counter ({1: 93, 2: 93, 0: 93, 3: 93})

6.2.4.1 Channel 1: CubicSVM and WNN

In this section, the top 10 extracted features from Channel 1—Mean, ShapeFactor, PeakValue, RMS, Std, ClearanceFactor, ImpulseFactor, Kurtosis, CrestFactor, and PeakAmplitude 1—were used to train classifiers, specifically the WNN and CubicSVM. These features, as presented in Table 6.4, were trained using MATLAB's Classification Learner application. The results, summarised in Table 6.7, indicate that both classifiers performed well. CubicSVM achieved a slightly higher overall accuracy of 96.28%, while WNN reached 94.95%. Both classifiers showed perfect accuracy in the Normal (fault-free) or Healthy condition and Severe states, with CubicSVM displaying superior performance in the Mild and Moderate states. Despite WNN requiring more time for training (48.63 s compared to CubicSVM's 26.71 s), both models demonstrated strong classification capabilities across the load-dependent fault subclasses.

Table 6.7: Channel 1 Classifiers Training.

Classifier	TTime [1] (s)	VA [2]	Test Dataset NA [3]	MA [4]	MoA [5]	SA [6]	Overall Accuracy
a. CubicSVM	26.71	96.80%	100%	89.36%	95.74%	100%	96.28%
b. WNN	48.63	96.70%	100%	95.74%	84.04%	100%	94.95%

[1] TTime is the training time, [2] VA is validation accuracy, [3] NA is the Normal (fault-free) or Healthy condition accuracy, [4]MA is the Mild state accuracy, [5] MoA is the Moderate state accuracy, [6] SA is the Severe state accuracy.

6.2.4.2 Channels 2 and 3: Pre-trained CNN Selection

In this section, Channel 2 and Channel 3 were utilised to exploit the transfer learning capabilities of CNNs, training on uniquely encoded images: CWT for Channel 2 and GADF for Channel 3. AlexNet and ResNet-18, trained initially on the extensive ImageNet dataset for classifying 1,000 distinct image classes, were repurposed to fit the specific dataset requirements in the current study. These pre-trained networks were fine-tuned to classify four distinct load-dependent fault subclasses using a learning rate of 0.0001, a mini-batch size of 30, and a maximum of 5 epochs, with validation performed every 30 iterations. The

effectiveness of these adapted models was then assessed based on their accuracy with the test dataset.

For both channels, the signal-encoded RGB images, with a resolution of 224 × 224 pixels, required uniform image preprocessing steps before the training of the CNNs could commence. The images were resized accordingly in the case of AlexNet, which has an original input dimension of 227 x 227 x 3. Subsequently, the network's final fully connected layer was replaced with a new one containing four neurons, corresponding to the categories of Normal (fault-free) or Healthy condition, Mild, Moderate, and Severe, to adapt to the classification needs of the study.

Similarly, ResNet-18, initially trained on the ImageNet dataset, required the images to be adjusted to their input size 224 x 224 x 3. Following the study's requirements, the last layer of ResNet-18 was replaced with a 4-neuron layer to fine-tune the model for the CLAF load-dependent fault subclasses—'Mild,' 'Moderate,' 'Severe,' and 'Normal (fault-free) or Healthy condition.'.

- *Channel 2: CWT-Enhanced Transfer Learning with ResNet-18 and AlexNet in CNN Models*

Table 6.8 compares the performance of ResNet-18 and AlexNet on Channel 2, focusing on CWT signal-encoded images.

Table 6.8: Pre-trained CNN Performance on Channel 2 (CWT Signal Encoded Images).

Pre-trained CNN	Validation		Test Dataset				
	Training Time (min)	Validation Accuracy	(Normal) Accuracy	(Mild) Accuracy	(Moderate) Accuracy	(Severe) Accuracy	Overall Accuracy
1. ResNet-18	17.35	97.55%	100%	95.74%	100%	100%	98.94%
2.AlexNet	7.20	97.28%	100%	96.81%	96.81%	100%	98.40%

ResNet-18 required 17.35 min of training to achieve a validation accuracy of 97.55% and an overall test accuracy of 98.94%. It performed well across all fault subclasses, reaching 100% accuracy in Normal (fault-free) or Healthy, Moderate, and Severe conditions and

95.74% in the Mild condition. AlexNet, on the other hand, completed training in 7.20 min, achieving a validation accuracy of 97.28% and a test accuracy of 98.40%. It also maintained 100% accuracy in the Normal (fault-free) or Healthy and Severe conditions, with 96.81% accuracy in both Mild and Moderate conditions.

Although ResNet-18 had a slight edge in overall test accuracy and performance in Mild conditions, AlexNet required less than half the training time with only a minimal difference in accuracy. Considering its efficiency and competitive accuracy, AlexNet has been chosen to handle CWT signal-encoded images on Channel 2.

- *Channel 3: GADF-Enhanced Transfer Learning with Residual Network-18 (ResNet-18) and AlexNet in CNN Models*

Table 6.9 compares the performance of two pre-trained CNNs: ResNet-18 and AlexNet on Channel 3, which encodes images using the GADF signal. Both networks used a learning rate of 0.0001 and validated every five epochs.

ResNet-18 completed training in 18.5 min, achieving a validation accuracy of 96.47% and an overall test accuracy of 95.21%. It performed strongly in the Normal (fault-free) or Healthy and Severe conditions, achieving 100% and 98.94% accuracy, respectively. However, its performance in the Mild and Moderate conditions was lower, with 89.36% accuracy in Mild and 92.55% in Moderate conditions.

In contrast, AlexNet completed training in 7.53 min, matching ResNet-18's validation accuracy of 96.47%, but outperformed ResNet-18 on the test dataset with an overall accuracy of 98.67%. AlexNet achieved 100% accuracy in the Normal (fault-free) or Healthy and Severe conditions, and also showed stronger performance in the Mild and Moderate conditions, with accuracies of 96.81% and 97.87%, respectively.

Although both models performed well, AlexNet demonstrated a better overall test performance, particularly in Mild and Moderate conditions, requiring less training time. Given its balance of efficiency and accuracy, AlexNet is the preferred model for handling GADF signal-encoded images in Channel 3.

Table 6.9: Pre-trained CNN Performance on Channel 3 (GADF Signal Encoded Images).

Pre-trained CNN	Validation		Test Dataset				
	Training Time (min)	Validation Accuracy	(Normal) Accuracy	(Mild) Accuracy	(Moderate) Accuracy	(Severe) Accuracy	Overall Accuracy
1. ResNet-18	18.5	96.47%	100%	89.36%	92.55%	98.94%	95.21%
2.AlexNet	7.53	96.47%	100%	96.81%	97.87%	100%	98.67%

6.2.5 Single-Channel Performance Analysis

Figure 6.9 presents the load-dependent subclass accuracy assessment for each channel. All classifiers performed well for the Normal (fault-free) or Healthy and Severe condition classes, with each achieving 100% accuracy. This high performance, while expected for these extreme conditions where the patterns are more distinct and more straightforward to differentiate, could be attributed to the more apparent fault or non-fault signals in the data. The clear distinction between the Healthy and Severe conditions allowed the classifiers to identify them without error consistently.

In the Mild condition class, Channel 2, using CWT (AlexNet), showed the best performance with an accuracy of 96.81%, followed closely by Channel 3 (GADF with AlexNet) at 95.74%. However, Channels 1a and 1b, which utilise CubicSVM and WNN classifiers, struggled more in detecting Mild conditions, with accuracies of 89.36% and 84.04%, respectively. This suggests that the Mild class presents more challenges for accurate classification, likely due to the less distinct signal patterns associated with early or mild faults.

While performance remained strong for the Moderate condition class, there were noticeable differences between the channels. Channel 3 (GADF with AlexNet) showed the highest accuracy at 97.87%, followed by Channel 2 (CWT with AlexNet) at 96.81% and Channel 1b (WNN) at 95.74%. Channel 1a (CubicSVM) exhibited the lowest performance in this category, with an accuracy of 84.04%. This indicates that Moderate conditions are more difficult to classify than extremes as the signal patterns become less clear.

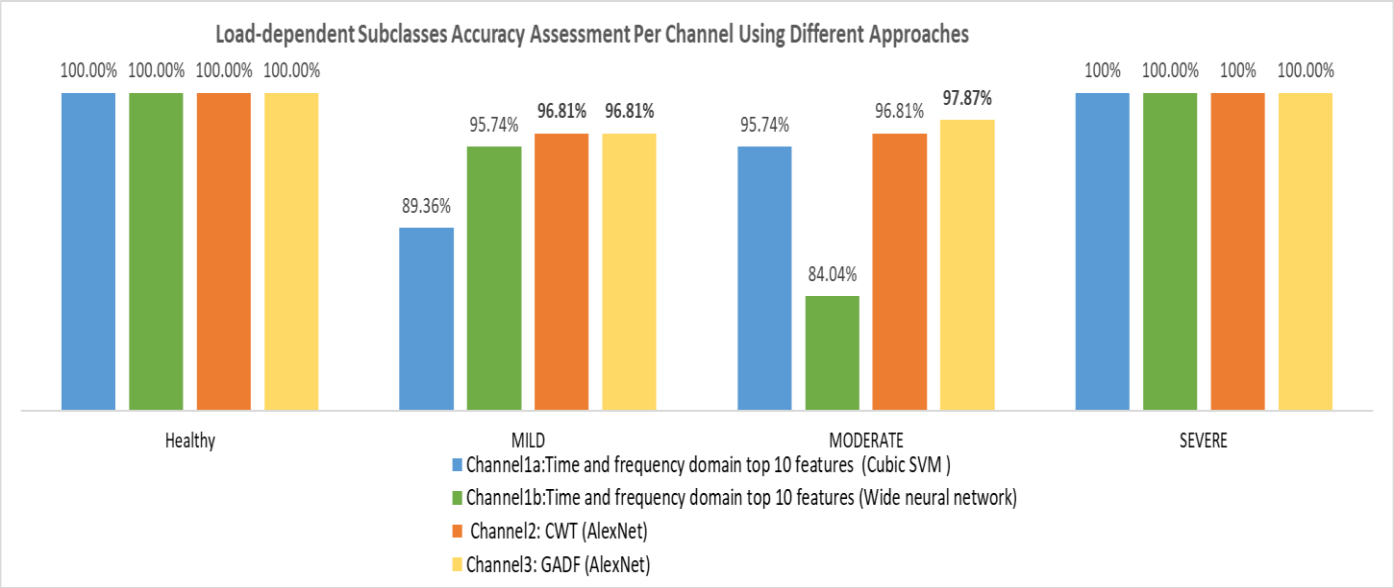


Figure 6.9: CLAF Load-Dependent Fault Subclass Accuracy Assessment Per Channel Using Different Approaches.

6.2.6 Decision Fusion

6.2.6.1 Weighted Decision Fusion Approach (Alternative Setting)

This section explores two weighted decision fusion approaches across three distinct alternatives, each employing two different systems of weight allocation for decision fusion. In the following alternatives, Weighting System 1 refers to Adaptive Weighting, where weights are dynamically assigned based on each classifier's accuracy for specific conditions. Higher-performing channels receive greater weight for the conditions in which they excel. Meanwhile, Weighting System 2 refers to Equal Weighting, where equal weights are assigned to all channels across all conditions, ensuring no single classifier dominates the decision-making process.

In all scenarios, the sum of the weights corresponds to one of the conditions, maintaining balance in the decision fusion system and ensuring a proportional contribution from each classifier based on their accuracy. The weight allocation in the decision fusion systems of the LD-MVSEF approach is justified by the accuracy assessments per channel for the different load-dependent fault subclasses, as illustrated in Figure 6.9. This chart highlights the performance of each classifier under Mild, Moderate, and Severe conditions, directly informing the weight distribution across the systems. Table 6.10 outlines the alternative setting and decision fusion weighting system :

Table 6.10: Alternative Setting and Decision Fusion Weighting System.

[1] TFD is the time and frequency domain extracted features

- Alternative 1In Weighting System 1, Channel 1b (WNN with TFD features) is assigned

	Channel No.	Input	Classifier	Weighting System 1				Weighting System 2			
				Healthy	**Mild**	**Moderate**	**Severe**	**Healthy**	**Mild**	**Moderate**	**Severe**
	Alternative No.			1.1 (TFDb -CWT)				1.2 (TFDb-CWT)			
Alternative 1	1b	TFD	WNN	0.5	0.4	0.3	0.5	0.5	0.5	0.5	0.5
	2	CWT	AlexNet	0.5	0.6	0.7	0.5	0.5	0.5	0.5	0.5
	Alternative No.			2.1 (TFDa -CWT)				2.2 (TFDa -CWT)			
Alternative 2	1a	TFD	CubicSVM	0.5	0.3	0.4	0.5	0.5	0.5	0.5	0.5
	2	CWT	AlexNet	0.5	0.7	0.6	0.5	0.5	0.5	0.5	0.5
	Alternative No.			3.1 (TFDa -CWT-GADF)				3.2 (TFDa -CWT-GADF)			
Alternative 3	1a	TFD	CubicSVM	0.33	0.2	0.2	0.33	0.33	0.33	0.33	0.33
	2	CWT	AlexNet	0.33	0.4	0.4	0.33	0.33	0.33	0.33	0.33
	3	GADF	AlexNet	0.33	0.4	0.4	0.33	0.33	0.33	0.33	0.33

a weight of 0.5 for the Healthy and Severe conditions, reflecting its accuracy of 100% in these categories. For the Mild and Moderate conditions, Channel 1b receives lower weights of 0.4 and 0.3, respectively, corresponding to its accuracies of 95.74% and 84.04%. In comparison, Channel 2 (CWT with AlexNet) performed better in Mild and Moderate conditions, with accuracies of 96.81% and 97.87%, and is therefore assigned higher weights of 0.6 and 0.7 in these categories. Both channels receive equal weights of 0.5 for the Healthy and Severe conditions. Under Weighting System 2, equal weights of 0.5 are assigned to both channels across all conditions, ensuring an equal contribution from WNN and AlexNet.

- Alternative 2: For Weighting System 1, Channel 1a (CubicSVM with TFD features) is given a weight of 0.5 for the Healthy and Severe conditions due to its 100% accuracy. For the Mild condition, Channel 1a receives a lower weight of 0.3, based on its accuracy of 89.36%, while Channel 2 (CWT with AlexNet) achieves a higher accuracy of 96.81% and is given a weight of 0.7. In the Moderate condition, Channel 1a achieves an accuracy of 95.74% and is assigned a weight of 0.4, while Channel 2 is given a slightly higher

weight of 0.6 due to its accuracy of 96.81%. Under Weighting System 2, both channels receive equal weights of 0.5 across all conditions, maintaining a balanced approach between CubicSVM and AlexNet.

- Alternative 3: In Weighting System 1, where three channels are used, weights are distributed according to each channel's performance. Channel 1a (CubicSVM with TFD features) is given a lower weight of 0.2 for the Mild and Moderate conditions, where it achieved 89.36% and 84.04% accuracy. Channel 2 (CWT with AlexNet) and Channel 3 (GADF with AlexNet) performed better in these categories, with 96.81% and 97.87% accuracy, and are therefore assigned higher weights of 0.4 each. All three channels achieved 100% accuracy for the Healthy and Severe conditions and were given equal weights of 0.33. Under Weighting System 2, all three channels are assigned equal weights of 0.33 across all conditions, providing a balanced decision-making approach.

6.2.6.2 Choose the Highest-Performing Weighted Decision Fusion Approach

The following outlines the weighing system alternatives. Each alternative within the weighted decision fusion system was meticulously designed, with weights assigned to each classifier based on their demonstrated accuracy under specific load-dependent fault subclasses. This approach helps establish a robust and precise condition-monitoring tool. Furthermore, the experiment was repeated five times, changing the training, validation, and test datasets for each run. This required training the selected single models for five runs across each channel, using a Random Seed (S) for reproducibility, varying from 1 to 12.

- Channel 1:
  - a) Cubic Support Vector Machine (CubicSVM)
  - b) Wide Neural Network (WNN)

Both classifiers were trained using the top 10 features selected by one-way ANOVA, explained in section 6.2.4.1. CubicSVM took an average of 15.68 s and achieved an average accuracy of 96.44%. WNN, on the other hand, required an average training time of 25.18 s and yielded an average accuracy of 97.50% (as shown in Tables 6.11 and 6.12). Below are the details of each run.

Table 6.11: a) CubicSVM 5 Runs.

Run	Classifier	TTime[1] (s)	VA[2]	Test Dataset NA[3]	MA[4]	MoA[5]	SA[6]	Overall Accuracy
1, S1	CubicSVM	26.71	96.80%	100%	89.36%	95.74%	100%	96.28%
2, S3	CubicSVM	12.87	95.80%	100%	88.36%	98.94%	100%	96.83%
3, S 6	CubicSVM	11.56	97.50%	100%	91.49%	95.74%	100%	96.81%
4, S9	CubicSVM	10.85	95.50%	100%	89.36%	98.94%	100%	97.08%
5, S12	CubicSVM	16.41	95.90%	100%	95.74%	100%	100%	95.18%

[1] TTime is the training time, [2] VA is validation accuracy, [3] NA is the Normal (fault-free) or Healthy condition accuracy, [4]MA is the Mild state accuracy, [5] MoA is the Moderate state accuracy, [6] SA is the Severe state accuracy.

Table 6.12: b) Wide Neural Network 5 Runs.

Run	Classifier	TTime[1] (s)	VA[2]	Test Dataset NA[3]	MA[4]	MoA[5]	SA[6]	Overall Accuracy
1, S1	WNN	48.63	96.70%	100%	95.74%	84.04%	100%	94.95%
2, S3	WNN	18.74	97.20%	100%	91.49%	100%	100%	97.87%
3, S 6	WNN	16.75	97.50%	100%	94.68%	100%	100%	98.67%
4, S9	WNN	19.75	97.00%	100%	89.36%	98.94%	100%	97.08%
5, S12	WNN	22.07	97.50%	100%	95.74%	100%	100%	98.94%

[1] TTime is the training time, [2] VA is validation accuracy, [3] NA is the Normal (fault-free) or Healthy condition accuracy, [4]MA is the Mild state accuracy, [5] MoA is the Moderate state accuracy, [6] SA is the Severe state accuracy.

- Channel 2: AlexNet was trained using the Stochastic Gradient Descent with Momentum (SGDM) solver with a Learning Rate (LR) of 0.0001, a mini-batch size of 30, and a maximum of 5 epochs. Validation was performed every 30 iterations to monitor performance during training. The training was conducted on CWT-encoded images for five runs, with training times varying across each run. On average, AlexNet took 9.36 min to complete the training and achieved an average accuracy of 97.37%. Below is the performance breakdown for each run (see Table 6.13).
- Channel 3: AlexNet was trained on GADF-encoded images over five runs with the same solver settings. The average training time for AlexNet on Channel 3 was 9.47 min, and the model achieved an average accuracy of 95.49%. Below (Table 6.14) is the performance breakdown for each run.

Table 6.13: Accuracy of Channel 2 (AlexNet) Over 5 Runs on CWT Images.

Run	Pre-trained CNN (CWT)	TTime [1] (min)	VA[2]	Test Dataset NA[3]	MA[4]	MoA[5]	SA [6]	Overall Accuracy
1, S1	AlexNet	7.20	97.28%	100%	96.81%	96.81%	100%	98.41%
2, S3	AlexNet	10.32	96.20%	100%	97.87%	97.87%	100%	98.94%
3, S 6	AlexNet	7.54	97.83%	100%	90.43%	97.87%	100%	97.08%
4, S9	AlexNet	7.34	96.74%	100%	91.49%	91.49%	100%	95.75%
5, S12	AlexNet	12.4	98.64%	100%	94.62%	100%	100%	98.66%

[1] TTime is the training time, [2] VA is validation accuracy, [3] NA is the Normal (fault-free) or Healthy condition accuracy, [4]MA is the Mild state accuracy, [5] MoA is the Moderate state accuracy, [6] SA is the Severe state accuracy.

Table 6.14: Accuracy of Channel 3 (AlexNet) Over 5 Runs on GADF Images.

Run	Pre-trained CNN (GADF)	TTime [1] (min)	VA[2]	Test Dataset NA[3]	MA[4]	MoA[5]	SA [6]	Overall Accuracy
1, S1	AlexNet	7.53	96.47%	100%	96.81%	97.87%	100%	98.67%
2, S3	AlexNet	11.49	96.20%	100%	92.55%	95.74%	100%	97.07%
3, S 6	AlexNet	7.56	93.21%	100%	98.94%	80.85%	100%	94.95%
4, S9	AlexNet	8.20	96.20%	100%	91.49%	91.49%	100%	95.75%
5, S12	AlexNet	11.59	95.65%	100%	87.00%	86.17%	100%	93.29%

[1] TTime is the training time, [2] VA is validation accuracy, [3] NA is the Normal (fault-free) or Healthy condition accuracy, [4]MA is the Mild state accuracy, [5] MoA is the Moderate state accuracy, [6] SA is the Severe state accuracy.

Consequently, the results of the five decision fusion runs were meticulously recorded (see *APPENDIX 4)*, and the derived scores were obtained. The experiment was repeated ten times, and the outcomes for each class and overall accuracy are discussed below. The experiment's alternatives are detailed in Table 6.9: 1.1 (TFDb - CWT), 1.2 (TFDb-CWT), 2.1 (TFDa - CWT), 2.2 (TFDa - CWT), 3.1 (TFDa - CWT - GADF), and 3.2 (TFDa - CWT - GADF).

It was observed that all channels performed well in the Healthy and Severe classes. However, they encountered challenges in classifying the Mild and Moderate classes. The analysis conducted across the Mild and Moderate classes on the test dataset revealed the following :

- Mild Class: Alternative 3.1 (TFDa - CWT - GADF) demonstrated the highest average accuracy of 97.02%, while Alternative 1.2 (TFDb - CWT) had a lower average accuracy of 94.25%. Although all alternatives performed reasonably well, there were noticeable

variations in the classification of the Mild class, which often presents more subtle patterns that can be challenging to detect (see Table 6.15).

- Moderate Class: Alternative 3.1 again delivered consistent results, with an average accuracy of 99.15%. Alternative 1.1 (TFDb - CWT) showed high consistency, achieving 100% accuracy across all five runs. However, Alternative 3.2 (TFDa - CWT - GADF) recorded an average accuracy of 98.01%, with some fluctuations in performance. This suggests that while all alternatives were effective, there were slight variations in handling the Moderate class, where the signal patterns are less distinct (see Table 6.16).

Table 6.15: Decision Fusion Mild Class Analysis on Test Accuracy Over the 5 Runs.

Alternatives	1	2	3	4	5	Min	Max	Avg.
1.1 (TFDb -CWT)	94.68%	95.74%	95.74%	94.68%	96.81%	94.68%	96.81%	95.53%
1.2 (TFDb-CWT)	95.74%	90.43%	94.68%	94.68%	95.74%	90.43%	95.74%	94.25%
2.1 (TFDa -CWT)	95.74%	96.81%	90.43%	95.74%	93.62%	90.43%	96.81%	94.47%
2.2 (TFDa -CWT)	96.81%	96.81%	92.55%	95.74%	94.68%	92.55%	96.81%	95.32%
3.1 (TFDa -CWT-GADF)	95.74%	95.74%	98.94%	95.74%	98.94%	95.74%	98.94%	**97.02%**
3.2 (TFDa -CWT-GADF)	95.74%	95.74%	96.81%	95.74%	97.87%	95.74%	97.87%	96.38%

Table 6.16: Decision Fusion Moderate Class Analysis on Test Accuracy Over the 5 Runs.

Alternatives	1	2	3	4	5	Min	Max	Avg.
1.1 (TFDb -CWT)	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
1.2 (TFDb-CWT)	100.00%	97.87%	100.00%	100.00%	100.00%	97.87%	100.00%	99.57%
2.1 (TFDa -CWT)	97.87%	97.87%	95.74%	98.94%	100.00%	95.74%	100.00%	98.08%
2.2 (TFDa -CWT)	97.87%	98.94%	95.74%	98.94%	100.00%	95.74%	100.00%	98.30%
3.1 (TFDa -CWT-GADF)	98.94%	100.00%	98.94%	100.00%	97.87%	97.87%	100.00%	**99.15%**
3.2 (TFDa -CWT-GADF)	97.87%	94.29%	98.94%	100.00%	98.94%	94.29%	100.00%	98.01%

The overall test accuracy for each run, along with the corresponding averages for the six decision fusion alternatives and training times, are shown in Table 6.17, where the training time of each alternative is shown in Table 6.18:

1.1 (TFDb - CWT): Accuracies ranged from 98.67% to 99.07%, with an average accuracy of 98.86%. The average training time for this alternative was 9 min 23 s.

2.1 (TFDa - CWT): Accuracies spanned from 97.08% to 98.94%, resulting in an average accuracy of 98.40%. The average training time was also 9 min 23 s.

2.2 (TFDa - CWT): Accuracies ranged from 97.07% to 98.94%, averaging 98.46%. The average training time for this alternative was also 9 min 13 s.

3.1 (TFDa - CWT - GADF): This alternative recorded accuracies ranging from 98.67% to 99.47%, with an average of 99.04%. The training time was longer, averaging 18 min 30 s.

3.2 (TFDa - CWT - GADF): The minimum accuracy recorded was 97.51%, the maximum was 99.20%, and the average accuracy was 98.60%. The average training time for this alternative was also 18 min 30 s.

Table 6.17: Decision Fusion Overall Test Accuracy Over the 5 Runs.

Alternatives	1	2	3	4	5	Avg.
1.1 (TFDb -CWT)	98.67%	98.94%	98.94%	98.67%	99.07%	98.86%
1.2 (TFDb-CWT)	98.94%	97.08%	98.67%	98.67%	98.67%	98.40%
2.1 (TFDa -CWT)	98.40%	98.67%	96.54%	98.67%	98.54%	98.16%
2.2 (TFDa -CWT)	98.67%	98.94%	97.07%	98.67%	98.94%	98.46%
3.1 (TFDa -CWT-GADF)	98.67%	98.94%	99.47%	98.94%	99.20%	**99.04%**
3.2 (TFDa - CWT - GADF)	98.40%	97.51%	98.94%	98.94%	99.20%	98.60%

Table 6.18: Decision Fusion Overall Training Time Over the 5 Runs.

Alternatives	1	2	3	4	5	Avg.
1.1 (TFDb -CWT)	0:08:01	0:10:38	0:07:49	0:07:40	0:12:47	0:09:23
1.2 (TFDb-CWT)	0:08:01	0:10:38	0:07:49	0:07:40	0:12:47	0:09:23
2.1 (TFDa -CWT)	0:07:39	0:10:32	0:07:44	0:07:31	0:12:41	0:09:13
2.2 (TFDa -CWT)	0:07:39	0:10:32	0:07:44	0:07:31	0:12:41	0:09:13
3.1 (TFDa -CWT-GADF)	0:15:11	0:22:01	0:15:18	0:15:43	0:24:16	**0:18:30**
3.2 (TFDa - CWT - GADF)	0:15:11	0:22:01	0:15:18	0:15:43	0:24:16	0:18:30

Based on these observations, Alternative 3.1 (TFDa - CWT - GADF) consistently offers strong performance, making it a reliable and effective option for condition monitoring across various severity levels. It achieved the highest overall average accuracy of 99.04% ± 0.22%, based on five runs, with an average training time of 18 min 30 s. The uncertainty reflects the minor variability observed in these runs. Its robust performance in the Mild class, with an average testing accuracy of 97.20% ± 1.75%, and 99.15% ± 0.89% testing accuracy in the Moderate class, demonstrates its suitability for load-specific condition monitoring and precise fault classification under different LF conditions.

6.3 Summary

This chapter introduced the LD-MVSEF framework, designed to enhance load-specific condition monitoring using the MFPT bearing dataset. The LD-MVSEF framework

incorporates load-dependent fault subclasses derived from the CLAF, shifting the focus from traditional fault classification to a load-specific approach. It integrates three distinct channels for analysis: Channel 1 extracts TFD features, including spectral features using Autoregression; Channel 2 converts vibration signals into CWT images; and Channel 3 encodes the signals into GADF 2D images.

Each channel was trained over five separate runs, and the best-performing classifiers were selected based on their accuracy in classifying four load-dependent fault subclasses: Healthy, Mild, Moderate, and Severe. In Channel 1, CubicSVM and WNN classifiers achieved average accuracies of 96.43% ± 0.76% and 97.50% ± 1.60%, respectively. For Channels 2 and 3, pre-trained AlexNet and ResNet-18 models were used, with AlexNet performing the best, achieving accuracies of 97.76% ± 1.33% on Channel 2 (CWT images) and 95.95% ± 2.05% on Channel 3 (GADF images).

One of the main challenges observed was the classification of the Mild and Moderate fault subclasses, which presented subtler signal variations compared to the Healthy and Severe conditions. The LD-MVSEF framework addressed these challenges by employing a weighted decision fusion approach, where decisions were tailored according to the strengths of each channel for specific fault subclasses. For instance, Channel 2 (CWT with AlexNet) performed well in classifying the Moderate class, while Channel 3 (GADF with AlexNet) showed high accuracy in both the Mild and Moderate conditions. By assigning dynamic weights to each classifier based on their strengths, the LD-MVSEF framework improved the classification of these more challenging subclasses.

The proposed weighted decision fusion approach demonstrated excellent performance across all fault conditions. Alternative 3.1 (TFDa - CWT - GADF) achieved the highest overall accuracy of 99.04% ± 0.22% across five runs, with an average training time of 18 min 30 s. This approach minimised the limitations of individual classifiers and effectively handled load-specific fault classification.

The contribution of this chapter has been to propose a novel LD-MVSEF method for load-specific condition monitoring, which encompasses the following sub-contributions:

1) Multimodal fusion and decision fusion: The LD-MVSEF framework combines features from GADF, CWT, and TFD data to enhance the Load-Dependent Fault Classification builds on the CLAF. By integrating these complementary patterns and

using a weighted decision fusion approach, the framework assigns classifier weights based on performance, helping to improve accuracy, particularly in the more challenging Mild and Moderate fault subclasses.

2) Comprehensive data integration: Insights from both 1D vibration signals and 2D RGB images (CWT and GADF) were combined to capture complementary patterns, enhancing the classification.

Chapter 7: Hybrid Graph-CNN Decision Fusion (HG-CDF) for Load-Dependent Fault Classification

7.1 Proposed Methodology

The proposed methodology of Hybrid Graph-CNN Decision Fusion (HG-CDF) for Load-Dependent Fault Classification builds on the Customised Load Adaptive Framework (CLAF) to enhance fault classification for the Machinery Failure Prevention Technology (MFPT) bearing dataset introduced in Chapter 5. This approach utilises Graph Convolutional Networks (GCNs) to transform tabular vibration signal data into graph structures, allowing for more effective load-specific condition monitoring. The methodology involves three key stages: First, the tabular vibration data are preprocessed to reflect different load conditions, and the k-Nearest Neighbour Graphs (k-NNGs) method is employed to convert the data into graph form, where nodes represent time-series points and edges are based on signal similarity. Second, the GCN model is optimised through the Taguchi Design of Experiments (DOE), where different configurations are tested to determine the optimal parameters, including the number of epochs, learning rate, and k-value. Lastly, recognising the limitations of GCNs in handling certain fault classes, particularly the Mild class, a hybrid approach is introduced, integrating One-Dimensional Convolutional Neural Networks (1D-CNNs) with a GCN in a decision fusion mechanism to further improve classification accuracy across all fault subclasses. Google Colab was used to implement the proposed methodology. For the full code, see (*APPENDIX 4)*.

7.1.1 Hybrid Graph-CNN Decision Fusion (HG-CDF) for Load-Dependent Fault Classification

The HG-CDF for Load-Dependent Fault Classification for Transforming Tabular Data into Graph Structures methodology involves three main steps. These are presented in Figure 7.1 and described in detail below:

1. **Step 1: Dataset Introduction and Preprocessing:**

   The research was initiated by preprocessing the MFPT bearing dataset utilising the CLAF to create load-dependent fault subclasses—'Mild,' 'Moderate,' 'Severe,' and 'Normal (fault-free) or Healthy condition'—tailored to specific datasets, forming the foundation for subsequent analysis. Essential steps included the following:

1.1. Input Dataset Preparation: The MFPT bearing dataset was preprocessed with the CLAF methodology to refine the input data to create load-dependent fault subclasses—'Mild,' 'Moderate,' 'Severe,' and 'Normal (fault-free) or Healthy condition'—tailored to specific datasets, forming the foundation for subsequent analysis.

1.2. Feature Extraction and Selection: Through the CLAF, critical Time and Frequency Domain (TFD) features were extracted and selected to serve as robust inputs for the model.

1.3. Dataset Cleaning and Splitting: The dataset underwent cleaning and was stratified into training (60.00%), validation (20.00%), and testing (20.00%) sets to ensure a balanced representation of classes across each dataset.

2. **Step 2: Deep Learning Model Preparation:**

The model development phase is divided into two distinct paths. Path 2.1 (refer to Figure 7.1) involves constructing GCN models using a Taguchi L09 matrix to optimise hyperparameters methodically. Path 2.2 (refer to Figure 7.1) retains the tabular dataset for formulating the 1D-CNN model. For the GCN pathway, a detailed preparation process includes variable k-Nearest Neighbours (kNN) settings for producing graphs and the creation of masks for training, validation, and testing inputs. The GCN models are diligently built and subjected to a stringent training and validation routine, fostering iterative enhancements from the feedback. The Taguchi method refines the model configurations, precipitating a re-evaluation of experiments to delineate and endorse the most efficacious GCN strategy. Conversely, Path 2.2 transforms tabular data into tensors and advances with the training of 1D-CNNs, guided by the optimal epochs and Learning Rates (LRs) derived from the Taguchi experiments. The following concerns the branching for the GCN and the 1D-CNN:

2.1 Graph Convolutional Network (GCN): This branch has two main sub-stages. The first sub-stage, detailed in pathway 2.1.1 (refer to Figure 7.1), relates to Taguchi's Design of Experiments (DOE) and parameter configurations. The second sub-stage, described in pathway 2.1.2 (refer to Figure 7.1), involves building the GCN model.

2.1.1. Taguchi Method Preparation (L09) Matrix*:* In this step, Taguchi's design of the experimental approach was implemented using three control factors: k-NNG number of neighbours, number of epochs, and LR. The first factor involves three steps:

- Nodes were created to represent each dataset instance.
- Edges were established using the kNN approach with *k* set to 3, 4, and 5.
- Masks were generated to demarcate training, validation, and test sets.

2.1.2. GCN Model building: GCN models were formulated.

2.2 One-Dimensional Convolutional Neural Network (1D-CNN): This is the second branch, and it focuses on building the 1D-CNN architecture.

2.2.1. 1D-CNN Model Formulation:

- Data were prepared as tensors, facilitating subsequent Data Loader object creation in PyTorch.
- 1D-CNN models were developed and then trained and validated.

3. Step 3: Comparative Model Evaluation and Fusion Approach:

The methodology encompassed a robust comparative analysis of models based on the test dataset:

3.1. Convolutional Network (GCN) Design of Experiment using Taguchi Method and Signal-to-Noise (S/N) Ratio: Multiple GCN configurations are tested and optimised using the Taguchi Method and S/N ratio to enhance classification accuracy and minimise noise.

3.2. One-Dimensional Convolutional Neural Network (1D-CNN) Training, Validation and Testing: A 1D-CNN model is trained on the vibration data to capture sequential patterns, providing an alternative classification method that complements the GCN's structure-based approach.

3.3. Hybrid Graph-CNN Decision Fusion (HG-CDF) for Load-Dependent Fault Classification: The GCN and 1D-CNN models are combined using a decision fusion mechanism, leveraging their complementary strengths to improve accuracy for challenging classes.

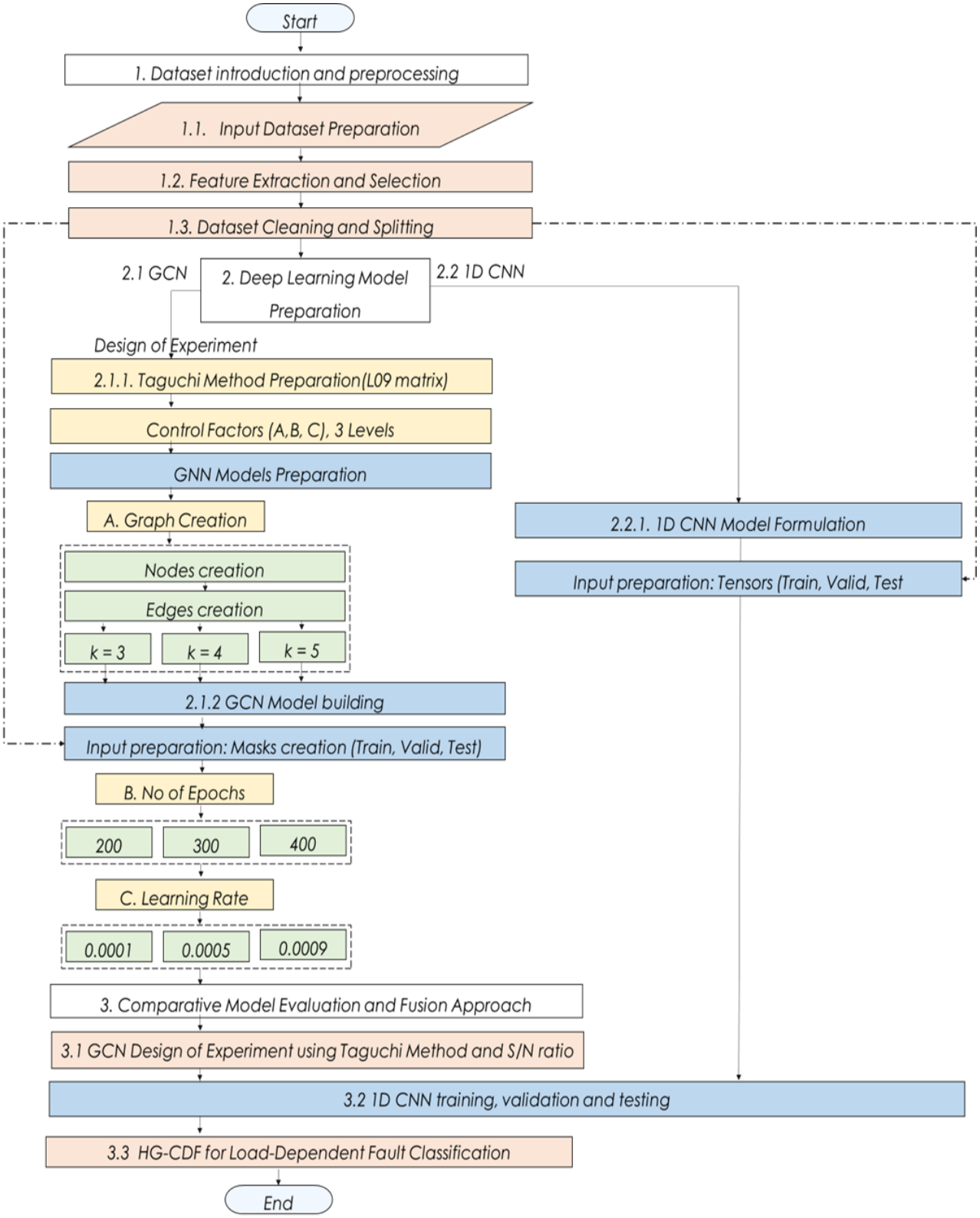


Figure 7.1: Hybrid Graph-CNN Decision Fusion (HG-CDF) for Load-Dependent Fault Classification.

7.1.2 Dataset

The chapter utilises the MFPT bearing dataset, previously described in Section 6.1.2, which includes Normal (fault-free) or Healthy condition, Outer Race Fault (ORF), and Inner Race Fault (IRF) samples recorded at different frequencies and Load Factors (LFs). The dataset serves as a benchmark, providing essential information such as radial LFs, shaft speed, and signal characteristics. A 5,000/97,566 split ratio is recommended to ensure consistent data segmentation and maintain reproducibility for accurate fault classification across the three channels.

7.2 Results and Discussion

7.2.1 Dataset Introduction and Preprocessing

7.2.1.1 Input Dataset Preparations

The MFPT bearing dataset is pre-processed using the CLAF methodology. The methodology was initiated by importing the MFPT bearing dataset into MATLAB 2023a and applying the CLAF. This pivotal step differentiated the dataset into new, load-dependent fault subclasses whilst extracting crucial TFD features, diverging from traditional fault classification approaches in Induction Motor (IM) bearings. By adhering to the CLAF's principles, introduced in Chapter 5, the data were systematically segmented based on LF conditions, setting the stage for analysis that is fine-tuned to load-specific patterns and establishing a departure from conventional classification methods.

7.2.1.2 Feature Extraction and Selection

Building on the CLAF, the top 20 features were selected based on their one-way Analysis of Variance (ANOVA) rankings, as detailed in Chapter 6 (see Table 6.4). These features include Mean, ShapeFactor, PeakValue, Root Mean Square (RMS), standard deviation (Std), ClearanceFactor, ImpulseFactor, Kurtosis, CrestFactor, PeakAmplitude, Band Power, and several PeakFrequency and PeakAmplitude measures. One-way ANOVA was instrumental in identifying the significance of these features, using p-values to rank their discriminative power, with lower values indicating greater relevance. These p-values were transformed into scores by taking the negative logarithm to further assess their importance for classification, culminating in a ranked list (see Table 7.1). Consequently, these top 20

features were utilised in the current research for graph representation, leveraging relational dynamics often overlooked by conventional analytical methods. Consequently, these features were captured for all data points corresponding to each severity class (see *APPENDIX 4)*.

Table 7.1: One-way ANOVA Ranking Including Spectral Features Extracted by Autoregressive (AR) Model; Order Fifteen, Peak = 5 (Top 20 Features).

Feature Rank	One-way ANOVA Score	Feature Rank	One-way ANOVA Score
1. Mean	316.44	11. BandPower	126.85
2. ShapeFactor	288.42	12. PeakFrequency 3	116.80
3. PeakValue	245.43	13. PeakAmplitude 5	84.33
4. RMS	240.93	14. Skewness	73.13
5. Std	240.27	15. PeakAmplitude 2	70.50
6. ClearanceFactor	235.23	16. PeakFreq1	69.14
7. ImpulseFactor	225.26	17. SINAD	58.72
8. Kurtosis	211.94	18. S/N	58.61
9. CrestFactor	198.26	19. PeakAmplitude 4	51.39
10. PeakAmplitude	161.22	20. PeakAmplitude 3	38.77

7.2.1.3 Dataset Cleaning and Splitting

The MFPT bearing dataset was processed using the CLAF methodology presented in Chapter 5. The classes were encoded as follows: Class 0 for 'Normal (fault-free) or Healthy condition', Class 1 for 'Mild', Class 2 for 'Moderate', and Class 3 for 'Severe,' representing 1,856 data points with balanced class distribution (see Table 7.2).

Table 7.2: Dataset Information.

Total Dataset Description		Dataset Splitting Count Per Class
Class distribution:		Training set class distribution: Counter ({1: 278, 2: 278, 0: 278, 3: 278})
Healthy: 0	464	
Mild: 1	464	Validation set class distribution: Counter ({1: 93, 2: 93, 0: 93, 3: 93})
Moderate: 2	464	
Severe : 3	464	Test set class distribution: Counter ({1: 93, 2: 93, 0: 93, 3: 93})

In the preprocessing phase of the current study, the dataset was cleansed of any missing values by substituting them with zeros, thereby ensuring data integrity and facilitating computational efficiency. Following this initial step, the categorical labels indicating health status ('Healthy,' 'Mild,' 'Moderate,' and 'Severe') located in the first column of the dataset were transformed from their categorical format into a numeric format using a

label encoding technique. This numeric conversion is crucial because it allows compatibility with Machine Learning (ML) algorithms requiring numerical inputs.

Subsequently, these numeric labels were converted into a one-hot encoded format to more effectively address the requirements of multi-class classification tasks utilised in the neural network models. Specifically, one-hot encoding transforms each numeric label into a binary vector representing all possible categories, with '1' indicating the presence of the category and '0' indicating the absence of the category. For example, the categories would be encoded as follows:

- 'Normal (fault-free) or Healthy condition': [1, 0, 0, 0]
- 'Mild': [0, 1, 0, 0]
- 'Moderate': [0, 0, 1, 0]
- 'Severe': [0, 0, 0, 1]

This encoding method ensures that each category is distinctly represented without any ordinal relationships implied, which is crucial for the unbiased functioning of the neural network in classifying these health statuses. Furthermore, the input features (X), excluding the label column, were isolated from the dataset and converted into a NumPy array. This transformation is essential because NumPy arrays are particularly well-suited to handle large data volumes efficiently, thus optimising the algorithm's performance during training. By ensuring that the features are in the appropriate format, the data are rendered ready for effective analysis and processing through advanced ML algorithms in subsequent stages.

7.2.2 Deep Learning Model Preparation

7.2.2.1 Graph Convolutional Network (GCN): Background and Formulation

This section discusses the steps of the GCN's formulation, beginning with the Taguchi method to propose the optimal GCN factors, followed by an overview of the GCN model's background and construction.

1) Taguchi Method Preparation (L09) Matrix

The Taguchi method was used here to propose the optimal GCN factors. Taguchi L09 was selected because there were three factors with three levels. The application of the Taguchi method in the current study provides an efficient strategy for optimising the performance of the GCN. Using the L09 orthogonal array, suitable

for evaluating three factors at three levels, the study effectively reduced the number of necessary experiments from 27 to 9 while still gathering comprehensive data regarding each factor's impact. The choice of factors and their specific levels in applying the Taguchi method for this study was driven by the objective of optimising the GCN for fault classification. The factors under consideration include the following:

1. **Factor A: Graph Creation:**
    - Level 1: k-Nearest Neighbour Graph (k-NNG) with k =3
    - Level 2: k-NNG with k =4
    - Level 3: k-NNG with k =5

The kNN algorithm classifies a new data point based on the labels of its closest neighbours. The number of neighbours, represented by "*k*," determines how many nearby points influence the classification. For example, if k =3, the new point is classified by considering the three nearest neighbours within a defined proximity. Increasing *k* smooths the decision boundary, making the model more robust but less sensitive to local variations. When *k* is very high, the classification might lean towards the majority class, potentially overlooking more minor yet significant patterns (Naman et al., 2020).

The selection of *k* in constructing a k-NNG is critical, impacting the graph's accuracy, efficiency, recall, and scalability. Optimising *k* requires a balance of these factors, tailored to the application's specific needs and computational resources. A higher *k* value enhances accuracy by considering more neighbours, better capturing the local data structure but increasing computational complexity and memory usage. Conversely, a lower *k* value speeds up computation but may overlook critical data connections, losing detail in the graph structure. Therefore, choosing *k* involves a trade-off between accuracy and computational speed, with higher values improving graph quality at the expense of efficiency and lower values boosting speed at the cost of accuracy (Dong et al., 2011; Yingfan et al., 2021).

Thus, the kNN algorithm is fundamental in graph-based learning because it defines the graph structure by connecting each node with its closest points. The chosen levels represent a range that allows the model to capture varying degrees of locality

and globality in the data. Thus, k =3 represents a tighter, more local neighbourhood which could capture fine-grained patterns, whereas k =5 expands the scope to more global structures, potentially capturing broader trends in the data. k =4 offers a mid-point that balances local detail with global context.

GCNs have been recognised for their ability to extract insight from data structured in graph form. The current study extends its application to tabular datasets in fault classification. A methodological transformation is proposed whereby traditional row-and-column data are envisioned as interconnected nodes. This section elaborates on the methodology employed to convert tabular data into graphs. Edges are drawn between nodes to represent relationships, with the kNN algorithm being utilised, setting k to 3, 4, and 5 to define these connections.

The following subsections outline the step-by-step process, starting with feature representation and concluding with graph construction. The use of the kNN function from scikit-learn to establish edges based on the chosen settings is detailed. This transformation encodes the latent relational information within the dataset into a graph structure, facilitating analysis by the GCN. Each kNN configuration is explored to reveal different connectivity patterns within the data, which is crucial for capturing the complex, nonlinear relationships characteristic of fault patterns.

Hence, this section delineates the conversion of tabular datasets into graph structures, an integral process for applying the GCN. Each instance in the dataset is treated as a node, with features as node attributes.

- Graph Construction

  Graphs are constructed by defining edges that represent relationships between instances. The kneighbours_graph from the scikit-learn library was employed to determine these connections based on the kNN algorithm, which is executed as follows:

  - Distance Calculation: Distances between instances are computed using the Euclidean metric.
  - Neighbour Selection: The 'k' nearest instances are identified for each node.

- Edge Creation: Edges are created between each node and its '*k*' nearest neighbours, resulting in a sparse adjacency matrix representation.

- Sparse Matrix Representation

The sparse matrix efficiently encodes the graph, mainly when the number of actual edges is much lower than the total possible number of edges, as is common in k-NNGs. The output is typically a sparse adjacency matrix, which efficiently represents the graph. This is beneficial when the graph is large, but the number of edges is relatively small compared to the number of possible edges (as is the case in a k-NNG). This matrix is then converted into a format (e.g., COO format) suitable for constructing a data object in PyTorch Geometric or for visualisation with libraries like NetworkX.

In PyTorch Geometric, the graph is encapsulated in a data object containing the following:

- x: Node feature matrix.
- edge_index: Graph connectivity in COO format.
- y: Labels for nodes (if applicable).
- Graph Visualisation

For visualisation, the NetworkX library is employed, translating the sparse matrix into a graphical format, which helps in understanding the graph's structure and node interconnectivity.

The lines or edges connecting the circles represent the 'nearest neighbour' relationships between records. An edge is drawn between two nodes if one of the nodes is among the '*k*' nearest neighbours of the other based on their features. In this graph, settings of v=3 were selected, meaning that each node is connected to its three closest neighbours. These edges help to highlight the local structure of the data because nodes with numerous shared connections are likely to belong to the same or similar classes.

This graph visualisation represents a dataset transformed into a k-NNG. Figure 7.2 illustrates the graph visualisation when v=3, Figure 7.3 when k =4, and Figure 7.4 when k =5. In these graphs, nodes correspond to individual records, and edges reflect their proximity based on feature similarities. Each

node is colour-coded to indicate its class, creating a visual distribution of the various categories: 'Normal (fault-free) or Healthy condition' (blue), 'Mild' (green), 'Moderate' (purple), and 'Severe' (red).

The distribution of colours throughout the graph suggests that certain classes tend to cluster together, as seen with several densely connected nodes of the same colour. This indicates that records within the same class share more substantial similarities than records of different classes. Notably, there are regions where different classes intermingle, such as the interface between the ‘Mild’ and ‘Moderate’ clusters, thereby implying that these classes share overlapping characteristics that are less distinguishable.

Furthermore, the visualisation highlights that 'Normal (fault-free) or Healthy condition' instances are relatively well-separated from other classes, indicating distinct features that can be leveraged for classification tasks. However, ‘Severe’ cases appear less numerous and somewhat interspersed within clusters of different classes, suggesting a more challenging classification scenario for these instances.

This graph provides valuable insights into the data structure, revealing patterns and relationships that can inform the development and refinement of predictive models. Understanding these visual cues is critical, especially in complex datasets that aim to discern intricate patterns that can improve model accuracy and interpretability.

The nodes (circles) and edges (lines) represent the data, reflecting the feature-based similarities among individual records and the broader structure of the dataset’s classes. The visualisation thus serves as a tool for exploring and understanding complex relationships within the data, which can be particularly useful for tasks such as classification, clustering, and anomaly detection.

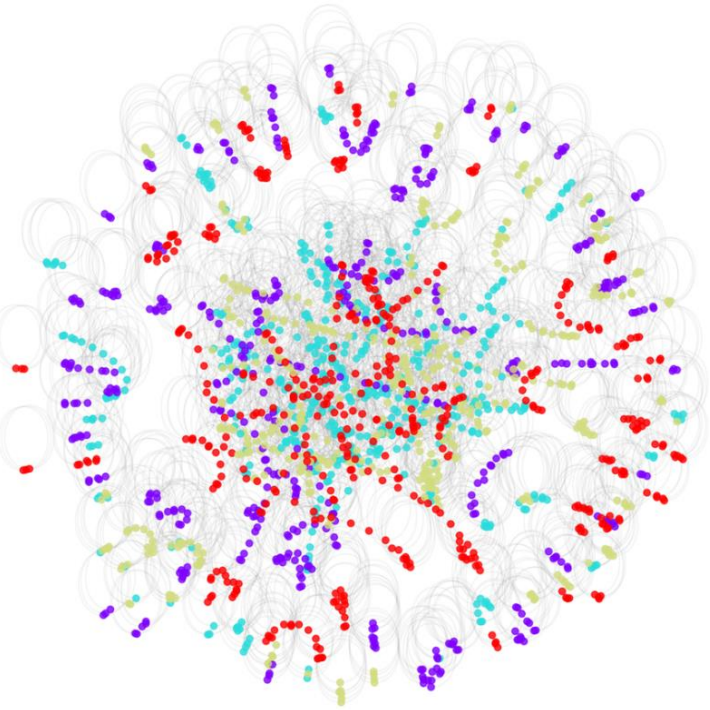


Figure 7.2: Graph Visualisation (k = 3).

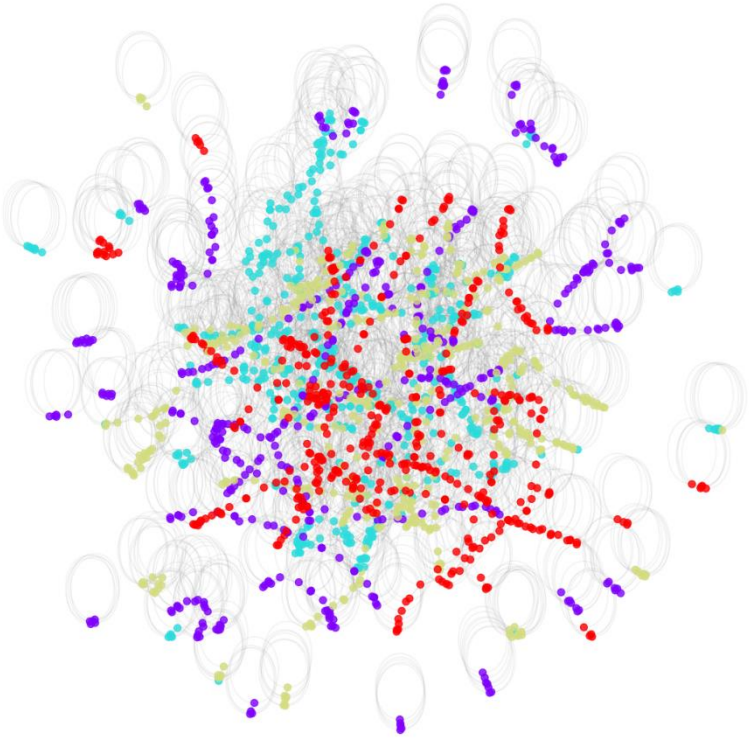


Figure 7.3: Graph Visualisation (k = 4).

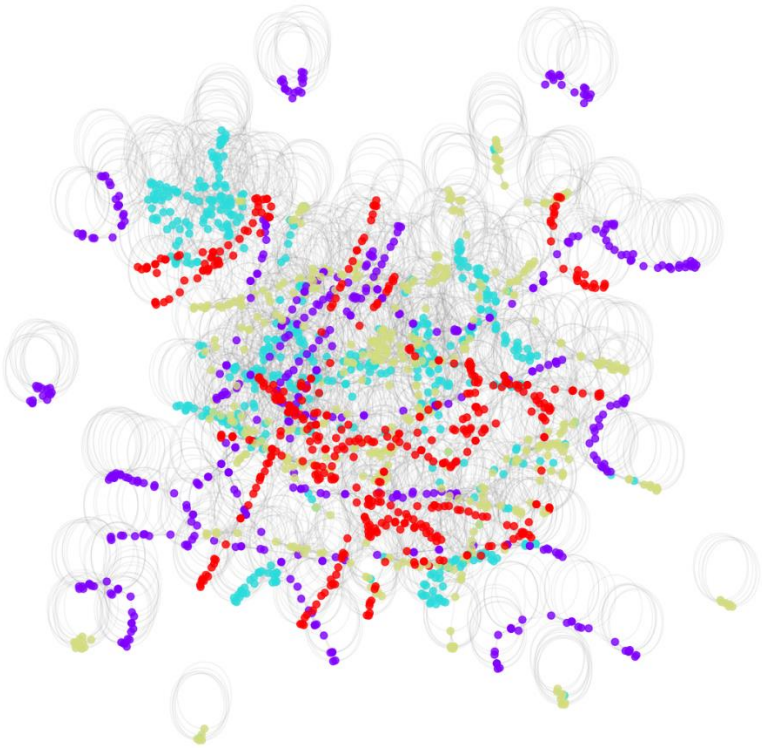


Figure 7.4: Graph Visualisation (k = 5).

2. **Factor B: Number of Epochs:**
    - Level 1: 200
    - Level 2: 300
    - Level 3: 400

    The number of epochs determines how often the model is exposed to the training dataset. Two hundred epochs ensure sufficient training without overfitting for simpler models; 300 epochs cater to more complex models, which may require additional training; and 400 epochs test the threshold for diminishing returns on model performance.

3. **Factor C: Learning Rate:**
    - Level 1: 0.0001
    - Level 2: 0.0005
    - Level3: 0.0009

    The Learning Rate (LR) controls the step size in the optimisation algorithm, affecting the convergence speed and stability. The learning rates 0.0001, 0.0005, and 0.0009 were chosen with a linear increase of 0.0004 to allow controlled

exploration. Small learning rates ensure stable convergence in complex models like Graph Neural Networks (GNNs), preventing instability and overfitting by updating weights more cautiously.

2) GCN Model Building

A GCN is a neural network designed for graph-structured data, extending the convolution concept from traditional Convolutional Neural Networks (CNNs) to graphs. GCNs are ideal for analysing data represented as graphs, such as social networks, citation networks, and molecular structures (Wei et al., 2020). The typical inputs for a GCN include the following:

- Adjacency Matrix (A): This matrix captures the graph's structure by detailing connections between nodes, allowing the GCN to learn from the graph's architecture (Niu et al., 2021; Yang, 2024)
- Node Feature Matrix (X): This matrix contains feature representations for each node, such as attributes or embeddings, enabling the GCN to learn about individual node characteristics (Yang, 2024).

Additional inputs may include edge features (e.g., edge weights) and node-level labels for tasks like node classification (Taslimipoor et al., 2019; Wang et al., 2023a).

The core components of a GCN include the following:

- Graph Convolution Layer: This layer performs spectral convolution operations on the graph, aggregating features from a node’s neighbours to capture structural details (Niu et al., 2021).
- Multilayer Structure: GCNs typically consist of multiple graph convolution layers, allowing the network to learn local and global graph structures (Zhang et al., 2021b).
- Nonlinear Activation: Functions, like Rectified Linear Unit (ReLU), follow each convolution layer, introducing nonlinearity to help the network learn complex patterns (Zhang et al., 2021b).
- Input and Output: A GCN typically takes an adjacency matrix, representing the graph’s structure, and a node feature matrix detailing information on each node. Outputs can include node-level predictions like classification, edge-level predictions such as link prediction, or graph-level predictions like graph classification (Wang et al., 2023a).

This chapter defines a GNN model with two GCN convolutional layers for graph data and trained using 5-fold cross-validation. A brief overview of the steps follows:

1. **Initialising Masks:** Boolean masks are created for the training, validation, and test sets. These masks indicate which indices belong to each split. Masks are typically Boolean arrays (or tensors) where each element corresponds to a node in the dataset:
   - True (1): If the element is True, the corresponding node is included in the operation (like training or evaluation).
   - False (0): If the element is False, the corresponding node is excluded from the operation.

   For example, if the dataset has 820 nodes divided into 60.00% training, 20.00% validation, and 20.00% testing, the training mask would be an array with indices of training elements set to True and the rest to False. The test mask and validation mask would be similar.
2. **Data Masking:** These masks are attached to the data object and will help select the correct data subsets during training, validation, and testing.
3. **GCN Model Definition:** A simple GCN model with two GCNConv layers is defined. The model takes the number of input features and the number of output classes as arguments. The architecture of the GCN is designed to capture the complex relationships between nodes in the graph by leveraging adjacency information. The model consists of the following layers:
   1. Input Layer:

      The input to the GCN model consists of node features and edge indices, represented as data.x and data.edge_index, respectively. These inputs encode the graph structure and node attributes essential for learning node representations. In the described methodology for applying GCNs to tabular datasets for fault classification, the input type used primarily includes the following components:
   2. Feature Matrix (X):

- The top 20 features were selected through the CLAF based on their one-way ANOVA rankings. These features include various statistical and frequency domain features such as the Mean, ShapeFactor, PeakValue, RMS, standard deviation (Std), ClearanceFactor, ImpulseFactor, Kurtosis, CrestFactor, PeakAmplitude, Band Power, and several PeakFrequency and PeakAmplitude measures.
- This matrix encapsulates the essential attributes of each node (data instance) in the graph, which are critical for the model to learn the patterns associated with fault classification.

3. Graph Structure: The adjacency matrix, crucial for a GCN, is created using the kNN algorithm to define the edges between nodes based on their proximity in the feature space. Nodes are connected if they are among the 'k' nearest neighbours of each other, and this matrix is derived from the distances between these nodes. This structure is essential for a GCN because it effectively aggregates and learns from the neighbourhood features. The following section provides information about the GCN structure:

   3.1 Graph Convolution Layers: The self.conv1 = GCNConv(num_features, 16) and self.conv2 = GCNConv(16, num_classes) in the code are graph convolution layers. These layers perform the core function of aggregating information from neighbouring nodes, applying the graph convolution operation to the input data. Each layer processes the node features, with conv1 transforming the input features to an intermediate dimension (16) and conv2 transforming these intermediate features to the final output size corresponding to the number of classes.

   3.2 Activation Function: F.relu(self.conv1(x, edge_index)) applies the ReLU activation function after the first convolutional layer, introducing non-linearity to the network. This is crucial to help the network learn complex patterns.

3.3 Normalisation and Regularisation: While specific normalisation such as L2 or Batch Normalisation (BN) is not applied here, dropout (F.dropout(x, training=self.training)) is used to prevent overfitting. This is especially useful in deep learning (DL) models like GCNs, where the model might learn too well from the training data at the expense of generalising it to new data.

3.4 Aggregation Function: The aggregation function is implicitly handled by the GCNConv layer itself, which defines how the features of the node and its neighbours are aggregated. While the aggregation function is not explicitly customised in this snippet, GCNConv typically uses a mean aggregator to combine node features based on the graph structure.

3.5 Output Layer: Return F.log_softmax(x, dim=1) serves as the output layer where the log softmax function is applied to the outputs from the last convolutional layer (conv2). This step converts the raw outputs to log probabilities essential for classification tasks, ensuring that the outputs are normalised and interpretable as probabilities. The final output of the model is obtained by applying a softmax function to the output features of the second graph convolutional layer. This step converts the features into log probabilities for each class, which are used for classification tasks.

4. **Node Labels (Y) (for supervised learning):** Each node is associated with a label indicating the class (types of faults).
5. **Model Training:** The model is trained and validated using 5-fold cross-validation under different Taguchi experiment configurations and the Adaptive Moment Estimation (Adam) optimiser. Appropriate data subsets are used for training, validation, and testing.

7.2.2.2 One-Dimensional Convolutional Neural Network (1D-CNN) Model: Background and Formulation

1D-CNNs are a specific type of neural network crafted to handle one-dimensional (1D) data, such as time series, signals, or sequential data. They are especially effective for applications such as fault detection, structural health monitoring, and various other pattern recognition tasks that involve sequential or time-series data (Camacho-Bello et al., 2022; Zhang et al., 2023c; Ahmadzadeh et al., 2024). 1D-CNN is a specialised neural network for handling 1D data such as time series, speech, or vibration signals. This architecture is a variation of the Two-Dimensional Convolutional Neural Network (2D-CNN), frequently used in image recognition tasks. Like its two-dimensional (2D) counterpart, the 1D-CNN architecture comprises 1D, activation, pooling, and fully connected layers. The convolutional layer utilises a set of learnable filters to process the input data and extract features. An activation layer follows, applying a nonlinear function to introduce non-linearity to the model. The pooling layer then reduces the data's spatial dimensions, helping to decrease the model's complexity and prevent overfitting. The fully connected layer finally projects the processed features into the desired output space (Qazi et al., 2022).

The 1D-CNN has found applications in various fields, including network intrusion detection, signal processing, and speech recognition. Specifically, the 1D-CNN effectively extracts features from time-series data and classifies signals in Vibration Signal Analysis (VSA). For instance, it can detect anomalies in vibration signals, potentially indicating faults in machinery. It is suitable for processing data such as sequences or signals and is relatively easy to train (Huang and Li, 2021).

Applying 1D-CNNs in fault detection is a prominent area of research in various fields. Researchers have utilised 1D-CNNs for fault diagnosis and detection in different domains, leveraging the unique capabilities of 1D-CNNs for processing time-series data efficiently. The following are some of the 1D-CNN applications in fault detection:

1) Aircraft Engine Fault Diagnosis: Wang et al. proposed a novel method that combines features from multiple sensors using 1D-CNNs for predicting bearing faults in aircraft engines (Wu et al., 2024).

2) Damage Detection: Abdeljaber et al. adopted a 1D-CNN for damage detection, showcasing its effectiveness in identifying structural damage (Chen et al., 2022b).
3) Unbalanced Data Fault Diagnosis: Researchers have developed new fault diagnosis methods for unbalanced data, utilising improved 1D-CNNs and the L2-Support Vector Machine (L2-SVM). The L2-SVM specifically uses the L2-norm of the error vector for regularisation in the loss function, effectively addressing the challenges posed by imbalanced datasets in fault detection (Hu et al., 2022).
4) Rolling Bearing Fault Diagnosis: The use of a 1D-CNN with demodulated frequency features has been explored for fault diagnosis of rolling bearings under time-varying speed conditions, highlighting the versatility of 1D-CNNs in detecting faults in mechanical systems (Lu et al., 2022).
5) Bearing Fault Diagnosis: An end-to-end intelligent fault diagnosis method for bearings combining 1D-CNN with Long Short-Term Memory (LSTM) networks has been proposed, demonstrating the effectiveness of 1D-CNNs in diagnosing bearing faults (Sun and Zhao, 2021).

These applications underscore the significance of 1D-CNNs in fault detection across various industries, showcasing its ability to process time-series data effectively and extract meaningful features for fault diagnosis and maintenance assurance.

- **1D-CNN Model Architecture Formulation**

The comparison between the established 1D-CNNs and the proposed GNNs in the current study is pivotal for benchmarking the innovative graph approach against a recognised standard in sequence data analysis. This comparison highlights the GCN's enhanced capability for relational data processing. It comprehensively evaluates its performance in handling tabular data for fault classification, a domain traditionally dominated by CNNs.

The Simple 1D-CNN is a custom CNN that processes 1D input data. It is implemented using PyTorch's neural network module, nn.Module. The architecture of this model is specifically tailored to applications where the input features are 1D, making it suitable for time series, sequence data or flattened representations of tabular data.

- **Key Components:**

Convolutional Layers: The model consists of two 1D convolutional layers. The first convolutional layer (conv1) has a single input channel and 16 output channels with a

kernel size of 3, a stride of 1, and a padding of 1. The second convolutional layer (conv2) increases the depth from 16 to 32 output channels with the same kernel size, stride, and padding. These layers extract hierarchical features from the input data (see Table 7.3).

Table 7.3: 1D-CNN Model Summary.

Layer Type	Output Shape	Param #
Conv1d	16, L	64
MaxPool1d	Variable	0
Conv1d	32, L	1,568
Flatten	Variable	0
Linear	120	96,120
Linear	4	484

Total Params: 98,236

- Pooling Layer: A max pooling layer (pool) with a kernel size of 2 and a stride of 2 follows each convolutional layer. Pooling layers reduce the dimensionality of the data by taking the maximum value over the window defined by the kernel size, helping to make the representation smaller and more manageable.
- Flattening: After the convolution and pooling layers series, a flattening operation (flatten) is applied to transform the multi-dimensional output into a 1D vector. This step is necessary to transition from convolutional layers to fully connected layers.
- Fully Connected Layers: The network transitions to dense layers by introducing two fully connected layers (fc1 and fc2). The first fully connected layer dynamically calculates its input size based on the output from the preceding pooling layer and connects to 120 units. The second fully connected layer (fc2) maps these 120 units to the number of classes (num_classes) in the dataset, serving as the output layer of the model.

- **Dynamic Initialisation:**

An innovative aspect of this model is its dynamic calculation of the input size for the first fully connected layer (fc1). This feature is implemented in the _init_fc1 method, which simulates a forward pass through the convolutional and pooling layers with a dummy input to determine the correct input size for fc1. This approach ensures that the model can adapt to different sizes of input features without manual adjustment.

- **Forward Pass:**

The forward method defines the data flow through the network. Input data are first unsqueezed to add a channel dimension, then sequentially passed through the convolutional, pooling, and fully connected layers. Activation functions (ReLU) are applied after each convolutional and the first fully connected layer to introduce non-linearities into the model, thereby enabling it to learn complex patterns in the data.

- **Training Preparation:**

The model is moved to the appropriate device for training in Google Colab. The number of input features (num_features) and classes (num_classes) are specified based on the dataset's characteristics. A custom function reset_weights is also defined to reinitialise the model's weights, ensuring that each training session or cross-validation fold starts with a fresh model. Both models (GCN and CNN) are trained using 5 cross-validation to ensure the robustness of the results.

- Input Preparation (Tabular Dataset Format as Tensors):
    - Regular Grid or Sequence: CNNs, especially 1D-CNNs designed for tabular data, expect data that can be interpreted as a regular grid or sequence. In the case of tabular data, each row (data point) can be seen as a sequence of features.
    - No Explicit Graph Structure: Unlike GCNs, CNNs for tabular data do not use an edge index or any graph connectivity information. They treat each dataset row independently, assuming that any relationships between features are captured through convolutional processing.
    - Batch Processing: While both models can use batch processing, how batches are prepared and fed into the model may differ. CNNs do not require masks to separate training, validation, and testing data within a graph structure. Instead, the dataset is split into separate tensors, or a Data Loader is used to manage batches and splits.

7.2.3 Comparative Model Evaluation and Fusion Approach

7.2.3.1 Graph Convolutional Network (GCN) Design of Experiment using Taguchi Method and Signal-to-Noise (S/N) Ratio

The Taguchi method, developed by Genichi Taguchi, aims to enhance the quality of products or processes by increasing their robustness to external variations. This is achieved through orthogonal arrays, which allow for the efficient analysis of multiple variables with relatively few experiments. Both methods improve decision-making and process optimisation across manufacturing, marketing, and quality control (de Oliveira et al., 2023). This section will use the Taguchi method to design this chapter's experiments.

The Taguchi method and the Signal-to-Noise (S/N) ratio are pivotal concepts in quality engineering and the DOE. Developed in the 1940s and 1950s by Genichi Taguchi, a prominent Japanese engineer, the Taguchi method streamlines product and process design optimisation. It focuses on pinpointing essential factors that influence quality and mitigates the effects of factors beyond control. This method employs orthogonal arrays for experimental design, facilitating the efficient examination of multiple variables at various levels (Pal and Gauri, 2017; Rathore, 2017).

The S/N ratio, integral to the Taguchi method, quantifies the robustness of a design by comparing the desired output (signal) to the variability caused by uncontrollable factors (noise). A higher S/N ratio denotes a design that effectively withstands variability, enhancing its robustness. In the context of alternative analysis, the S/N ratio helps assess the efficacy of various design alternatives or configurations. By evaluating the S/N ratio, engineers can discern the most impactful factors on performance and refine the design to optimise outcomes. The general formula for the S/N ratio in the Taguchi method is shown in Equation (7.1) (Bisht et al., 2013):

$$\mathrm{S/N} = -10 \log \left(Mean\ Squared\ Deviation(MSD)\right) \quad \textbf{(7.1)}$$

The Taguchi method recognises three types of S/N ratios:

1. Smaller-the-better: This is used when the objective is to minimise a response variable, aiming to reduce process variability, for example. The S/N ratio increases as observed values decrease. It is calculated using the formula for

MSD: $MSD = \sum(y^2)/n$ where y represents the observed data and n is the number of observations (Yang et al., 2023c).

2. Nominal-the-best: This targets a specific value, such as achieving a precise dimension. The formula applied is $MSD = \sum(y - m)^2/n$, where mm is the target value, and n is the number of observations. Higher S/N ratios correspond to values closer to the target, facilitating the optimisation towards desired outcomes (Bisht et al., 2013; Yang et al., 2023c).
3. Larger-the-better: The goal here is to maximise response, such as enhancing the strength of a material. The mean square deviation is calculated as $MSD = \sum\left(\frac{1}{y^2}\right)/n$, with larger observed values yielding higher S/N ratios (Yang et al., 2023c).

The Taguchi method is a systematic approach to optimise products or processes by minimising output variation and is applicable across various systems, including ML algorithms. In one instance, the Taguchi method optimises ML parameters for predicting turning precision and identifying key influential factors. This methodology trains three models, enhancing the best-performing model, XGBoost, with techniques like Synthetic Minority Over-sampling Technique for Regression with Gaussian Noise (SMOGN) and various optimisation algorithms, such as Centre Particle Swarm Optimisation (CPSO) (Wang et al., 2022).

In another example, the Taguchi method is employed to refine hyperparameters in a Back Propagation Neural Network (BPNN) to predict plastic injection moulding outcomes. It identifies critical hyperparameters and optimises them using the Taguchi orthogonal approach to enhance prediction accuracy (Jong et al., 2020). Furthermore, the Taguchi method optimises rotor barrier designs in PM-assisted Synchronous Reluctance Machines (SynRMs). It identifies key factors affecting Power Factor (PF) and torque ripple, using these insights to refine rotor barrier designs through a Taguchi experimental approach (Naseh et al., 2022).

In the evaluation phase, a 10-fold cross-validation strategy was employed to ensure the robustness of the models. For each fold, the model was trained on nine parts of the data and validated on the remaining part. This process was repeated ten times, and each part was

used once for validation. The test dataset, which was held constant across all folds, was used to evaluate the model post-training, as shown in Table 7.1.

The S/N, in the context of the Taguchi method, is a metric used to measure the quality characteristic of a process. The S/N combines both the mean and the variability into a single statistic. When maximising the performance characteristic (the Overall Test Accuracy), the larger-the-better type of S/N is used, as indicated in the last column of Table 7.1. The Taguchi Approach Summary: S/N by Factor Levels is presented in Table 7.2. The formula for the larger-the-better S/N is given in Equation (7.2):

$$\text{S/N} = -10 \log_{10} \left(\frac{1}{n} \sum_{i=1}^{n} \frac{1}{y_i^2} \right) \quad \textbf{(7.2)}$$

where n is the number of observations (or trials), y_i is the value of the performance characteristic for the i-th observation, and $\log_{10}$ represents the logarithm to base 10 (also known as the common logarithm).

In this chapter's Taguchi experiment, as shown in Table 7.4, each experiment is conducted only once ($n = 1$). Therefore, the S/N formula is simplified as shown in Equation (7.3):

$$\text{S/N} = -10 \log_{10} \left(\frac{1}{y^2} \right) \quad \textbf{(7.3)}$$

The negative sign is used because higher S/Ns are typically considered to be better, and a larger y (which is better for larger-the-better quality characteristics) will yield a smaller one, that is, $\log_{10}(\frac{1}{y^2})$ (hence the need to take the negative to make the S/N larger for better outcomes). These S/N values are in negative decibels (dB) because the reciprocal of a number less than 1 (when converted from a percentage) is greater than 1, and taking a log of a number greater than one yields a positive number. Because the S/N formula includes a negative sign, this results in negative values for the S/N.

Each S/N value is calculated for the overall test accuracy of each experiment, and the mean S/N is then computed for each level of each factor. The level with the highest mean S/N is considered the most robust setting for that factor because it offers the highest quality with the slightest variation, as shown in Table 7.4. In this study, the overall accuracy of the GCN was assessed across nine experiments using the Taguchi design, emphasising the importance of fine-tuning model parameters for optimal performance and reducing the number of potential combinations. The mean accuracy was approximately 89.01%, with a

standard deviation of ±1.25%, reflecting the variability observed across the experiments, where each experiment takes around 28 s.

Table 7.4: Taguchi-Derived GCN Model Performance Evaluation.

No.	Control Factors			VA[1]	Testing					S/N Ratio
	A. Graph	B.Epochs	C.LR		MA[2]	MoA[3]	SA[4]	NA[5]	Overall Accuracy	on Overall Test Accuracy
1	GCN with k =3	200	0.0001	89.80%	60.22%	98.92%	97.85%	98.92%	88.98%	-1.01
2	GCN with k =3	300	0.0005	90.02%	62.37%	97.85%	100%	100%	90.06%	-0.91
3	GCN with k =3	400	0.0009	89.42%	59.14%	98.92%	98.92%	98.92%	88.98%	-1.02
4	GCN with k =4	200	0.0005	88.04%	59.14%	97.85%	95.70%	100%	88.17%	-1.09
5	GCN with k =4	300	0.0009	89.58%	61.29%	96.77%	98.92%	100%	89.25%	-0.99
6	GCN with k =4	400	0.0001	92.28%	66.67%	98.92%	100%	100%	91.40%	-0.78
7	GCN with k =5	200	0.0009	88.03%	59.14%	93.55%	91.40%	100%	86.02%	-1.31
8	GCN with k =5	300	0.0001	90.10%	62.37%	95.70%	98.92%	100%	89.25%	-0.99
9	GCN with k =5	400	0.0005	90.31%	60.22%	96.77%	98.92%	100%	88.98%	-1.01

[1] VA is validation accuracy, [2] MA is the Mild state accuracy, [3] MoA is the Moderate state accuracy, [4] SA is the Severe state accuracy, [5] NA is the Normal (fault-free) or Healthy condition accuracy.

For factor level analysis, the higher (less negative) the S/N, the better the performance. Therefore the highest S/N value should be targeted. Experiment 6 scores the least negative S/N, indicating the most robust performance. Further insights about S/N ratios are summarised in Table 7.5:

Table 7.5: Taguchi Approach Summary: Signal-to-Noise Ratio by Factor Levels.

Control Factor	Control Factor Levels	Avg S/N for each factor
A	GCN with k =3	-0.98
	GCN with k =4	-0.95
	GCN with k =5	-1.10
B	200	-1.14
	300	-0.96
	400	-0.94
C	0.0001	-0.93
	0.0005	-1.01
	0.0009	-1.10

To enhance the clarity of the analysis, each factor was represented in a separate figure along with its corresponding levels. Factor A, involving Graph Creation using the k-NNG with three levels of the k factor, is shown in Figure 7.5. Factor B, representing Epochs, is depicted in Figure 7.6, while Factor C, representing the Learning Rate (LR), is illustrated in Figure 7.7.

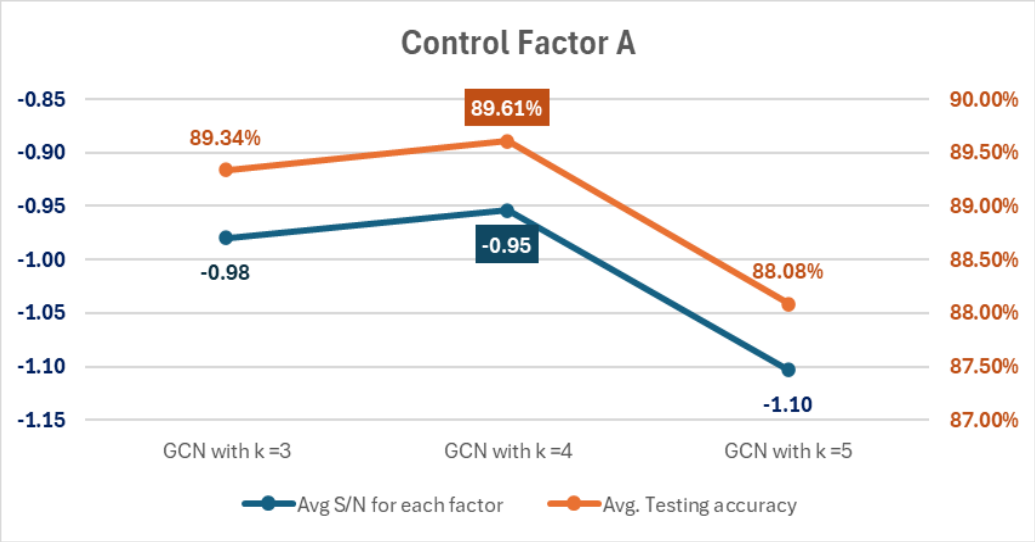


Figure 7.5: Levels of Control Factor A, S/N Ratio, and Test Accuracy.

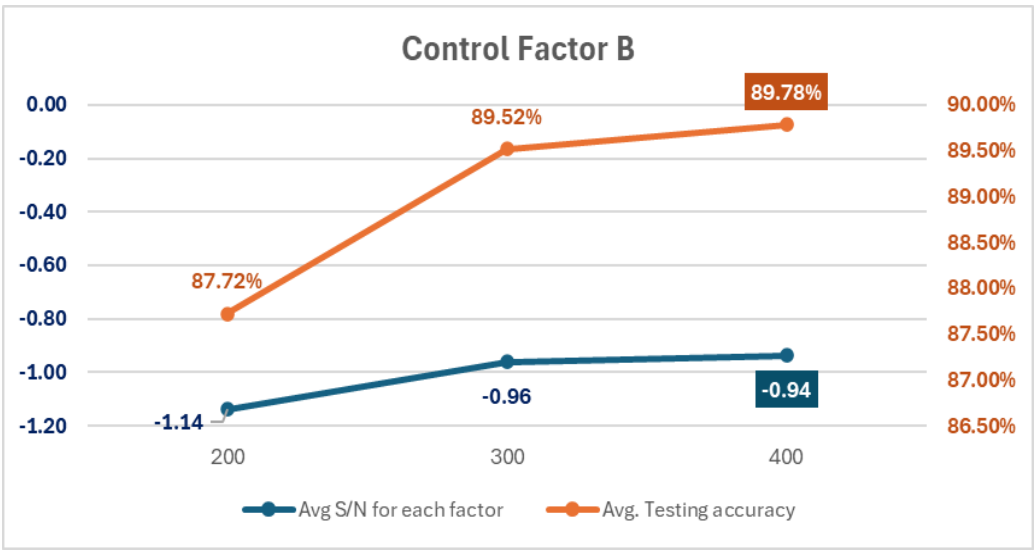


Figure 7.6: Levels of Control Factor B, S/N Ratio, and Test Accuracy.

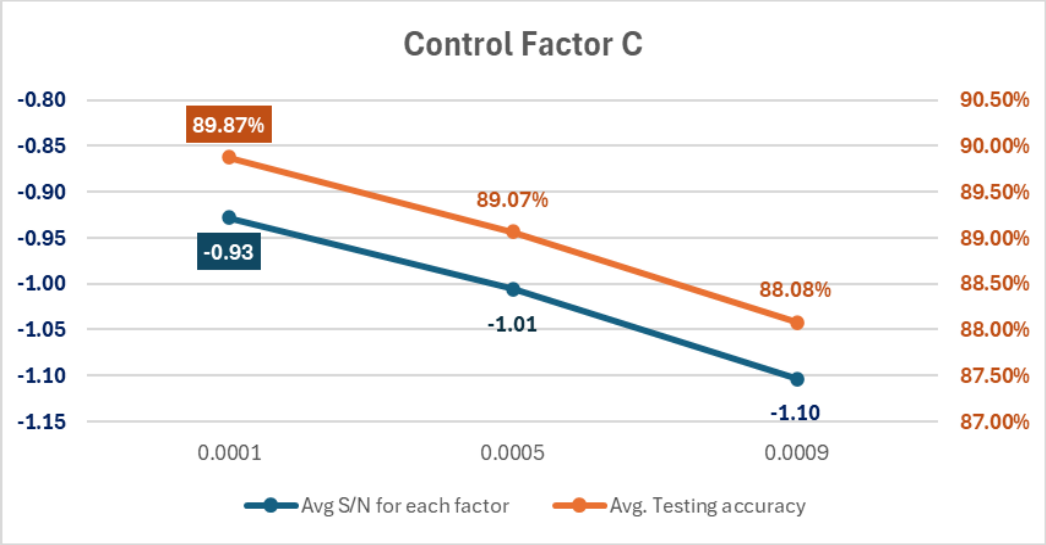


Figure 7.7: Levels of Control Factor C, S/N Ratio, and Test Accuracy.

From the S/N ratio analysis, it can be observed that the negative values of S/N may appear counterintuitive, but they result from the calculation method whereby performance characteristics are converted into a noise metric;the larger the S/N ratio the smaller the variation and therefore larger S/N results are selected.Experiment 6, which corresponds to a GCN with k =4, 400 epochs, and aLR of 0.0001, resulted in the highest overall accuracy and the largestS/N, thereby suggesting that it is the best performer among all of the tested experiments.

While the GCN, leveraging the k-NNG approach, presents a novel and powerful method for detecting faults in complex systems, it displayed a notable limitation in the Mild fault class. Despite excelling in the Moderate (98.92%), Severe (100%), and Normal (fault-free) or Healthy condition (100%) classes, its accuracy in the Mild class was significantly lower at 66.67%. This discrepancy can be attributed to the sparse distribution of Mild class samples within the graph structure, as depicted in the graph visualisation (Figure 7.3). The Mild fault samples were scattered and less connected, making it difficult for the GCN to exploit neighbourhood signals fully. Consequently, this led to misclassification, mainly due to an overlap with the Moderate and Normal (fault-free) or Healthy condition classes, limiting the GCN's ability to distinguish between them accurately. This observation underscores the need for further refinement and opens the path for improvement through the proposed hybrid approach.

7.2.3.2 Proposed GCN using Taguchi with Selective Weighted Loss (SWL) Method for Refining Mild Class Performance

To refine the GCN model, particularly in the Mild class, after selecting the optimal Taguchi parameters, a weighted loss function approach was implemented to address the observed 33.33% misclassification rate in the Mild class. While the dataset was balanced, the lower accuracy for the Mild class could be attributed to its representation within the k-NNG structure, which created challenges for the GCN in correctly classifying this class. To mitigate this, a custom weighting scheme was applied based on the optimal Taguchi factors from Experiment 6, which corresponds to a GCN with k =4, 400 epochs, and an LR of 0.0001 (Table 7.4). The loss function was adjusted during training by assigning a higher weight to the Mild class. Specifically, the weight for the Mild class was doubled relative to the other classes, with the following weight vector: [1.0, 2.0, 1.0, 1.0]. This adjustment was incorporated using the torch.nn.CrossEntropyLoss function in PyTorch, allowing the model to focus more on correctly classifying the Mild class without sacrificing performance in the other classes.

This adjustment aimed to improve the model's sensitivity to the Mild class, reducing its misclassification rate while maintaining overall accuracy across all classes. The training was repeated five times, and the results showed an improvement in Mild class accuracy, increasing from 66.67% to 84.52% ± 1.96%. Meanwhile, the overall testing accuracy remained almost the same, at 90.70% ± 0.15%, compared to the 91.40% achieved in Experiment 6 (Table 7.4), while maintaining the same training time of 28 s. However, while the model's performance in the Mild class improved, there was a slight compromise in its ability to classify the Moderate class accurately, with the accuracy dropping from 98.92% in Table 7.4 to 78.82% ± 2.33%. This supports the idea that the model was previously "confused" or was misallocating resources between these classes.

This demonstrates the importance of SWL in addressing class-specific performance gaps, mainly when certain classes, like the Mild class, are harder to classify due to sparse graph connections or overlapping features. The SWL method effectively reallocated the model’s focus, allowing it to improve its performance in underperforming classes.

Table 7.6: GCN using Taguchi with Selective Weighted Loss (SWL) Trials for Mild Class Performance Improvement.

Training Runs	Validation Accuracy	Mild Accuracy	Moderate Accuracy	Severe Accuracy	Normal Accuracy	Overall Test Accuracy
1	89.53%	84.95%	77.42%	100%	100%	90.59%
2	91.60%	81.72%	81.72%	100%	100%	90.86%
3	90.60%	83.87%	78.49%	100%	100%	90.59%
4	90.59%	84.95%	78.49%	100%	100%	90.86%
5	90.58%	87.10%	75.27%	100%	100%	90.59%
Avg.	90.58%	84.52%	78.28%	100%	100%	90.70%

7.2.3.3 One-Dimensional Convolutional Neural Network (1D-CNN) Compared with Proposed GCNs

1D-CNNs and GNNs differ in the types of data they handle, their feature extraction mechanisms, and their areas of strength. 1D-CNNs are well-suited for detecting sequential patterns in structured, grid-like data, such as time series, where local dependencies and patterns can be effectively captured through convolutional layers. In contrast, GNNs, which include models like GCNs, excel at capturing relationships in irregular, graph-structured data, where the connections between data points are crucial for learning complex interdependencies in tasks such as fault classification in systems with complex data structures.

This section explores the potential of 1D-CNNs in handling CLAF load-dependent fault subclasses. The training, validation, and test datasets used here are similar to those previously employed in the GCN model. The data were transformed into tensors to train the 1D-CNNs, using an LR of 0.005 over 300 epochs, as shown in Table 7.6. The table compares the proposed GCN using Taguchi, presented earlier in Table 7.4 (Experiment 6), with the 1D-CNN. Furthermore, it compares the proposed GCN with SWL (Table 7.7) to the 1D-CNN.

Table 7.7: 1D-CNN and Proposed GCN Configurations Performance Evaluation.

Training Model	Training Time	Validation Accuracy	Mild Accuracy	Moderate Accuracy	Severe Accuracy	Normal Accuracy	Overall Test Accuracy
1D-CNN	3. min = 180 sec	95.31%	97.85%	92.47%	100%	100%	97.58%
Proposed GCN using Taguchi	28 sec	92.28%	66.67%	98.92%	100%	100%	91.40%
Proposed GCN with SWL	28 sec	90.58%	84.52%	78.28%	100%	100%	90.70%

Comparison Summary from Table 7.7:

- 1D-CNN compared to proposed GCN using Taguchi

The 1D-CNN model performed exceptionally well in the Mild class, achieving an impressive accuracy of 97.85%. The 1D-CNN also showed a strong overall test performance, with an overall accuracy of 97.58%, which is higher than the proposed GCN's 91.40%, as shown in Table 7.4 (Experiment 6). However, the 1D-CNN faced challenges in classifying the Moderate class, achieving an accuracy of 92.47%, whereas the GCN excelled in this class with 98.92%.

- 1D-CNN compared to proposed GCN using Taguchi with SWL

When comparing the 1D-CNN and the proposed SWL method, it is clear that both approaches offer distinct advantages in handling the Mild class. The 1D-CNN achieved an impressive 97.85% accuracy for the Mild class, significantly outperforming the proposed SWL, which improved the Mild class accuracy to 84.52%. Additionally, the 1D-CNN maintained a high overall test accuracy of 97.58%, compared to 90.70% for the SWL method. However, SWL achieved these results with the same computational efficiency as the original GCN model (28 s), whereas the 1D-CNN required more training time (180 s). Despite the computational advantage of SWL, its moderate performance and the drop in accuracy for the Moderate class make it less suited for inclusion in the hybrid approach, which aims to balance class performance optimally.

7.2.3.4 Hybrid Graph-CNN Decision Fusion (HG-CDF) for Load-Dependent Fault Classification

Although the proposed GCN using Taguchi with SWL proved helpful in refining class-specific accuracy, the hybrid method is specifically designed to leverage the individual strengths of the GCN and the 1D-CNN without compromising performance in any class. In this setup, the GCN retains its strength in the Moderate class, while the 1D-CNN provides robustness in the Mild class, thereby avoiding the trade-offs introduced by SWL. The hybrid approach ensures better class performance across all fault types without needing selective loss weighting for balance.

The GCN demonstrated significant strengths in capturing complex relationships, particularly excelling in the Moderate (98.92% accuracy), Normal (fault-free) or Healthy condition (100% accuracy), and Severe (100% accuracy) fault classes. One key advantage of the GCN is its ability to handle complex and large datasets efficiently, with experiments completed in approximately 28 s. However, its performance in the Mild class was limited, achieving only 66.67% accuracy due to sparse node connectivity and overlap with other classes, as shown in the graph visualisation (Figure 7.3). Despite these limitations, the k-NNG approach employed by the GCN offers a robust method for fault classification in complex systems, mainly when modelling tabular vibration signals. To address the performance gap in the Mild class, the HG-CDF approach was proposed, combining the strengths of both the GCN and the 1D-CNN to ensure enhanced performance across all fault classes. The findings from the decision fusion approach are outlined in Table 7.8.

Table 7.8 shows an optimised weighting system can significantly improve classification performance, particularly in the Moderate class. Two systems were evaluated: Equal Weighting and the Adaptive Weighting System. The Equal Weighting system, assigning equal weights (0.5) to both the GCN and the 1D-CNN for all classes, achieved a solid overall test accuracy of 98.66%. However, the Moderate class performed slightly below expectations at 95.70%, despite the GCN's inherent strengths in this class. In terms of training time, the GCN took only 28 s to complete training, while the 1D-CNN required around 3 min.

When considering the total training time for the hybrid approach, the combined training time for both models amounts to approximately 3 min 28 s. This highlights the efficiency of the GCN in delivering high-accuracy results quickly and the role of the 1D-CNN in complementing areas where the GCN struggles, such as the Mild class.

In contrast, the Adaptive Weighting System, which gave a higher weight to the 1D-CNN for the Mild class (0.7) and to the GCN for the Moderate class (0.6), led to a marked improvement. The overall test accuracy rose to 99.19%, with the Moderate class improving from 95.70% to 97.85%. This result demonstrates that leveraging the unique capabilities of both models—the GCN for complex graph-structured data and the 1D-CNN for sequential patterns—can enhance classification performance. Both systems maintained perfect accuracy in the Severe and Normal (fault-free) or Healthy condition classes, highlighting their robustness for these fault types.

Table 7.8: Hybrid Graph-CNN Decision Fusion (HG-CDF) Weighting Systems and Performance Comparison.

Weighting system Alternatives	**Class**	**Proposed GCN using Taguchi**	**1D-CNN Weight**	**Testing Accuracy**	**Overall Test Accuracy**	**Notes**
1. Equal Weighting	Mild	0.5	0.5	98.92%	98.66%	Basic equal weighting for all classes.
	Moderate	0.5	0.5	95.70%		
	Severe	0.5	0.5	100%		
	Normal	0.5	0.5	100%		
2. Adaptive Weighting	Mild	0.3	0.7	98.92%	99.19%	Higher CNN weight for Mild, higher GCN weight for Moderate
	Moderate	0.6	0.4	97.85%		
	Severe	0.5	0.5	100%		
	Normal	0.5	0.5	100%		

7.2.4 Comparison of LD-MVSEF (Chapter 6) and HG-CDF (Chapter 7) in Mild and Moderate Class Fault Detection

Comparing the performance of the Load-Dependent Multimodal Vibration Signal Enhancement and Fusion (LD-MVSEF) method from Chapter 6 and the HG-CDF approach

from Chapter 7, we can see that both methods demonstrate strong classification capabilities for the Mild and Moderate fault subclasses, as presented in Table 7.9. Both models achieved 100% accuracy in the Normal and Severe fault classes. LD-MVSEF achieved a test accuracy of 97.20% for the Mild class and 99.15% for the Moderate class but required a training time of 18 min 30 s. In contrast, HG-CDF slightly outperformed LD-MVSEF in the Mild class with a test accuracy of 98.92%, though it achieved a slightly lower accuracy of 97.85% in the Moderate class. What sets HG-CDF apart is its efficiency, completing the training in just 3 min 28 s—considerably faster than LD-MVSEF. This comparison shows that while LD-MVSEF excels in Moderate class accuracy, HG-CDF offers a more efficient solution with competitive accuracy, particularly in the Mild class, suggesting its potential for faster, less complex condition monitoring.

Table 7.9: Accuracy Comparison of LD-MVSEF (Chapter 6) and HG-CDF (Chapter 7) Across Fault Subclasses.

Proposed Methodology	Training Time	Mild Accuracy	Moderate Accuracy	Severe Accuracy	Normal Accuracy	Overall Test Accuracy
LD-MVSEF (Chapter 6)	18 min 30 s	97.20%	99.15%	100%	100%	99.04%
HG-CDF (Chapter 7)	3 min 28 s	98.92%	97.85%	100%	100%	99.19%

7.3 Summary

This chapter presented a novel approach for load-dependent fault classification by applying GCNs to transform tabular vibration signal data into graph structures using k-NNG method. Through the Taguchi DOE, nine GCN configurations were tested, achieving a mean accuracy of 89.01% ± 1.25%, with a strong performance in the Moderate (98.92%), Severe (100%), and Normal (fault-free) or Healthy condition (100%) classes. However, the GCN encountered limitations in the Mild class, with accuracy reaching only 66.67% due to sparse node connectivity and class overlap.

To address this limitation, the chapter first introduced the Selective Weighted Loss (SWL) method as an attempt to improve the Mild class accuracy by adjusting the loss function to focus more on this underperforming class. While the SWL method successfully

raised the Mild class accuracy to 84.52% ± 1.96%, it resulted in a drop in the Moderate class accuracy, showing a trade-off between classes that limited its overall effectiveness.

Recognising the need for a more balanced solution, the HG-CDF method was proposed, integrating the GCN with a 1D-CNN. This hybrid approach leveraged the GCN's ability to model complex relationships and the 1D-CNN's strength in detecting sequential patterns. The HG-CDF method significantly improved performance, particularly in the Mild class, while maintaining perfect accuracy in the Severe and Normal (fault-free) or Healthy condition classes. Furthermore, the hybrid model demonstrated computational efficiency, completing training in just 3 min 28 s—combining the GCN's 28-second speed with the 1D-CNN's 3-minute training time, and achieving a high testing accuracy of 99.19%.

This chapter contributes to the field by validating the GCN's effectiveness for fault classification, exploring the use of SWL for class-specific improvement, and introducing a hybrid model that overcomes identified limitations, enhancing performance across all CLAF load-dependent fault subclasses. The main contributions of the current chapter are as follows

1. **Proposed GCN using Taguchi:** Explored and validated the use of GNNs within the CLAF, using TFD features, including spectral features extracted via Autoregression instead of raw vibration signals. The k-NN algorithm was applied to represent extracted tabular data as k-NNGs, which were then used as inputs for the GCN. Optimal configurations for the GCN were selected based on Taguchi experiments.

2. GCN using Taguchi with Selective Weighted Loss (SWL): Introduced the SWL method to improve class-specific accuracy, particularly targeting the Mild class. The SWL method allowed the model to reallocate focus and improve the accuracy of the Mild class, while still maintaining a strong performance across other classes.

3. Hybrid Graph-CNN Decision Fusion (HG-CDF) for Load-Dependent Fault Classification: Developed the HG-CDF approach, combining the strengths of the GCN and the 1D-CNN for load-dependent fault classification. This hybrid model significantly improved performance across all fault classes, especially in the Mild class, while maintaining high computational efficiency with a total training time of 3.28 min.

Chapter 8: Conclusion

8.1 Conclusion

In the rapidly evolving fields of Machine Learning (ML) and Deep Learning (DL), innovative applications are continually being developed to address complex problems across various industries. These technologies have proven especially transformative in industrial machinery maintenance, enhancing fault classification accuracy and operational efficiency. This thesis has contributed to this transformative wave by advancing Induction Motor (IM) bearings fault classification through innovative methodologies that leverage multimodal data, Artificial Intelligence (AI), and adaptive techniques.

The research has focused on multimodal data preprocessing for IM fault classification, proposing a methodology that combines multi-sensor images into a single Stitched Multimodal Image, which is then processed using pre-trained Convolutional Neural Networks (CNNs), specifically SqueezeNet and Residual Network-18 (ResNet-18). This study first assessed the classification accuracy of compromised-quality thermal images. Then, this was enhanced through multimodal preprocessing by integrating Continuous Wavelet Transform (CWT)-encoded vibration signals with the original thermal images and applying the Gramian Angular Field (GAF) technique. The results demonstrate that the images stitched with GAF achieved an overall mean accuracy of 98.39% ± 1.07%, while those using CWT recorded 96.89% ± 1.38%. The proposed image fusion preprocessing approach, utilising the GAF signal encoding technique, improved classification accuracy by 12.5%, achieving 99.1% ± 0.5% accuracy when using both ResNet-18 and SqueezeNet, compared to the 87.96% accuracy achieved with compromised thermal images alone.

Recognising the importance of thermal images, this research has explored generating artificial thermal images using Wasserstein Generative Adversarial Networks withGradient Penalty (WGAN-GP) and conditional Wasserstein Generative Adversarial Networks with Gradient Penalty (cWGAN-GP). A three-level evaluation approach has been applied to ensure that the generated images for IM bearing faults closely resembled real images collected in the lab. This included visual and qualitative assessments, quantitative metrics such as Earth Mover's Distance (EMD) and Maximum Mean Discrepancy (MMD), and classification accuracy using a pre-trained AlexNet model. The cWGAN-GP, which incorporates class information based on the bearing health conditions used as classes, demonstrated further improvements in image generation. While WGAN-GP slightly

outperformed in EMD, cWGAN-GP-generated images exhibited a closer statistical resemblance to real images with a lower MMD score. The cWGAN-GP dataset achieved 98.41% accuracy on unseen real images, with the entire training process completed in just 7.5 hours for all classes, compared to the 18-37 hours required by WGAN-GP for each class. This evaluation has demonstrated that the cWGAN-GP approach effectively generates artificial thermal images that resemble real images. Thus, this evaluation indicates that the cWGAN-GP approach is suitable for generating artificial thermal images that resemble real ones, making them effective for more accurate fault classification in IM bearings.

Furthermore, this study has proposed a Customised Load Adaptive Framework (CLAF) to address the literature's rarely discussed impact of Load Factors (LFs). The CLAF's first phase has revealed load-dependent patterns using wavelet energy, while the second phase has dealt with tailored methodologies specifically for the Machinery Failure Prevention Technology (MFPT) bearing dataset, identifying new load-dependent fault subclasses: Mild, Moderate, Severe, and Normal (fault-free) or Healthy condition'. The CLAF has demonstrated its ability to detect nuanced fault variations under various LF conditions, achieving 96.30% ± 0.50% accuracy in 18.155 s during five-fold cross-validation with a Wide Neural Network (WNN). Building on CLAF, two methodologies for Load-Dependent Fault Classification have been proposed: the Load-Dependent Multimodal Vibration Signal Enhancement and Fusion (LD-MVSEF) method and the Hybrid Graph-CNN Decision Fusion (HG-CDF) method.

The LD-MVSEF method has been used to effectively integrate diverse machine learning models to optimise CLAF load-dependent fault subclass classification. It extracts features independently from multiple data representations within a single source, utilising three distinct feature extraction channels. A fusion module has been used to consolidate these individual decisions into a unified classification result. LD-MVSEF utilises three channels: Channel 1 extracts features from the time and frequency domains, Channel 2 converts raw vibration signals into wavelet scalograms, and Channel 3 uses the Gramian Angular Difference Field (GADF) to generate two-dimensional (2D) images. Each channel was trained using different classifiers, with the most accurate being selected for all CLAF load-dependent fault subclasses. Using top-performing models like CubicSVM and AlexNet

across these input channels, LD-MVSEF achieved an overall accuracy of 99.04% ± 0.22%, based on five runs, confirming its efficacy and stability across multiple trials.

Furthermore, this thesis has explored the potential of Graph Convolutional Networks (GCNs) for condition monitoring, focusing on fast model training and accurate fault categorisation. The study began with the GCN using Taguchi, which transformed tabular data into graph structures using the k-Nearest Neighbours (kNN) method, demonstrating strong performance in load-dependent fault classification with a mean accuracy of 89.01% ± 1.25% across nine experiments. The GCN performed exceptionally well in the Moderate (98.92%), Severe (100%), and Normal (fault-free) or Healthy condition (100%) classes, while showing room for improvement in the Mild class, achieving 66.67% accuracy. To address this, the GCN using Taguchi with Selective Weighted Loss (SWL) was introduced, enhancing class-specific accuracy and improving the Mild class performance from 66.67% to 84.52% ± 1.96% over five runs. This improvement was achieved while maintaining overall model robustness, with accuracy at 90.70% ± 0.15%. Finally, the HG-CDF method was developed to further enhance overall accuracy across all classes. By combining the GCN's ability to capture complex relationships with the 1D-CNN's strength in detecting sequential patterns, this hybrid approach resulted in an impressive overall accuracy of 99.19%, while maintaining computational efficiency with a total training time of just 3.28 min.

LD-MVSEF and HG-CDF methodologies have demonstrated strong classification capabilities for the Mild and Moderate fault subclasses. LD-MVSEF achieved higher accuracy in the Moderate class (99.15%) than HG-CDF (97.85%). Still, HG-CDF outperformed LD-MVSEF in the Mild class (98.92%) and completed training in significantly less time—3 min 28 s compared to LD-MVSEF's 18 min 30 s. This comparison suggests that while LD-MVSEF excels in accuracy for Moderate conditions, HG-CDF offers a faster, more efficient solution with a competitive performance, making it a promising approach for condition monitoring.

The methodologies developed in this thesis have the potential to be generalised for broader applications. This research lays the groundwork for advancing condition monitoring systems, providing a framework that could significantly enhance fault classification in future machinery maintenance and monitoring innovations.

8.2 Contributions to Knowledge

This thesis has made significant strides in advancing fault classification technologies for IMs, aligning closely with the structured research questions designed and presented in Section 1.3 to probe various aspects of this field. The following clarifies how each contribution addresses the corresponding research question:

1. Novel Multimodal Data Preprocessing for IM Fault Classification: This contribution addresses the first research question, demonstrating how integrating thermal and vibration data in a multimodal system enhances fault detection capabilities.
2. Conditional Wasserstein Generative Adversarial Network with Gradient Penalty (cWGAN-GP) for Generating Artificial Thermal Images of Motor Faults: This technique relates to the second research question by demonstrating the use of Generative Adversarial Networks (GANs) to create a new dataset of artificial thermal images. It illustrates how these images are able to replicate various health conditions of IMs and discusses methods for effectively assessing their similarity to real images.
3. Customised Load Adaptive Framework (CLAF) for Fault Classification: This framework supports the third research question by developing an innovative methodology to identify and classify new load-dependent fault subclasses using advanced techniques, thus enhancing fault classification precision.
4. Load-Dependent Multimodal Vibration Signal Enhancement and Fusion (LD-MVSEF) for Load-Specific Condition Monitoring**:** This contribution is linked to the fourth research question. It explains how the three-channel fusion technique can be optimised to improve the health assessment of machinery, focusing primarily on load-dependent fault subclasses in industrial settings.
5. Graph Convolutional Network (GCN) and Hybrid Graph-CNN Decision Fusion (HG-CDF) for Load-Dependent Fault Classification: This final contribution addresses the fifth research question, demonstrating how Graph Neural Networks (GNNs) effectively classify faults using tabular vibration signals by transforming them into graph structures through the k-Nearest Neighbour Graph (k-NNG) method. The introduction of the HG-CDF method, which integrates GCNs with 1D-CNNs, resolves classification limitations, enhancing performance across all CLAF load-dependent subclasses while maintaining computational efficiency.

8.3 Study Limitations

The current study was conducted using Google Colab's GPU resources and MATLAB software to manage computational demands. Notably, creating artificial images required significant computational capabilities, highlighting the need for robust processing environments. The current research utilised the MFPT bearing dataset, widely recognised as a standard dataset in this field, thereby helping to ensure the relevance and comparability of the findings. The artificial images were also generated based on data collected in a suitable environment at Cardiff University's laboratory. This controlled setting helped to maintain consistent conditions during image capture but may also have the effect of limiting the generalisability of the results to less controlled environments. Such factors should be considered when interpreting the outcomes and applicability of this research.

8.4 Future Work

Building on the achievements of this thesis, future research should focus on expanding the integration of additional sensory data, such as Acoustic Emission (AE) sensing and electrical signals, to enrich multimodal datasets for enhanced fault detection accuracy. Optimising models for real-time data processing and employing edge computing is critical to ensure that the methodologies developed are viable in operational environments. There is a significant opportunity to investigate advanced neural network structures, including further exploration of GNNs, to enhance model robustness and efficiency across various types of machinery. Developing algorithms that predict mechanical failures before they occur could significantly reduce maintenance costs and downtime, making fault classification more proactive. Further work is needed to make artificial image creation using the cWGAN-GP more successful and to test this approach on different datasets and potentially make these datasets available online for organisations concerned with data privacy.

Additionally, exploring different data representations and developing more sophisticated decision fusion techniques could enhance algorithm decision accuracy, including utilising data augmentation techniques to handle anomalies in thermal images. Continuing to create the CLAF will enable deeper exploration into load-dependent fault classification, fostering a more customised approach that accommodates operational variances. Lastly, given the promising results in minimising training times and converting

tabular data into graphical structures, future research should extend the use of GNNs within industrial condition monitoring, potentially establishing GNNs as a cornerstone in future fault classification technologies. By prioritising these areas, future research will extend the theoretical advances made and focus on practical applications and real-world deployment, ensuring that the next generation of fault classification tools is innovative and directly applicable to the needs of the industry.

References

Abdel-Jaber, H., Devassy, D., Al Salam, A., Hidaytallah, L. and El-Amir, M. 2022. A Review of Deep Learning Algorithms and Their Applications in Healthcare. *Algorithms* 15(2). doi: 10.3390/a15020071.

Abdelrazik, M.A., Zekry, A. and Mohamed, W.A. 2023. Efficient Deep Learning Algorithm for Egyptian Sign Language Recognition. In: *2023 33rd Conference of Open Innovations Association (FRUCT)*. IEEE, pp. 3–8. Available at: https://ieeexplore.ieee.org/document/10142991/.

Ahmadzadeh, M., Zahrai, S.M. and Bitaraf, M. 2024. An integrated deep neural network model combining 1D CNN and LSTM for structural health monitoring utilizing multisensor time-series data. *Structural Health Monitoring* (3). Available at: https://journals.sagepub.com/doi/10.1177/14759217241239041.

Ahmed, H. and Nandi, A.K. 2018. Compressive Sampling and Feature Ranking Framework for Bearing Fault Classification With Vibration Signals. *IEEE Access* 6, pp. 44731–44746. doi: 10.1109/ACCESS.2018.2865116.

Ahmed, H.O.A. and Nandi, A.K. 2022. Vibration Image Representations for Fault Diagnosis of Rotating Machines: A Review. *Machines* 10(12), pp. 1–36. doi: 10.3390/machines10121113.

Al-Musawi, A.K., Anayi, F. and Packianather, M. 2020. Three-phase induction motor fault detection based on thermal image segmentation. *Infrared Physics & Technology* 104, p. 103140. Available at: https://linkinghub.elsevier.com/retrieve/pii/S1350449519304207.

Al-Qerem, A., Alsalman, Y.S. and Mansour, K. 2019. Image generation using different models of generative adversarial network. *Proceedings - 2019 International Arab Conference on Information Technology, ACIT 2019* , pp. 241–245. doi: 10.1109/ACIT47987.2019.8991120.

Alharbi, A.H., Towfek, S.K., Abdelhamid, A.A., Ibrahim, A., Eid, M.M., Khafaga, D.S.,

Khodadadi, N., Abualigah, L. and Saber, M. 2023. Diagnosis of Monkeypox Disease Using Transfer Learning and Binary Advanced Dipper Throated Optimization Algorithm. *Biomimetics* 8, 313(3), pp. 1–21. doi: 10.3390/biomimetics8030313.

Alotaibi, A. 2020. Deep generative adversarial networks for image-to-image translation: A review. *Symmetry* 12(10), pp. 1–26. doi: 10.3390/sym12101705.

Alotaibi, M., Asli, B.H.S. and Khan, M. 2021. Non-invasive inspections: A review on methods and tools. *Sensors* 21(24). doi: 10.3390/s21248474.

Alqahtani, H., Kavakli-Thorne, M. and Kumar, G. 2019. An Analysis of Evaluation Metrics of GANs. *International Conference on Information Technology and Applications* 7(July)

Alqahtani, H., Kavakli-Thorne, M. and Kumar, G. 2021. Applications of Generative Adversarial Networks (GANs): An Updated Review. *Archives of Computational Methods in Engineering* 28(2), pp. 525–552. Available at: https://doi.org/10.1007/s11831-019-09388-y.

Alrashedy, H.H.N., Almansour, A.F., Ibrahim, D.M. and Hammoudeh, M.A.A. 2022. BrainGAN: Brain MRI Image Generation and Classification Framework Using GAN Architectures and CNN Models. *Sensors* 22(11). doi: 10.3390/s22114297.

Alvarado-Hernandez, A.I., Zamudio-Ramirez, I., Jaen-Cuellar, A.Y., Osornio-Rios, R.A., Donderis-Quiles, V. and Antonino-Daviu, J.A. 2022. Infrared Thermography Smart Sensor for the Condition Monitoring of Gearbox and Bearings Faults in Induction Motors. *Sensors* 22(16). doi: 10.3390/s22166075.

Alzubaidi, L., Duan, Y., Al-Dujaili, A., Ibraheem, I.K., Alkenani, A.H., Santamaria, J., Fadhel, M.A., Al-Shamma, O. and Zhang, J. 2021. Deepening into the suitability of using pre-trained models of ImageNet against a lightweight convolutional neural network in medical imaging: an experimental study. *PeerJ Computer Science* 7, pp. 1–27. doi: 10.7717/peerj-cs.715.

Amanollah, H., Asghari, A. and Mashayekhi, M. 2023. Damage detection of structures based on wavelet analysis using improved AlexNet. *Structures* 56(May), p. 105019. Available at: https://doi.org/10.1016/j.istruc.2023.105019.

Ambrożkiewicz, B., Syta, A., Georgiadis, A., Gassner, A., Litak, G. and Meier, N. 2023. Intelligent Diagnostics of Radial Internal Clearance in Ball Bearings with Machine Learning Methods. *Sensors* 23, 5875(13). doi: 10.3390/s23135875.

Arellano-Espitia, F., Delgado-Prieto, M., Martinez-Viol, V., Saucedo-Dorantes, J.J. and Osornio-Rios, R.A. 2020. Deep-Learning-Based Methodology for Fault Diagnosis in Electromechanical Systems. *Sensors* 20(14), p. 3949. Available at: https://www.mdpi.com/1424-8220/20/14/3949.

Arjovsky, M., Chintala, S. and Bottou, L. 2017a. Wasserstein GAN. Available at: http://arxiv.org/abs/1701.07875.

Arjovsky, M., Chintala, S. and Bottou, L. 2017b. Wasserstein generative adversarial networks. *34th International Conference on Machine Learning, ICML 2017* 1, pp. 298–321.

Asutkar, S. and Tallur, S. 2023. Deep transfer learning strategy for efficient domain generalisation in machine fault diagnosis. *Scientific Reports* 13(1), pp. 1–9. Available at: https://doi.org/10.1038/s41598-023-33887-5.

Bai, H., Yan, H., Zhan, X., Wen, L. and Jia, X. 2022. Fault Diagnosis Method of Planetary Gearbox Based on Compressed Sensing and Transfer Learning. *Electronics* 11(11), p. 1708. Available at: https://www.mdpi.com/2079-9292/11/11/1708.

Barrera-Llanga, K., Burriel-Valencia, J., Sapena-Bañó, Á. and Martínez-Román, J. 2023. A Comparative Analysis of Deep Learning Convolutional Neural Network Architectures for Fault Diagnosis of Broken Rotor Bars in Induction Motors. *Sensors* 23(19). doi: 10.3390/s23198196.

Bechhoefer, E. 2013. Condition Based Maintenance Fault Database for Testing of Diagnostic and Prognostics Algorithms. Available at: https://www.mfpt.org/fault-data-sets/ [Accessed: 10 October 2023].

Bechhoefer, E. 2016. A Quick Introduction to Bearing Envelope Analysis. *Journal of Chemical Information and Modeling* 53(802), pp. 1–10. Available at: https://www.mfpt.org/wp-content/uploads/2018/03/MFPT-Bearing-Envelope-Analysis.pdf.

Bisht, B., Vimal, J. and Chaturvedi, V. 2013. Parametric Optimization of Electrochemical Machining Using Signal-To-Noise (S / N) Ratio. *International Journal of Modern Engineering Research* 3(4), pp. 1999–2006.

Bodo, R., Bertocco, M. and Bianchi, A. 2021. Cost-Sensitive Fault Identification in Predictive Maintenance Applications: a Case Study. In: *2021 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. IEEE, pp. 1–6. Available at: https://ieeexplore.ieee.org/document/9459829/.

Borga, M. and Carlsson, T. 1992. *A Survey of Current Techniques for Reinforcement Learning*. Linköping University, Department of Electrical Engineering. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.46.1631.

Borghi, P.H., Zakordonets, O. and Teixeira, J.P. 2021. A COVID-19 time series forecasting model based on MLP ANN. *Procedia Computer Science* 181(2019), pp. 940–947. Available at: https://doi.org/10.1016/j.procs.2021.01.250.

Borji, A. 2019. Pros and cons of GAN evaluation measures. *Computer Vision and Image Understanding* 179, pp. 41–65. doi: 10.1016/j.cviu.2018.10.009.

Boudiaf, R., Abdelkarim, B. and Issam, H. 2024. Bearing fault diagnosis in induction motor using continuous wavelet transform and convolutional neural networks. *International Journal of Power Electronics and Drive Systems* 15(1), pp. 591–602. doi: 10.11591/ijpeds.v15.i1.pp591-602.

Camacho-Bello, C.J., Gutiérrez-Lazcano, L. and Ortega-Mendoza, R.M. 2022. Rotation-invariant image classification using a novel 1D CNN and Multichannel Accurate Bessel-Fourier moments. *Pädi Boletín Científico de Ciencias Básicas e Ingenierías del ICBI* 10(Especial3), pp. 1–4. Available at: https://repository.uaeh.edu.mx/revistas/index.php/icbi/article/view/8874.

Cao, P., Zhu, Z., Wang, Z., Zhu, Y. and Niu, Q. 2022. Applications of graph convolutional networks in computer vision. *Neural Computing and Applications* 34(16), pp. 13387–13405. Available at: https://doi.org/10.1007/s00521-022-07368-1.

Castro-Ospina, A.E., Solarte-Sanchez, M.A., Vega-Escobar, L.S., Isaza, C. and Martínez-Vargas, J.D. 2024. Graph-Based Audio Classification Using Pre-Trained Models and Graph Neural Networks. *Sensors* 24(7), pp. 1–12. doi: 10.3390/s24072106.

Chandaliya, R., Timilsina, M., Breslin, J. and Serrano, M. 2023. Towards Graph-Based Semi-Supervised Learning on Audio Embeddings for Label Classification. In: *2023 International Conference on Machine Learning and Applications (ICMLA)*. IEEE, pp. 1385–1391. Available at: https://ieeexplore.ieee.org/document/10459852/.

Chandran, L.R., Jayagopal, N., Lal, L.S., Narayanan, C., Deepak, S. and Harikrishnan, V. 2021. Residential Load Time Series Forecasting using ANN and Classical Methods. *Proceedings of the 6th International Conference on Communication and Electronics Systems, ICCES 2021* , pp. 1508–1515. doi: 10.1109/ICCES51350.2021.9488969.

Chang, H.C., Wang, Y.C., Shih, Y.Y. and Kuo, C.C. 2022. Fault Diagnosis of Induction Motors with Imbalanced Data Using Deep Convolutional Generative Adversarial Network. *Applied Sciences (Switzerland)* 12(8). doi: 10.3390/app12084080.

Chang, M., Yao, D. and Yang, J. 2023. Intelligent Fault Diagnosis of Rolling Bearings Using Efficient and Lightweight ResNet Networks Based on an Attention Mechanism. *IEEE Sensors Journal* 23(9), pp. 9136–9145. doi: 10.1109/JSEN.2023.3251654.

Chao, M.A., Kulkarni, C., Goebel, K. and Fink, O. 2019. Hybrid deep fault detection and isolation: Combining deep neural networks and system performance models. *International Journal of Prognostics and Health Management* 10(4). doi: 10.36001/ijphm.2019.v10i4.2621.

Chen, B., Zhu, D., Wang, Y. and Zhang, P. 2022a. An Approach to Combine the Power of Deep Reinforcement Learning with a Graph Neural Network for Routing Optimization. *Electronics (Switzerland)* 11(3). doi: 10.3390/electronics11030368.

Chen, R., Huang, W., Huang, B., Sun, F. and Fang, B. 2020a. Reusing Discriminators for Encoding: Towards Unsupervised Image-To-Image Translation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* , pp. 8165–8174.

doi: 10.1109/CVPR42600.2020.00819.

Chen, S.-H., Kuo, Y. and Lin, J.-K. 2021. Using Mahalanobis distance and decision tree to analyze abnormal patterns of behavior in a maintenance outsourcing process-a case study. *Journal of Quality in Maintenance Engineering* 27(2), pp. 253–263. Available at: https://www.emerald.com/insight/content/doi/10.1108/JQME-04-2019-0037/full/html.

Chen, S., Yu, J. and Wang, S. 2022b. One-dimensional convolutional neural network-based active feature extraction for fault detection and diagnosis of industrial processes and its understanding via visualization. *ISA Transactions* 122, pp. 424–443. Available at: https://doi.org/10.1016/j.isatra.2021.04.042.

Chen, Y., Xu, X. and Jia, J. 2020b. Domain Adaptive Image-to-Image Translation. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 5273–5282. Available at: https://ieeexplore.ieee.org/document/9156656/.

Choudhary, A., Mian, T. and Fatima, S. 2021. Convolutional neural network based bearing fault diagnosis of rotating machine using thermal images. *Measurement* 176(February), p. 109196. Available at: https://doi.org/10.1016/j.measurement.2021.109196.

Cinar, E. 2022. A Sensor Fusion Method using Deep Transfer Learning for Fault Detection in Equipment Condition Monitoring. In: *2022 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*. Biarritz, France, 8–12 August: IEEE, pp. 1–6. Available at: https://ieeexplore.ieee.org/document/9894141/.

Cui, J., Zhong, Q., Zheng, S., Peng, L. and Wen, J. 2022. A Lightweight Model for Bearing Fault Diagnosis Based on Gramian Angular Field and Coordinate Attention. *Machines* 10(4), p. 282. Available at: https://www.mdpi.com/2075-1702/10/4/282.

Debie, E., Fernandez Rojas, R., Fidock, J., Barlow, M., Kasmarik, K., Anavatti, S., Garratt, M. and Abbass, H.A. 2021. Multimodal Fusion for Objective Assessment of Cognitive Workload: A Review. *IEEE Transactions on Cybernetics* 51(3), pp. 1542–1555. doi: 10.1109/TCYB.2019.2939399.

Delgoshaei, P., Heidarinejad, M. and Austin, M.A. 2022. A Semantic Approach for Building

System Operations: Knowledge Representation and Reasoning. *Sustainability* 14(10), p. 5810. Available at: https://www.mdpi.com/2071-1050/14/10/5810.

Deng, A. and Hooi, B. 2021. Graph Neural Network-Based Anomaly Detection in Multivariate Time Series. *35th AAAI Conference on Artificial Intelligence, AAAI 2021* 5A, pp. 4027–4035. doi: 10.1609/aaai.v35i5.16523.

Diao, Y. and Zhang, Q. 2021. Optimization of Management Mode of Small- and Medium-Sized Enterprises Based on Decision Tree Model. Chen, M. ed. *Journal of Mathematics* 2021, pp. 1–9. Available at: https://www.hindawi.com/journals/jmath/2021/2815086/.

Divya, P. and Devi, B.A. 2021. Epileptic EEG Signal Denoising Enhancement Using Improved Threshold Based Wavelet Method. In: *2021 International Conference on System, Computation, Automation and Networking (ICSCAN)*. IEEE, pp. 1–4. Available at: https://ieeexplore.ieee.org/document/9526343/.

Djemili, I., Medoued, A. and Soufi, Y. 2023. A Wind Turbine Bearing Fault Detection Method Based on Improved CEEMDAN and AR-MEDA. *Journal of Vibration Engineering & Technologies* (0123456789), pp. 1–22. Available at: https://link.springer.com/10.1007/s42417-023-01117-x.

Dong, W., Moses, C. and Li, K. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In: *Proceedings of the 20th international conference on World wide web*. New York, NY, USA: ACM, pp. 577–586. Available at: https://dl.acm.org/doi/10.1145/1963405.1963487.

Du, B., Huang, J., Barton, R., Sam, S., Yuan, C., Galloway, G. and Tutar, I. 2023. Enhancing Catalog Relationship Problems with Heterogeneous Graphs and Graph Neural Networks Distillation. *International Conference on Information and Knowledge Management, Proceedings* , pp. 4545–4551. doi: 10.1145/3583780.3615472.

Du, Y., Zhang, W., Wang, J. and Wu, H. 2019. DCGAN based data generation for process monitoring. *Proceedings of 2019 IEEE 8th Data Driven Control and Learning Systems Conference, DDCLS 2019* , pp. 410–415. doi: 10.1109/DDCLS.2019.8908922.

Edeh, M.O., Khalaf, O.I., Tavera, C.A., Tayeb, S., Ghouali, S., Abdulsahib, G.M., Richard-Nnabu, N.E. and Louni, A. 2022. A Classification Algorithm-Based Hybrid Diabetes Prediction Model. *Frontiers in Public Health* 10(March), pp. 1–7. Available at: https://www.frontiersin.org/articles/10.3389/fpubh.2022.829519/full.

Elshenawy, L.M., Chakour, C. and Mahmoud, T.A. 2022. Fault detection and diagnosis strategy based on k-nearest neighbors and fuzzy C-means clustering algorithm for industrial processes. *Journal of the Franklin Institute* 359(13), pp. 7115–7139. Available at: https://doi.org/10.1016/j.jfranklin.2022.06.022.

Emem Patrick Ekpo and James Eke 2024. An intelligent control for reducing third-party interference in oil and gas pipeline using Deep Q-Networks (DQN). *Global Journal of Engineering and Technology Advances* 18(3), pp. 075–081. doi: 10.30574/gjeta.2024.18.3.0233.

Engelmann, J. and Lessmann, S. 2021. Conditional Wasserstein GAN-based oversampling of tabular data for imbalanced learning. *Expert Systems with Applications* 174(Ml). doi: 10.1016/j.eswa.2021.114582.

Fan, H., Ma, J., Zhang, X., Xue, C., Yan, Y. and Ma, N. 2022. Intelligent data expansion approach of vibration gray texture images of rolling bearing based on improved WGAN-GP. *Advances in Mechanical Engineering* 14(3), pp. 1–11. doi: 10.1177/16878132221086132.

Fatima Ezzahra, K., Najat, R. and Jaafar, A. 2023. Comparative Analysis of Transfer Learning-Based CNN Approaches for Recognition of Traffic Signs in Autonomous Vehicles. Bourekkadi, S. et al. eds. *E3S Web of Conferences* 412, p. 01096. Available at: https://www.e3s-conferences.org/10.1051/e3sconf/202341201096.

Fazel, M., Ge, R., Kakade, S.M. and Mesbahi, M. 2018. Global Convergence of Policy Gradient Methods for the Linear Quadratic Regulator. *35th International Conference on Machine Learning, ICML 2018* 4, pp. 2385–2413.

Fei, Z., Guo, J., Gong, H., Ye, L., Attahi, E. and Huang, B. 2023. A GNN Architecture With Local and Global-Attention Feature for Image Classification. *IEEE Access* 11(October), pp.

110221–110233. Available at: https://ieeexplore.ieee.org/document/10148955/.

Ferraro, A., Galli, A., Moscato, V. and Sperli, G. 2020. A novel approach for predictive maintenance combining GAF encoding strategies and deep networks. In: *2020 IEEE 6th International Conference on Dependability in Sensor, Cloud and Big Data Systems and Application (DependSys)*. IEEE, pp. 127–132. Available at: https://ieeexplore.ieee.org/document/9356422/.

Frosini, L. and Bassi, E. 2010. Stator Current and Motor Efficiency as Indicators for Different Types of Bearing Faults in Induction Motors. *IEEE Transactions on Industrial Electronics* 57(1), pp. 244–251. Available at: http://ieeexplore.ieee.org/document/5166499/.

Ganapathy, S., Mallidi, S.H. and Hermansky, H. 2014. Robust feature extraction using modulation filtering of autoregressive models. *IEEE Transactions on Audio, Speech and Language Processing* 22(8), pp. 1285–1295. doi: 10.1109/taslp.2014.2329190.

Gangsar, P. and Tiwari, R. 2020. Signal based condition monitoring techniques for fault detection and diagnosis of induction motors: A state-of-the-art review. *Mechanical Systems and Signal Processing* 144, p. 106908. Available at: https://doi.org/10.1016/j.ymssp.2020.106908.

Gao, X., Deng, F. and Yue, X. 2020. Data augmentation in fault diagnosis based on the Wasserstein generative adversarial network with gradient penalty. *Neurocomputing* 396(xxxx), pp. 487–494. Available at: https://linkinghub.elsevier.com/retrieve/pii/S0925231219304953.

Gao, Y., Chai, C., Li, H. and Fu, W. 2023. A Deep Learning Framework for Intelligent Fault Diagnosis Using AutoML-CNN and Image-like Data Fusion. *Machines* 11(10), p. 932. Available at: https://www.mdpi.com/2075-1702/11/10/932.

Grabowski, N., Kremser, R., Düssel, R., Mulder, A. and Tutsch, D. 2023. Information Extraction from Industrial Sensor Data Using Time Series Meta-Features. *Applied Sciences (Switzerland)* 13(12), pp. 1–11. doi: 10.3390/app13127065.

Granados-Lieberman, D., Huerta-Rosales, J.R., Gonzalez-Cordoba, J.L., Amezquita-

Sanchez, J.P., Valtierra-Rodriguez, M. and Camarena-Martinez, D. 2023. Time-Frequency Analysis and Neural Networks for Detecting Short-Circuited Turns in Transformers in Both Transient and Steady-State Regimes Using Vibration Signals. *Applied Sciences* 13(22), p. 12218. doi: 10.3390/app132212218.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V. and Courville, A. 2017. Improved Training of Wasserstein GANs. Available at: http://arxiv.org/abs/1704.00028.

Guney, G., Yigin, B.O., Guven, N., Alici, Y.H., Colak, B., Erzin, G. and Saygili, G. 2021. An overview of deep learning algorithms and their applications in neuropsychiatry. *Clinical Psychopharmacology and Neuroscience* 19(2), pp. 206–219. doi: 10.9758/cpn.2021.19.2.206.

Guo, T., Zhang, T., Lim, E., Lopez-Benitez, M., Ma, F. and Yu, L. 2022. A Review of Wavelet Analysis and Its Applications: Challenges and Opportunities. *IEEE Access* 10, pp. 58869–58903. doi: 10.1109/ACCESS.2022.3179517.

Gupta, P., Malhotra, P., Vig, L. and Shroff, G. 2018. Using features from pre-trained timenet for clinical predictions. *CEUR Workshop Proceedings* 2148, pp. 38–44.

Haarnoja, T., Zhou, A., Abbeel, P. and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *35th International Conference on Machine Learning, ICML 2018* 5, pp. 2976–2989.

Hakim, M., Omran, A.A.B., Ahmed, A.N., Al-Waily, M. and Abdellatif, A. 2023. A systematic review of rolling bearing fault diagnoses based on deep learning and transfer learning: Taxonomy, overview, application, open challenges, weaknesses and recommendations. *Ain Shams Engineering Journal* 14(4), p. 101945. Available at: https://doi.org/10.1016/j.asej.2022.101945.

Han, B., Jia, S., Liu, G. and Wang, J. 2020. Imbalanced Fault Classification of Bearing via Wasserstein Generative Adversarial Networks with Gradient Penalty. *Shock and Vibration* 2020. doi: 10.1155/2020/8836477.

Han, B., Zhang, H., Sun, M. and Wu, F. 2021a. A New Bearing Fault Diagnosis Method

Based on Capsule Network and Markov Transition Field/Gramian Angular Field. *Sensors* 21(22), p. 7762. Available at: https://www.mdpi.com/1424-8220/21/22/7762.

Han, T., Zhang, L., Yin, Z. and Tan, A.C.C. 2021b. Rolling bearing fault diagnosis with combined convolutional neural networks and support vector machine. *Measurement: Journal of the International Measurement Confederation* 177(February), p. 109022. Available at: https://doi.org/10.1016/j.measurement.2021.109022.

Hassanpour, M. and Malek, H. 2019. Document Image Classification using SqueezeNet Convolutional Neural Network. *5th Iranian Conference on Signal Processing and Intelligent Systems, ICSPIS 2019* (December), pp. 1–4. doi: 10.1109/ICSPIS48872.2019.9066032.

He, K., Zhang, X., Ren, S. and Sun, J. 2016. Deep Residual Learning for Image Recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 770–778. Available at: http://ieeexplore.ieee.org/document/7780459/.

He, X., Chang, Z., Zhang, L., Xu, H., Chen, H. and Luo, Z. 2022. A Survey of Defect Detection Applications Based on Generative Adversarial Networks. *IEEE Access* 10(September), pp. 113493–113512. doi: 10.1109/ACCESS.2022.3217227.

He, Y., Tang, H., Ren, Y. and Kumar, A. 2021. A semi-supervised fault diagnosis method for axial piston pump bearings based on DCGAN. *Measurement Science and Technology* 32(12), p. 125104. Available at: https://iopscience.iop.org/article/10.1088/1361-6501/ac1fbe.

He, Z., Fu, L., Lin, S. and Bo, Z. 2010. Fault detection and classification in EHV transmission line based on wavelet singular entropy. *IEEE Transactions on Power Delivery* 25(4), pp. 2156–2163. doi: 10.1109/TPWRD.2010.2042624.

Hejazi, S., Packianather, M. and Liu, Y. 2022. Novel Preprocessing of Multimodal Condition Monitoring Data for Classifying Induction Motor Faults Using Deep Learning Methods. In: *2022 IEEE 2nd International Symposium on Sustainable Energy, Signal Processing and Cyber Security (iSSSC)*. Gunupur Odisha, India, 15–17 December 2022: IEEE, pp. 1–6. Available at: https://ieeexplore.ieee.org/document/10051321/.

Hejazi, S., Packianather, M. and Liu, Y. 2023. A Novel approach using WGAN-GP and Conditional WGAN-GP for Generating Artificial Thermal Images of Induction Motor Faults. *Procedia Computer Science* 225, pp. 3681–3691. Available at: https://doi.org/10.1016/j.procs.2023.10.363.

Hejazi, S.Z., Packianather, M. and Liu, Y. 2024. A Novel Customised Load Adaptive Framework for Induction Motor Fault Classification Utilising MFPT Bearing Dataset. *Machines* 12(1), p. 44. Available at: https://www.mdpi.com/2075-1702/12/1/44.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B. and Hochreiter, S. 2017. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. *Advances in Neural Information Processing Systems* 2017-Decem(Nips), pp. 6627–6638. Available at: http://arxiv.org/abs/1706.08500.

Hu, B., Liu, J., Zhao, R., Xu, Y. and Huo, T. 2022. A New Fault Diagnosis Method for Unbalanced Data Based on 1DCNN and L2-SVM. *Applied Sciences (Switzerland)* 12(19). doi: 10.3390/app12199880.

Hu, L. and Zhang, Z. 2019. *EEG Signal Processing and Feature Extraction*. Hu, L. and Zhang, Z. eds. Singapore: Springer Singapore. Available at: http://link.springer.com/10.1007/978-981-13-9113-2.

Huang, M.-L. and Li, Y.-Z. 2021. Use of Machine Learning and Deep Learning to Predict the Outcomes of Major League Baseball Matches. *Applied Sciences* 11(10), p. 4499. Available at: https://www.mdpi.com/2076-3417/11/10/4499.

Huang, X., Han, K., Yang, Y., Bao, D., Tao, Q., Chai, Z. and Zhu, Q. 2024. Can GNN be Good Adapter for LLMs? Available at: http://arxiv.org/abs/2402.12984.

Hussain, M., Bird, J.J. and Faria, D.R. 2019. A Study on CNN Transfer Learning for Image Classification. In: *Advances in Intelligent Systems and Computing*., pp. 191–202. Available at: http://link.springer.com/10.1007/978-3-319-97982-3_16.

Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J. and Keutzer, K. 2016a. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size.

Available at: http://arxiv.org/abs/1602.07360.

Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J. and Keutzer, K. 2016b. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. (December), pp. 0–13. Available at: http://arxiv.org/abs/1602.07360.

Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J. and Keutzer, K. 2016c. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size., pp. 1–13. Available at: http://arxiv.org/abs/1602.07360.

Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J. and Keutzer, K. 2017. 50 X Fewer Parameters and < 0 . 5Mb Model Size. *Iclr* (April 2016), pp. 1–13.

Jain, P.H. and Bhosle, S.P. 2021. Study of effects of radial load on vibration of bearing using time-Domain statistical parameters. *IOP Conference Series: Materials Science and Engineering* 1070(1), p. 012130. doi: 10.1088/1757-899x/1070/1/012130.

Jain, P.H. and Bhosle, S.P. 2022. Analysis of vibration signals caused by ball bearing defects using time-domain statistical indicators. *International Journal of Advanced Technology and Engineering Exploration* 9(90), pp. 700–715. doi: 10.19101/IJATEE.2021.875416.

Jang, G.-B. and Cho, S.-B. 2021. Feature Space Transformation for Fault Diagnosis of Rotating Machinery under Different Working Conditions. *Sensors* 21(4), p. 1417. Available at: https://www.mdpi.com/1424-8220/21/4/1417.

Jayamaha, D.K.J.S., Lidula, N.W.A. and Rajapakse, A.D. 2019. Wavelet-Multi Resolution Analysis Based ANN Architecture for Fault Detection and Localization in DC Microgrids. *IEEE Access* 7, pp. 145371–145384. doi: 10.1109/ACCESS.2019.2945397.

Jia, Z., Liu, Z., Vong, C.M. and Pecht, M. 2019. A Rotating Machinery Fault Diagnosis Method Based on Feature Learning of Thermal Images. *IEEE Access* 7, pp. 12348–12359. doi: 10.1109/ACCESS.2019.2893331.

Jong, W.R., Huang, Y.M., Lin, Y.Z., Chen, S.C. and Chen, Y.W. 2020. Integrating Taguchi method and artificial neural network to explore machine learning of computer aided

engineering. *Journal of the Chinese Institute of Engineers, Transactions of the Chinese Institute of Engineers,Series A* 43(4), pp. 346–356. Available at: https://doi.org/10.1080/02533839.2019.1708804.

Jose, J.P., Ananthan, T. and Prakash, N.K. 2022. Ensemble Learning Methods for Machine Fault Diagnosis. In: *2022 Third International Conference on Intelligent Computing Instrumentation and Control Technologies (ICICICT)*. IEEE, pp. 1127–1134. Available at: https://ieeexplore.ieee.org/document/9917966/.

Kadam, V., Kumar, S., Bongale, A., Wazarkar, S., Kamat, P. and Patil, S. 2021. Enhancing Surface Fault Detection Using Machine Learning for 3D Printed Products. *Applied System Innovation* 4(2), p. 34. Available at: https://www.mdpi.com/2571-5577/4/2/34.

Kaji, M., Parvizian, J. and van de Venn, H.W. 2020. Constructing a Reliable Health Indicator for Bearings Using Convolutional Autoencoder and Continuous Wavelet Transform. *Applied Sciences* 10(24), p. 8948. Available at: https://www.mdpi.com/2076-3417/10/24/8948.

Kareem, A.B. and Hur, J.-W. 2022. Towards Data-Driven Fault Diagnostics Framework for SMPS-AEC Using Supervised Learning Algorithms. *Electronics* 11(16), p. 2492. Available at: https://www.mdpi.com/2079-9292/11/16/2492.

Khanjani, M. and Ezoji, M. 2021. Electrical fault detection in three-phase induction motor using deep network-based features of thermograms. *Measurement* 173(July 2020), p. 108622. Available at: https://doi.org/10.1016/j.measurement.2020.108622.

Koehler, F., Mehta, V., Zhou, C. and Risteski, A. 2021. Variational autoencoders in the presence of low-dimensional data: landscape and implicit bias., pp. 1–21. Available at: http://arxiv.org/abs/2112.06868.

Kou, R., Lian, S., Xie, N., Lu, B. and Liu, X. 2022. Image-based tool condition monitoring based on convolution neural network in turning process. *The International Journal of Advanced Manufacturing Technology* 119(5–6), pp. 3279–3291. Available at: https://link.springer.com/10.1007/s00170-021-08282-x.

Krizhevsky, A., Sutskever, I. and Hinton, G.E.H. 2007. *Handbook of Approximation*

Algorithms and Metaheuristics. Gonzalez, T. F. ed. Lake Tahoe, Nevada: Chapman and Hall/CRC. Available at: https://www.taylorfrancis.com/books/9781420010749.

Krizhevsky, A., Sutskever, I. and Hinton, G.E. 2017. ImageNet classification with deep convolutional neural networks. *Communications of the ACM* 60(6), pp. 84–90. doi: 10.1145/3065386.

Kufel, J., Bargieł-Łączek, K., Kocot, S., Koźlik, M., Bartnikowska, W., Janik, M., Czogalik, Ł., Dudek, P., Magiera, M., Lis, A., Paszkiewicz, I., Nawrat, Z., Cebula, M. and Gruszczyńska, K. 2023. What Is Machine Learning, Artificial Neural Networks and Deep Learning?—Examples of Practical Applications in Medicine. *Diagnostics* 13(15), p. 2582. Available at: https://www.mdpi.com/2075-4418/13/15/2582.

Kullu, O. and Cinar, E. 2022. A Deep-Learning-Based Multi-Modal Sensor Fusion Approach for Detection of Equipment Faults. *Machines* 10(11), p. 1105. doi: 10.3390/machines10111105.

Kumar, V., Mukherjee, S., Verma, A.K. and Sarangi, S. 2022. An AI-Based Nonparametric Filter Approach for Gearbox Fault Diagnosis. *IEEE Transactions on Instrumentation and Measurement* 71, 351661, pp. 1–11. doi: 10.1109/TIM.2022.3186700.

Kuncan, M., Kaplan, K., Minaz, M.R., Kaya, Y. and Ertunç, H.M. 2020. A novel feature extraction method for bearing fault classification with one dimensional ternary patterns. *ISA Transactions* 100, pp. 346–357. doi: 10.1016/j.isatra.2019.11.006.

Kusiak, A. 2020. Convolutional and generative adversarial neural networks in manufacturing. *International Journal of Production Research* 58(5), pp. 1594–1604. doi: 10.1080/00207543.2019.1662133.

Latha, R.S., Sreekanth, G.R., Suganthe, R.C., Geetha, M., Selvaraj, R.E., Balaji, S., Harini, K.R. and Ponnusamy, P.P. 2022. Stock Movement Prediction using KNN Machine Learning Algorithm. *2022 International Conference on Computer Communication and Informatics, ICCCI 2022* , pp. 0–4. doi: 10.1109/ICCCI54379.2022.9740781.

LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. 1998. Gradient-based learning applied to

document recognition. *Proceedings of the IEEE* 86(11), pp. 2278–2323. doi: 10.1109/5.726791.

Lee, G. and Kim, J. 2023. MTGEA: A Multimodal Two-Stream GNN Framework for Efficient Point Cloud and Skeleton Data Alignment. *Sensors* 23(5), p. 2787. Available at: https://www.mdpi.com/1424-8220/23/5/2787.

Lee, H., Han, S.-Y. and Park, K.-J. 2020. Generative Adversarial Network-based Missing Data Handling and Remaining Useful Life Estimation for Smart Train Control and Monitoring Systems. Routil, L. ed. *Journal of Advanced Transportation* 2020, pp. 1–15. Available at: https://www.hindawi.com/journals/jat/2020/8861942/.

Lee, S.Y., Bu, F., Yoo, J. and Shin, K. 2023. Towards Deep Attention in Graph Neural Networks: Problems and Remedies. *Proceedings of Machine Learning Research* 202, pp. 18733–18773.

Lee, Y.O., Jo, J. and Hwang, J. 2017. Application of deep neural network and generative adversarial network to industrial maintenance: A case study of induction motor fault detection. In: *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, pp. 3248–3253. Available at: http://ieeexplore.ieee.org/document/8258307/.

Li, B., Xian, Y., Zhang, D., Su, J., Hu, X. and Guo, W. 2021. Multi-Sensor Image Fusion: A Survey of the State of the Art. *Journal of Computer and Communications* 09(06), pp. 73–108. doi: 10.4236/jcc.2021.96005.

Li, D., Cao, M., Deng, T. and Zhang, S. 2019a. Wavelet packet singular entropy-based method for damage identification in curved continuous girder bridges under seismic excitations. *Sensors (Switzerland)* 19, 4272(19). doi: 10.3390/s19194272.

Li, J., Ying, Y., Ren, Y., Xu, S., Bi, D., Chen, X. and Xu, Y. 2019b. Research on rolling bearing fault diagnosis based on multi-dimensional feature extraction and evidence fusion theory. *Royal Society Open Science* 6(2), p. 181488. Available at: https://royalsocietypublishing.org/doi/10.1098/rsos.181488.

Li, S., Mao, J., Gong, X. and Li, Z. 2023a. A novel lidar signal noise reduction algorithm

based on improved deep belief network A novel lidar signal noise reduction algorithm based on improved deep belief network.

Li, X., Xu, L., Guo, H. and Yang, L. 2023b. Application of Graph Convolutional Neural Networks Combined with Single-Model Decision-Making Fusion Neural Networks in Structural Damage Recognition. *Sensors* 23(23), p. 9327. Available at: https://www.mdpi.com/1424-8220/23/23/9327.

Lin, C.-H., Lin, Y.-C., Wu, Y.-J., Chung, W.-H. and Lee, T.-S. 2021. A Survey on Deep Learning-Based Vehicular Communication Applications. *Journal of Signal Processing Systems* 93(4), pp. 369–388. Available at: https://link.springer.com/10.1007/s11265-020-01587-2.

Liu, L. and Si, Y.-W. 2022. 1D convolutional neural networks for chart pattern classification in financial time series. *The Journal of Supercomputing* 78(12), pp. 14191–14214. Available at: https://doi.org/10.1007/s11227-022-04431-5.

Liu, M.K. and Weng, P.Y. 2019. Fault diagnosis of ball bearing elements: A generic procedure based on time-frequency analysis. *Measurement Science Review* 19(4), pp. 185–194. doi: 10.2478/msr-2019-0024.

Liu, X., Li, T., Zhang, R., Wu, D., Liu, Y. and Yang, Z. 2021. A GAN and Feature Selection-Based Oversampling Technique for Intrusion Detection. *Security and Communication Networks* 2021. doi: 10.1155/2021/9947059.

Lorenz, A., Siewertsen, B., Kyhe Clemmensen, V., Blaamann Petersen, J., Friederich, J. and Lazarova-Molnar, S. 2022. Vibration Data Analysis for Fault Detection in Manufacturing Systems - A Systematic Literature Review. In: *2022 IEEE 17th Conference on Industrial Electronics and Applications (ICIEA)*. IEEE, pp. 851–857. Available at: https://ieeexplore.ieee.org/document/10006127/.

Lu, F., Tong, Q., Feng, Z., Wan, Q., An, G., Li, Y., Wang, M., Cao, J. and Guo, T. 2022. Explainable 1DCNN with demodulated frequency features method for fault diagnosis of rolling bearing under time-varying speed conditions. *Measurement Science and Technology*

33(9), p. 095022. Available at: https://iopscience.iop.org/article/10.1088/1361-6501/ac78c5.

Lu, T., Yu, F., Han, B. and Wang, J. 2020. A generic intelligent bearing fault diagnosis system using convolutional neural networks with transfer learning. *IEEE Access* 8, pp. 164807–164814. doi: 10.1109/ACCESS.2020.3022840.

Luan, S., Hua, C., Lu, Q., Zhu, J., Zhao, M., Zhang, S., Chang, X.-W. and Precup, D. 2022. Revisiting Heterophily For Graph Neural Networks. In: *36th Conference on Neural Information Processing Systems.*, pp. 1–38. Available at: http://arxiv.org/abs/2210.07606.

Łuczak, D. 2024. Machine Fault Diagnosis through Vibration Analysis: Continuous Wavelet Transform with Complex Morlet Wavelet and Time–Frequency RGB Image Recognition via Convolutional Neural Network. *Electronics (Switzerland)* 13(2). doi: 10.3390/electronics13020452.

Lupea, I. and Lupea, M. 2022. Machine Learning Techniques for Multi-Fault Analysis and Detection on a Rotating Test Rig Using Vibration Signal. *Symmetry* 15(1), p. 86. Available at: https://www.mdpi.com/2073-8994/15/1/86.

Lyu, J. and Yu, Y. 2021. Potential Practical Applications of Basic Neural Networks. In: *2021 2nd International Conference on Intelligent Computing and Human-Computer Interaction (ICHCI)*. IEEE, pp. 91–94. Available at: https://ieeexplore.ieee.org/document/9708622/.

Ma, J., Jiang, X., Han, B., Wang, J., Zhang, Z. and Bao, H., 2023. Dynamic simulation model-driven fault diagnosis method for bearing under missing fault-type samples. Applied Sciences, 13(5), p.2857.

Ma, Y., Ren, H., Khailany, B., Sikka, H., Luo, L., Natarajan, K. and Yu, B. 2019. High Performance Graph Convolutional Networks with Applications in Testability Analysis. In: *Proceedings of the 56th Annual Design Automation Conference 2019*. New York, NY, USA: ACM, pp. 1–6. Available at: https://dl.acm.org/doi/10.1145/3316781.3317838.

Ma, Y., Wen, G., Cheng, S., He, X. and Mei, S. 2022. Multimodal Convolutional Neural Network Model With Information Fusion for Intelligent Fault Diagnosis in Rotating Machinery. *Measurement Science and Technology* . doi: 10.1088/1361-6501/ac7eb0.

Mahesh, A., Aadhavan, B.A., Meenaa, V.V., Omar, M.B., Ibrahim, R. Bin, Salehuddin, N.F. and Sujatha, R. 2022. Employment of ANN for Predictive Motor Maintenance and Bearing Fault Detection Using Park's Vector Analysis. In: *2022 IEEE 5th International Symposium in Robotics and Manufacturing Automation (ROMA)*. IEEE, pp. 1–6. Available at: https://ieeexplore.ieee.org/document/9915683/.

Martin, S.S. 2021. *Precision Medicine in Cardiovascular Disease Prevention*. Martin, S. S. ed. Cham: Springer International Publishing. Available at: https://link.springer.com/10.1007/978-3-030-75055-8.

Martinez-Herrera, A.L., Ferrucho-Alvarez, E.R., Ledesma-Carrillo, L.M., Mata-Chavez, R.I., Lopez-Ramirez, M. and Cabal-Yepez, E. 2022. Multiple Fault Detection in Induction Motors through Homogeneity and Kurtosis Computation. *Energies* 15, 1541(4). doi: 10.3390/en15041541.

Masud, M.A., Rahman, M.M., Bhadra, S. and Saha, S. 2019. Improved k-means algorithm using density estimation. *2019 International Conference on Sustainable Technologies for Industry 4.0, STI 2019* 0, pp. 1–6. doi: 10.1109/STI47673.2019.9068033.

Mathur, B. and Kaushik, M. 2018. In Object-Oriented Software Framework Improving Maintenance Exercises Through K-Means Clustering Approach. In: *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*. IEEE, pp. 1–7. Available at: https://ieeexplore.ieee.org/document/8519897/.

MathWorks-3 2024. Choose Classifier Options. Available at: https://www.mathworks.com/help/stats/choose-a-classifier.html [Accessed: 23 January 2024].

MathWorks-4 2023. squeezenet. Available at: https://www.mathworks.com/help/deeplearning/ref/squeezenet.html [Accessed: 8 February 2024].

MathWorks-5 2023. Pretrained Deep Neural Networks. Available at: https://www.mathworks.com/help/deeplearning/ug/pretrained-convolutional-neural-

networks.html [Accessed: 18 September 2022].

MathWorks-6 2024. Neural Network Structure. Available at: https://www.mathworks.com/help/stats/fitcnet.html#mw_a9fa4524-aecf-4d9b-95ef-58914f7780f3 [Accessed: 15 May 2024].

MathWorks-7 2024. Analyze and Select Features for Pump Diagnostics. Available at: https://www.mathworks.com/help/predmaint/ug/analyze-and-select-features-for-pump-diagnostics.html [Accessed: 27 November 2023].

McGhan, F. 2020a. *Transfer Learning for Induction Motor Fault Diagnosis*. Cardiff University.

McGhan, F. 2020b. Transfer Learning for Induction Motor Fault Diagnosis. Available at: https://github.com/frasermcghan/Year3Project [Accessed: 24 September 2022].

McGhan, F. and Feayherston, C. 2020. *Transfer Learning for Induction Motor Fault Diagnosis Fraser McGhan*. Cardiff University.

Mehmood, R. and Selwal, A. 2020. Fingerprint Biometric Template Security Schemes: Attacks and Countermeasures. In: *Lecture Notes in Electrical Engineering*., pp. 455–467. Available at: http://link.springer.com/10.1007/978-3-030-29407-6_33.

Metwally, M., Hassan, M.M. and Hassaan, G. 2020. Diagnosis of Rotating Machines Faults Using Artificial Intelligence Based on Preprocessing for Input Data. In: *In Proceedings of the 26th IEEE Conference of Open Innovations Association FRUCT (FRUCT26), Yaroslavl, Russia, 23–25 April 2020.*

Miikkulainen, R. 2023. Evolution of Neural Networks. In: *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, pp. 1008–1025. Available at: https://dl.acm.org/doi/10.1145/3583133.3595036.

Mo, H., Peng, Y., Wei, W., Xi, W. and Cai, T. 2023. SR-GNN Based Fault Classification and Location in Power Distribution Network. *Energies* 16(1). doi: 10.3390/en16010433.

Moreno-Gutierrez, S.S. and Garcia-Lopez, M. 2023. PREVENTION STRATEGY OF MAIN NON-COMMUNICABLE DISEASES USING ARTIFICIAL NEURAL NETWORKS. *DYNA NEW TECHNOLOGIES* 10(1), p. [10P.]-[10P.]. Available at: http://www.dyna-newtech.com/Articulos/Ficha.aspx?IdMenu=84f599bc-033c-4468-ae4e-43e44d36138f&Cod=10765&Idioma=en-GB.

Mustaqim, A.Z., Fadil, N.A. and Tyas, D.A. 2023. Artificial Neural Network for Classification Task in Tabular Datasets and Image Processing: A Systematic Literature Review. *Jurnal Online Informatika* 8(2), pp. 158–168. Available at: https://join.if.uinsgd.ac.id/index.php/join/article/view/1002.

Najwaini, E., Tarigan, T.E. and Putra, F.P. 2023. Application of the K-Nearest Neighbors (KNN) Algorithm on the Brain Tumor Dataset. *... of Artificial Intelligence in ...* 1(1), pp. 18–26. Available at: https://www.jurnal.yoctobrain.org/index.php/ijaimi/article/view/85.

Naman, S., Bhattacharyya, S. and Saha, T. 2020. *Remote Sensing and Advanced Encryption Standard Using 256-Bit Key*. doi: 10.1007/978-981-13-7403-6_18.

Napoletano, P., Piccoli, F. and Schettini, R. 2018. Anomaly detection in nanofibrous materials by CNN-based self-similarity. *Sensors (Switzerland)* 18(1). doi: 10.3390/s18010209.

Narayan, Y. 2021. Hb vsEMG signal classification with time domain and Frequency domain features using LDA and ANN classifier Materials Today : Proceedings Hb vsEMG signal classification with time domain and Frequency domain features using LDA and ANN classifier. *Materials Today: Proceedings* 37(October 2020), pp. 3226–3230. Available at: https://doi.org/10.1016/j.matpr.2020.09.091.

Naresh, M., Saxena, P. and Gupta, M. 2023. PPO-ABR: Proximal Policy Optimization based Deep Reinforcement Learning for Adaptive BitRate streaming. *2023 International Wireless Communications and Mobile Computing, IWCMC 2023* , pp. 199–204. doi: 10.1109/IWCMC58020.2023.10182379.

Naseh, M., Hasanzadeh, S., Dehghan, S.M., Rezaei, H. and Al-Sumaiti, A.S. 2022.

Optimized Design of Rotor Barriers in PM-Assisted Synchronous Reluctance Machines With Taguchi Method. *IEEE Access* 10, pp. 38165–38173. doi: 10.1109/ACCESS.2022.3165549.

Nayana, B.R. and Geethanjali, P. 2020. Improved Identification of Various Conditions of Induction Motor Bearing Faults. *IEEE Transactions on Instrumentation and Measurement* 69(5), pp. 1908–1919. doi: 10.1109/TIM.2019.2917981.

Nemani, V., Bray, A., Thelen, A., Hu, C. and Daining, S. 2022. Health index construction with feature fusion optimization for predictive maintenance of physical systems. *Structural and Multidisciplinary Optimization* 65, 349(12), pp. 1–23. Available at: https://doi.org/10.1007/s00158-022-03437-0.

Nguyen, C.D., Ahmad, Z. and Kim, J. 2021. Gearbox Fault Identification Framework Based on Novel Localized Adaptive Denoising Technique, Wavelet-Based Vibration Imaging, and Deep Convolutional Neural Network. *Applied Sciences* 11(16), p. 7575. Available at: https://www.mdpi.com/2076-3417/11/16/7575.

Nishat Toma, R., Kim, C.-H. and Kim, J.-M. 2021. Bearing Fault Classification Using Ensemble Empirical Mode Decomposition and Convolutional Neural Network. *Electronics* 10(11), p. 1248. Available at: https://www.mdpi.com/2079-9292/10/11/1248.

Nishat Toma, R. and Kim, J.-M. 2020. Bearing Fault Classification of Induction Motors Using Discrete Wavelet Transform and Ensemble Machine Learning Algorithms. *Applied Sciences* 10(15), p. 5251. Available at: https://www.mdpi.com/2076-3417/10/15/5251.

Niu, J., Yang, R., Guan, W. and Xie, Z. 2021. Spatial-Temporal Graph Convolutional Networks for Action Recognition with Adjacency Matrix Generation Network. In: *2021 2nd International Conference on Electronics, Communications and Information Technology (CECIT)*. IEEE, pp. 1150–1154. Available at: https://ieeexplore.ieee.org/document/9742187/.

Niu, S., Li, B., Wang, X. and Lin, H. 2020. Defect Image Sample Generation With GAN for Improving Defect Recognition. *IEEE Transactions on Automation Science and Engineering*

17(3), pp. 1–12. Available at: https://ieeexplore.ieee.org/document/9000806/.

Niu, Y., Pu, Y., Yang, Z., Li, X., Zhou, T., Ren, J., Hu, S., Li, H. and Liu, Y. 2023. LightZero: A Unified Benchmark for Monte Carlo Tree Search in General Sequential Decision Scenarios. (NeurIPS). Available at: http://arxiv.org/abs/2310.08348.

Njoku, O. 2019. Decision Trees and Their Application for Classification and Regression Problems. *MSU Graduate Theses*

de Oliveira, A.F.M., Magalhães, E. dos S., Paes, L.E. do. S., Pereira, M. and da Silva, L.R.R. 2023. A Thermal Analysis of LASER Beam Welding Using Statistical Approaches. *Processes* 11(7). doi: 10.3390/pr11072023.

Ozaltin, O. and Yeniay, O. 2023. A novel proposed CNN–SVM architecture for ECG scalograms classification. *Soft Computing* 27(8), pp. 4639–4658. Available at: https://doi.org/10.1007/s00500-022-07729-x.

Packianather, M.S., Al-Musawi, A.K. and Anayi, F. 2019. Bee for mining (B4M) – A novel rule discovery method using the Bees algorithm with quality-weight and coverage-weight. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science* 233(14), pp. 5101–5112. Available at: http://journals.sagepub.com/doi/10.1177/0954406219833719.

Pal, S. and Gauri, S.K. 2017. Optimization of multi-response dynamic systems integrating multiple regression and Taguchi's dynamic signal-to-noise ratio concept. *International Journal of Engineering, Science and Technology* 9(1), pp. 16–33. Available at: https://www.ajol.info/index.php/ijest/article/view/154706.

Pan, Z., Yu, W., Yi, X., Khan, A., Yuan, F. and Zheng, Y. 2019. Recent Progress on Generative Adversarial Networks (GANs): A Survey. *IEEE Access* 7, pp. 36322–36333. doi: 10.1109/ACCESS.2019.2905015.

Pan, Z., Zhang, Z., Meng, Z. and Wang, Y. 2023. A novel fault classification feature extraction method for rolling bearing based on multi-sensor fusion technology and EB-1D-TP encoding algorithm. *ISA Transactions* 142, pp. 427–444. Available at:

https://doi.org/10.1016/j.isatra.2023.07.015.

Pandey, G.K. and Srivastava, S. 2023. ResNet-18 comparative analysis of various activation functions for image classification. *6th International Conference on Inventive Computation Technologies, ICICT 2023 - Proceedings* (Icict), pp. 595–601. doi: 10.1109/ICICT57646.2023.10134464.

Pang, Y., Lin, J., Qin, T. and Chen, Z. 2022. Image-to-Image Translation: Methods and Applications. *IEEE Transactions on Multimedia* 24, pp. 3859–3881. doi: 10.1109/TMM.2021.3109419.

Pinedo-Sánchez, L.A., Mercado-Ravell, D.A. and Carballo-Monsivais, C.A. 2020. Vibration analysis in bearings for failure prevention using CNN. *Journal of the Brazilian Society of Mechanical Sciences and Engineering* 42, 628(12), pp. 1–16. doi: 10.1007/s40430-020-02711-w.

Prabha, D. 2022. A Survey on Alleviating the Naive Bayes Conditional Independence Assumption., pp. 2022–2025. doi: 10.1109/ICAISS55157.2022.10011103.

Qazi, E.U.H., Almorjan, A. and Zia, T. 2022. A One-Dimensional Convolutional Neural Network (1D-CNN) Based Deep Learning System for Network Intrusion Detection. *Applied Sciences (Switzerland)* 12(16), pp. 4–17. doi: 10.3390/app12167986.

Qiu, S., Cui, X., Ping, Z., Shan, N., Li, Z., Bao, X. and Xu, X. 2023. Deep Learning Techniques in Intelligent Fault Diagnosis and Prognosis for Industrial Systems: A Review. *Sensors* 23(3), p. 1305. Available at: https://www.mdpi.com/1424-8220/23/3/1305.

Radtke, M.-P., Huber, M. and Bock, J. 2023. Increasing Robustness of Data-Driven Fault Diagnostics with Knowledge Graphs. *Annual Conference of the PHM Society* 15(1), pp. 1–9. Available at: https://papers.phmsociety.org/index.php/phmconf/article/view/3552.

Rahmawan, H.A., Widjianto, B.L., Indrawati, K. and Ariefianto, R.M. 2023. Advancing Fault Diagnosis for Parallel Misalignment Detection in Induction Motors Based on Convolutional Neural Networks. *Jurnal EECCIS (Electrics, Electronics, Communications, Controls, Informatics, Systems)* 17(2), pp. 66–71. Available at:

https://jurnaleeccis.ub.ac.id/index.php/eeccis/article/view/1655.

Ramzan, F., Khan, M.U.G., Rehmat, A., Iqbal, S., Saba, T., Rehman, A. and Mehmood, Z. 2020. A Deep Learning Approach for Automated Diagnosis and Multi-Class Classification of Alzheimer's Disease Stages Using Resting-State fMRI and Residual Neural Networks. *Journal of Medical Systems* 44(2). doi: 10.1007/s10916-019-1475-2.

Rangel-Rodriguez, A.H., Granados-Lieberman, D., Amezquita-Sanchez, J.P., Bueno-Lopez, M. and Valtierra-Rodriguez, M. 2023. Analysis of Vibration Signals Based on Machine Learning for Crack Detection in a Low-Power Wind Turbine. *Entropy* 25(8), p. 1188. Available at: https://www.mdpi.com/1099-4300/25/8/1188.

Rathore, A. 2017. DEFECTS ANALYSIS AND OPTIMIZATION OF PROCESS PARAMETERS USING TAGUCHI DoE TECHNIQUE FOR SAND CASTING. *International Research Journal of Engineering and Technology (IRJET)* 4(8), pp. 432–437. Available at: https://irjet.net/archives/V4/i8/IRJET-V4I877.pdf.

Refinetti, M. and Goldt, S. 2022. The dynamics of representation learning in shallow, non-linear autoencoders. Available at: http://arxiv.org/abs/2201.02115.

Ren, Z., Tang, Y. and Zhang, W. 2021. Quality-related fault diagnosis based on k -nearest neighbor rule for non-linear industrial processes. *International Journal of Distributed Sensor Networks* 17(11), p. 155014772110559. Available at: http://journals.sagepub.com/doi/10.1177/15501477211055931.

Resendiz-Ochoa, E., Osornio-Rios, R.A., Benitez-Rangel, J.P., Romero-Troncoso, R.D.J. and Morales-Hernandez, L.A. 2018. Induction Motor Failure Analysis: An Automatic Methodology Based on Infrared Imaging. *IEEE Access* 6, pp. 76993–77003. doi: 10.1109/ACCESS.2018.2883988.

Reshadi, M., Zarft, N., Terheide, A. and Dick, S. 2023. Condition Monitoring and Fault Detection in Small Induction Motors Using Machine Learning Algorithms.

Salehi, A.W., Khan, S., Gupta, G., Alabduallah, B.I. and Almjally, A. 2023. Cnn1.Pdf.

Sauer, A., Chitta, K., Müller, J. and Geiger, A. 2021. Projected GANs Converge Faster. *Advances in Neural Information Processing Systems* 21(NeurIPS), pp. 17480–17492. Available at: http://arxiv.org/abs/2111.01007.

Sawant, Y.S. and Agashe, S.D. 2022. Fault Identification in Investment Casting Process Using Naive Bayes Method. *International Journal of Engineering Sciences* 15(1). Available at: https://www.scmrglobal.org/pdfs/Issue 15-1/ES150102 - Yatish sawant 9 - 16.pdf.

Sayyad, S., Kumar, S., Bongale, A., Kamat, P., Patil, S. and Kotecha, K. 2021. Data-Driven Remaining Useful Life Estimation for Milling Process: Sensors, Algorithms, Datasets, and Future Directions. *IEEE Access* 9, pp. 110255–110286. Available at: https://ieeexplore.ieee.org/document/9502093/.

Sewak, M., Sahay, S.K. and Rathore, H. 2020. An Overview of Deep Learning Architecture of Deep Neural Networks and Autoencoders. *Journal of Computational and Theoretical Nanoscience* 17(1), pp. 182–188. Available at: https://www.ingentaconnect.com/content/10.1166/jctn.2020.8648.

Shamsin, M., Krilova, N., Bazhanova, M., Kazantsev, V., Makarov, V.A. and Lobov, S. 2018. Supervised and unsupervised learning in processing myographic patterns. *Journal of Physics: Conference Series* 1117(1), p. 012008. Available at: https://iopscience.iop.org/article/10.1088/1742-6596/1117/1/012008.

Shao, H., Xia, M., Han, G., Zhang, Y. and Wan, J. 2021. Intelligent Fault Diagnosis of Rotor-Bearing System under Varying Working Conditions with Modified Transfer Convolutional Neural Network and Thermal Images. *IEEE Transactions on Industrial Informatics* 17(5), pp. 3488–3496. doi: 10.1109/TII.2020.3005965.

Shao, H., Li, W., Cai, B., Wan, J., Xiao, Y. and Yan, S., 2023. Dual-threshold attention-guided GAN and limited infrared thermal images for rotating machinery fault diagnosis under speed fluctuation. IEEE Transactions on Industrial Informatics, 19(9), pp.9933-9942.

Shao, S., Yan, R., Lu, Y., Wang, P. and Gao, R.X. 2020. DCNN-Based Multi-Signal Induction Motor Fault Diagnosis. *IEEE Transactions on Instrumentation and Measurement*

69(6), pp. 2658–2669. Available at: https://ieeexplore.ieee.org/document/8751989/.

Sheinker, A. and Moldwin, M.B. 2016. Magnetic anomaly detection (MAD) of ferromagnetic pipelines using principal component analysis (PCA). *Measurement Science and Technology* 27(4), p. 045104. Available at: https://iopscience.iop.org/article/10.1088/0957-0233/27/4/045104.

Shi, Z., Li, Y. and Liu, S. 2020. A review of fault diagnosis methods for rotating machinery. In: *2020 IEEE 16th International Conference on Control & Automation (ICCA)*. Singapore, 9–11 October 2020: IEEE, pp. 1618–1623. Available at: https://ieeexplore.ieee.org/document/9264309/.

Sihag, N. and Sangwan, K.S. 2020. A systematic literature review on machine tool energy consumption. *Journal of Cleaner Production* 275, p. 123125. Available at: https://doi.org/10.1016/j.jclepro.2020.123125.

Silik, A., Noori, M., Altabey, W.A., Ghiasi, R. and Wu, Z. 2021. Comparative analysis of wavelet transform for time-frequency analysis and transient localization in structural health monitoring. *SDHM Structural Durability and Health Monitoring* 15(1), pp. 1–22. doi: 10.32604/sdhm.2021.012751.

Singh, P.K. and Susan, S. 2023. Transfer Learning using Very Deep Pre-Trained Models for Food Image Classification. *2023 14th International Conference on Computing Communication and Networking Technologies, ICCCNT 2023* , pp. 1–6. doi: 10.1109/ICCCNT56998.2023.10307479.

Sinitsin, V., Ibryaeva, O., Sakovskaya, V. and Eremeeva, V. 2022. Intelligent bearing fault diagnosis method combining mixed input and hybrid CNN-MLP model. *Mechanical Systems and Signal Processing* 180(June), p. 109454. Available at: https://doi.org/10.1016/j.ymssp.2022.109454.

Skowron, M., Wolkiewicz, M. and Tarchała, G. 2020. Stator winding fault diagnosis of induction motor operating under the field-oriented control with convolutional neural networks. *Bulletin of the Polish Academy of Sciences Technical Sciences* 68(4), pp. 1039–

1048. Available at: https://journals.pan.pl/dlibra/publication/134660/edition/117693/content.

Souza, G.B., Santos, D.F.S., Pires, R.G., Marana, A.N. and Papa, J.P. 2017. Deep Boltzmann machines for robust fingerprint spoofing attack detection. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1863–1870. Available at: http://ieeexplore.ieee.org/document/7966077/.

Speck, C. and Bucci, D.J. 2018. Distributed UAV Swarm Formation Control via Object-Focused, Multi-Objective SARSA. *Proceedings of the American Control Conference* 2018-June, pp. 6596–6601. doi: 10.23919/ACC.2018.8430773.

Su, Y., Huang, J., Yao, H., Guan, L., Guo, M. and Zhong, Z. 2021. Multi-task Transient Contingency Screening with Temporal Graph Convolutional Network in Power Systems. *Journal of Physics: Conference Series* 2095(1). doi: 10.1088/1742-6596/2095/1/012027.

Sun, H. and Zhao, S. 2021. Fault Diagnosis for Bearing Based on 1DCNN and LSTM. Qin, Y. ed. *Shock and Vibration* 2021, pp. 1–17. Available at: https://www.hindawi.com/journals/sv/2021/1221462/.

Sun, M. 2020. A method for determining parameter weight early warning model based on reinforcement learning. *Computer Communications* 157(2018), pp. 417–422. Available at: https://doi.org/10.1016/j.comcom.2020.04.044.

Suresh, S. and Naidu, V.P.S. 2022. Mahalanobis-ANOVA criterion for optimum feature subset selection in multi-class planetary gear fault diagnosis. *JVC/Journal of Vibration and Control* 28(21–22), pp. 3257–3268. doi: 10.1177/10775463211029153.

Tangirala, S. 2020. Evaluating the Impact of GINI Index and Information Gain on Classification using Decision Tree Classifier Algorithm *. 11(2), pp. 612–619.

Taniguchi, S., Suzuki, M., Iwasawa, Y. and Matsuo, Y. 2023. End-to-end Training of Deep Boltzmann Machines by Unbiased Contrastive Divergence with Local Mode Initialization. Available at: http://arxiv.org/abs/2305.19684.

Taslimipoor, S., Rohanian, O. and Može, S. 2019. GCN-Sem at SemEval-2019 Task 1: Semantic Parsing using Graph Convolutional and Recurrent Neural Networks. In: *Proceedings of the 13th International Workshop on Semantic Evaluation*. Stroudsburg, PA, USA: Association for Computational Linguistics, pp. 102–106. Available at: https://www.aclweb.org/anthology/S19-2014.

Tepe, E. and Bilgin, G. 2022. Graph Neural Networks for Colorectal Histopathological Image Classification. In: *2022 Medical Technologies Congress (TIPTEKNO)*. IEEE, pp. 1–4. Available at: https://ieeexplore.ieee.org/document/9960184/.

Thaipisutikul, T., Chen, Y.C., Hui, L., Chen, S.C., Mongkolwat, P. and Shih, T.K. 2019. The matter of deep reinforcement learning towards practical ai applications. *Proceedings - 2019 12th International Conference on Ubi-Media Computing, Ubi-Media 2019* , pp. 24–29. doi: 10.1109/Ubi-Media.2019.00014.

Thalagala, S. and Walgampaya, C. 2021. Application of AlexNet convolutional neural network architecture-based transfer learning for automated recognition of casting surface defects. In: *2021 International Research Conference on Smart Computing and Systems Engineering (SCSE)*. IEEE, pp. 129–136. Available at: https://ieeexplore.ieee.org/document/9568315/.

Tian, B., Fan, X., Xu, Z., Wang, Z. and Du, H. 2022. Finite Element Simulation on Transformer Vibration Characteristics under Typical Mechanical Faults. In: *Proceedings of the 9th International Conference on Power Electronics Systems and Applications, (PESA 2022)*. Hong Kong, China, 20–22 September 2022: IEEE, pp. 1–4. doi: 10.1109/PESA55501.2022.10038342.

Toma, R.N., Prosvirin, A.E. and Kim, J.-M. 2020. Bearing Fault Diagnosis of Induction Motors Using a Genetic Algorithm and Machine Learning Classifiers. *Sensors* 20(7), p. 1884. Available at: https://www.mdpi.com/1424-8220/20/7/1884.

Toma, R.N., Toma, F.H. and Kim, J. 2021. Comparative Analysis of Continuous Wavelet Transforms on Vibration signal in Bearing Fault Diagnosis of Induction Motor. In: *2021 International Conference on Electronics, Communications and Information Technology*

(ICECIT). Khulna, Bangladesh, 14–16 September 2021: IEEE, pp. 1–4. Available at: https://ieeexplore.ieee.org/document/9641199/.

Toma, R.N., Piltan, F., Im, K., Shon, D., Yoon, T.H., Yoo, D. and Kim, J. 2022a. A Bearing Fault Classification Framework Based on Image Encoding Techniques and a Convolutional Neural Network under Different Operating Conditions. *Sensors* 22(13), p. 4881. Available at: https://www.mdpi.com/1424-8220/22/13/4881.

Toma, R.N., Gao, Y., Piltan, F., Im, K., Shon, D., Yoon, T.H., Yoo, D.-S. and Kim, J.-M. 2022b. Classification Framework of the Bearing Faults of an Induction Motor Using Wavelet Scattering Transform-Based Features. *Sensors* 22(22), p. 8958. Available at: https://www.mdpi.com/1424-8220/22/22/8958.

Tran, V.-L., Vo, T.-C. and Nguyen, T.-Q. 2024. One-dimensional convolutional neural network for damage detection of structures using time series data. *Asian Journal of Civil Engineering* 25(1), pp. 827–860. Available at: https://doi.org/10.1007/s42107-023-00816-w.

Vaibhaw, Sarraf, J. and Pattnaik, P.K. 2020. Brain–computer interfaces and their applications. In: *An Industrial IoT Approach for Pharmaceutical Industry Growth*. Elsevier, pp. 31–54. Available at: https://linkinghub.elsevier.com/retrieve/pii/B9780128213261000024.

Ve, I.V.T., Karpov, P., Theis, F. and Goos, G. 2019. *and Machine Learning – ICANN 2019 Lecture Notes in Computer Science*. Available at: http://dx.doi.org/10.1007/978-3-030-30484-3_22.

Wang, C.C., Kuo, P.H. and Chen, G.Y. 2022. Machine Learning Prediction of Turning Precision Using Optimized XGBoost Model. *Applied Sciences (Switzerland)* 12(15). doi: 10.3390/app12157739.

Wang, H., Yang, W., Ouyang, R., Hu, R., Li, K. and Li, K. 2023a. A Heterogeneous Parallel Computing Approach Optimizing SpTTM on CPU-GPU via GCN. *ACM Transactions on Parallel Computing* 10(2). doi: 10.1145/3584373.

Wang, J.-T. and Wang, C. 2019. High Performance WGAN-GP based Multiple-category

Network Anomaly Classification System. In: *2019 International Conference on Cyber Security for Emerging Technologies (CSET)*. Doha, Qatar: IEEE, pp. 1–7. Available at: https://ieeexplore.ieee.org/document/8904890/.

Wang, L. and Sng, D. 2015. Deep Learning Algorithms with Applications to Video Analytics for A Smart City: A Survey., pp. 1–8. Available at: http://arxiv.org/abs/1512.03131.

Wang, R., Zhang, S., Chen, Z. and Li, W. 2021a. Enhanced generative adversarial network for extremely imbalanced fault diagnosis of rotating machine. *Measurement: Journal of the International Measurement Confederation* 180(April), p. 109467. Available at: https://doi.org/10.1016/j.measurement.2021.109467.

Wang, S.H., Xing, S.B., Lei, Y.G., Lu, N. and Li, N.P. 2021b. Vibration indicator-based graph convolutional network for semi-supervised bearing fault diagnosis. *IOP Conference Series: Materials Science and Engineering* 1043(5), p. 052026. Available at: https://iopscience.iop.org/article/10.1088/1757-899X/1043/5/052026.

Wang, X., Li, A. and Han, G. 2023b. applied sciences A Deep-Learning-Based Fault Diagnosis Method of Industrial Bearings Using Multi-Source Information., pp. 1–26.

Wang, Z., She, Q. and Ward, T.E. 2021c. Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy. *ACM Computing Surveys* 54(2). doi: 10.1145/3439723.

Wang, Z. and Oates, T. 2015. Imaging time-series to improve classification and imputation. *IJCAI International Joint Conference on Artificial Intelligence* 2015-Janua, pp. 3939–3945.

Wei, H., Zhang, Q., Shang, M. and Gu, Y. 2021. Extreme learning Machine-based classifier for fault diagnosis of rotating Machinery using a residual network and continuous wavelet transform. 183(April). doi: 10.1016/j.measurement.2021.109864.

Wei, X., Yu, R. and Sun, J. 2020. View-GCN: View-Based Graph Convolutional Network for 3D Shape Analysis. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 1847–1856. Available at: https://ieeexplore.ieee.org/document/9156567/.

Wodecki, J. and Michalak, A. 2021. Fault-related impulsive component detection for vibration-based diagnostics in the presence of random impulsive noise. *IOP Conference Series: Earth and Environmental Science* 942(1). doi: 10.1088/1755-1315/942/1/012016.

Woodward, K., Kanjo, E. and Tsanas, A. 2024. Combining Deep Learning with Signal-image Encoding for Multi-Modal Mental Wellbeing Classification. *ACM Transactions on Computing for Healthcare* 5(1), pp. 1–23. Available at: https://dl.acm.org/doi/10.1145/3631618.

Wu, C., Yang, K., Ni, J., Lu, S., Yao, L. and Li, X. 2023a. Investigations for vibration and friction torque behaviors of thrust ball bearing with self-driven textured guiding surface. *Friction* 11(6), pp. 894–910.

Wu, G., Ji, X., Yang, G., Jia, Y. and Cao, C. 2023b. Signal-to-Image: Rolling Bearing Fault Diagnosis Using ResNet Family Deep-Learning Models. *Processes* 11(5). doi: 10.3390/pr11051527.

Wu, J., Kong, L., Kang, S., Zuo, H., Yang, Y. and Cheng, Z. 2024. Aircraft Engine Fault Diagnosis Model Based on 1DCNN-BiLSTM with CBAM. *Sensors* 24(3), p. 780. Available at: https://www.mdpi.com/1424-8220/24/3/780.

Wu, W., Cao, K., Li, C., Qian, C., Change, C. and Reed, L. 2019. TransGaGa : Geometry-Aware Unsupervised Image-to-Image Translation.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C. and Yu, P.S. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems* 32(1), pp. 4–24. Available at: https://ieeexplore.ieee.org/document/9046288/.

Xiang, X. and Foo, S. 2021. Recent Advances in Deep Reinforcement Learning Applications for Solving Partially Observable Markov Decision Processes (POMDP) Problems: Part 1—Fundamentals and Applications in Games, Robotics and Natural Language Processing. *Machine Learning and Knowledge Extraction* 3(3), pp. 554–581. doi: 10.3390/make3030029.

Xiao, D., Chen, Y. and Li, D.D. 2021a. One-dimensional deep learning architecture for fast

fluorescence lifetime imaging. *IEEE Journal of Selected Topics in Quantum Electronics* 27(4), pp. 1–10.

Xiao, R., Zhang, Z., Wu, Y., Jiang, P. and Deng, J. 2021b. Multi-scale information fusion model for feature extraction of converter transformer vibration signal. *Measurement: Journal of the International Measurement Confederation* 180(May), p. 109555. Available at: https://doi.org/10.1016/j.measurement.2021.109555.

Xie, F., Li, G., Fan, Q., Xiao, Q. and Zhou, S. 2023. Optimizing and Analyzing Performance of Motor Fault Diagnosis Algorithms for Autonomous Vehicles via Cross-Domain Data Fusion. *Processes* 11(10), p. 2862. Available at: https://www.mdpi.com/2227-9717/11/10/2862.

Xu, S., Cui, Y., Yang, C., Wei, S., Dong, W., Huang, L., Liu, C., Ren, Z. and Wang, W. 2021. The fuzzy comprehensive evaluation (FCE) and the principal component analysis (PCA) model simulation and its applications in water quality assessment of Nansi lake basin, China. *Environmental Engineering Research* 26(2), pp. 0–2. doi: 10.4491/eer.2020.022.

Xue, J., Hu, X., Chen, H. and Zhou, G. 2022. Research on LSTM-XGBoost Integrated Model of Photovoltaic Power Forecasting System. In: *2022 14th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*. IEEE, pp. 22–25. Available at: https://ieeexplore.ieee.org/document/9903235/.

Yang, D., Karimi, H.R. and Gelman, L. 2022. A Fuzzy Fusion Rotating Machinery Fault Diagnosis Framework Based on the Enhancement Deep Convolutional Neural Networks., pp. 1–15.

Yang, F., Song, M., Ma, X., Guo, N. and Xue, Y. 2023a. Research on H7006C Angular Contact Ball Bearing Slipping Behavior under Operating Conditions. *Lubricants* 11, 298(7), pp. 1–14. doi: 10.3390/lubricants11070298.

Yang, F. 2024. Learning-Based Hierarchical Decision-Making Framework for Automatic Driving in Incompletely Connected Traffic Scenarios.

Yang, Q., Tang, B., Shen, Y. and Li, Q. 2023b. Self-Attention Parallel Fusion Network for

Wind Turbine Gearboxes Fault Diagnosis. *IEEE Sensors Journal* 23(19), pp. 23210–23220. Available at: https://ieeexplore.ieee.org/document/10236986/.

Yang, S., Zhu, H., Pang, S., Ruan, Z., Lin, S., Ding, Y., Cao, P. and Shen, Z. 2023c. Preparation a High-Performance of Gangue-Based Geopolymer Backfill Material: Recipes Optimization Using the Taguchi Method. *Materials* 16(15), p. 5360. Available at: https://www.mdpi.com/1996-1944/16/15/5360.

Ye, L., Ma, X. and Wen, C. 2021. Rotating machinery fault diagnosis method by combining time-frequency domain features and cnn knowledge transfer. *Sensors* 21, 8168(24). doi: 10.3390/s21248168.

Ye, Q., Liu, S. and Liu, C. 2020. A Deep Learning Model for Fault Diagnosis with a Deep Neural Network and Feature Fusion on Multi-Channel Sensory Signals. *Sensors* 20(15), p. 4300. Available at: https://www.mdpi.com/1424-8220/20/15/4300.

Ye, Z. and Yu, J. 2022. Multi-level features fusion network-based feature learning for machinery fault diagnosis. *Applied Soft Computing* 122, p. 108900. Available at: https://doi.org/10.1016/j.asoc.2022.108900.

Yingfan, L., Hong, C. and Jiangtao, C. 2021. Revisiting k-Nearest Neighbor Graph Construction on High-Dimensional Data : Experiments and Analyses., pp. 1–16. Available at: http://arxiv.org/abs/2112.02234.

You, Z., Wang, X. and Xu, B. 2013. Investigation of deep Boltzmann machines for phone recognition. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, pp. 7600–7603. Available at: http://ieeexplore.ieee.org/document/6639141/.

Yousuf, M., Alsuwian, T., Amin, A.A., Fareed, S. and Hamza, M. 2024. IoT-based health monitoring and fault detection of industrial AC induction motor for efficient predictive maintenance. *Measurement and Control* . Available at: http://journals.sagepub.com/doi/10.1177/00202940241231473.

Yu, X., Fan, Z., Jamil, M., Aziz, M.Z., Hou, Y., Li, H. and Lv, J. 2021. Transacting Multiple

Mother Wavelets in Continuous Wavelet Transform for Epilepsy EEG Classification via CNN. *2021 IEEE 9th International Conference on Information, Communication and Networks, ICICN 2021* (January 2022), pp. 76–80. doi: 10.1109/ICICN52636.2021.9673990.

Yuan, L., Lian, D., Kang, X., Chen, Y. and Zhai, K. 2020. Rolling Bearing Fault Diagnosis Based on Convolutional Neural Network and Support Vector Machine. *IEEE Access* 8, pp. 137395–137406. doi: 10.1109/ACCESS.2020.3012053.

Zambra, M., Testolin, A. and Zorzi, M. 2023. A Developmental Approach for Training Deep Belief Networks. *Cognitive Computation* 15(1), pp. 103–120. Available at: http://arxiv.org/abs/2207.05473.

Zeng, M., Wang, H., Cheng, Y. and Wei, J. 2024. A compound fault diagnosis model for gearboxes using correlation information between single faults. *Measurement Science and Technology* 35(3), p. 036202. Available at: https://iopscience.iop.org/article/10.1088/1361-6501/ad1312.

Zhang, B., Pang, X., Zhao, P. and Lu, K. 2023a. A New Method Based on Encoding Data Probability Density and Convolutional Neural Network for Rotating Machinery Fault Diagnosis. *IEEE Access* 11(February), pp. 26099–26113. doi: 10.1109/ACCESS.2023.3257041.

Zhang, C., Geng, T., Guo, A., Tian, J., Herbordt, M., Li, A. and Tao, D. 2022a. H-GCN: A Graph Convolutional Network Accelerator on Versal ACAP Architecture. In: *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, pp. 200–208. Available at: https://ieeexplore.ieee.org/document/10035160/.

Zhang, H., Borghesani, P., Randall, R.B. and Peng, Z. 2022b. A benchmark of measurement approaches to track the natural evolution of spall severity in rolling element bearings. *Mechanical Systems and Signal Processing* 166(September 2021), p. 108466. Available at: https://doi.org/10.1016/j.ymssp.2021.108466.

Zhang, H., Zhang, S., Qiu, L., Zhang, Y., Wang, Y., Wang, Z. and Yang, G. 2022c. A

remaining useful life prediction method based on time–frequency images of the mechanical vibration signals. *Scientific Reports* 12, 17887(1). doi: 10.1038/s41598-022-22941-3.

Zhang, R., Song, Y., Chen, P., Li, X., Chen, J. and Liu, Z. 2023b. A Novel Multi-Scale Neural Transformation Graph Method for Micro-Service System Fault Multi-Classification. In: *2023 2nd International Conference on Machine Learning, Control, and Robotics (MLCR)*. IEEE, pp. 7–12. Available at: https://ieeexplore.ieee.org/document/10475450/.

Zhang, S., Zhang, S., Wang, B. and Habetler, T.G. 2020. Deep Learning Algorithms for Bearing Fault Diagnostics—A Comprehensive Review. *IEEE Access* 8, pp. 29857–29881. Available at: https://ieeexplore.ieee.org/document/8988271/.

Zhang, S., Wei, H.L. and Ding, J. 2023c. An effective zero-shot learning approach for intelligent fault detection using 1D CNN. *Applied Intelligence* 53(12), pp. 16041–16058. doi: 10.1007/s10489-022-04342-1.

Zhang, X., Zhao, B. and Lin, Y. 2021a. Machine Learning Based Bearing Fault Diagnosis Using the Case Western Reserve University Data: A Review. *IEEE Access* 9, pp. 155598–155608. doi: 10.1109/ACCESS.2021.3128669.

Zhang, X., Wang, J., Wu, L., Meng, F., Wang, L. and Liu, Z. 2022d. Wavelet sparsity enhancement for extracting transient vibration signatures of bearing structural damages. *Structural Health Monitoring* , p. 147592172211171. Available at: http://journals.sagepub.com/doi/10.1177/14759217221117101.

Zhang, Y., Ren, H., Ye, J., Gao, X., Wang, Y., Ye, K. and Xu, C.-Z. 2021b. AOAM: Automatic Optimization of Adjacency Matrix for Graph Convolutional Network. In: *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, pp. 5130–5136. Available at: https://ieeexplore.ieee.org/document/9412046/.

Zhang, Y., Guo, H., Zhou, Y., Xu, C. and Liao, Y. 2023d. Recognising drivers' mental fatigue based on EEG multi-dimensional feature selection and fusion. *Biomedical Signal Processing and Control* 79(P2), p. 104237. Available at: https://doi.org/10.1016/j.bspc.2022.104237.

Zheng, T., Li, J., Tian, H. and Wu, Q. 2023. The Process Analysis Method of SAR Target Recognition in Pre-Trained CNN Models. *Sensors* 23(14). doi: 10.3390/s23146461.

Zhong, H., Yu, S., Trinh, H., Lv, Y., Yuan, R. and Wang, Y. 2023. Fine-tuning Transfer Learning based on DCGAN Integrated with Self-attention and Spectral Normalization for Bearing Fault Diagnosis. *Measurement* 210(January), p. 112421. Available at: https://doi.org/10.1016/j.measurement.2022.112421.

Zhou, Y., Long, X., Sun, M. and Chen, Z. 2022. Bearing fault diagnosis based on Gramian angular field and DenseNet. *Mathematical Biosciences and Engineering* 19(12), pp. 14086–14101. doi: 10.3934/mbe.2022656.

Zhou, Z., Li, Z., Cai, Z. and Wang, P. 2019. Fault Identification Using Fast k-Nearest Neighbor Reconstruction. *Processes* 7(6), p. 340. Available at: https://www.mdpi.com/2227-9717/7/6/340.

Zhu, B., Dang, M. and Grover, A. 2023a. Scaling Pareto-Efficient Decision Making Via Offline Multi-Objective RL., pp. 1–18. Available at: http://arxiv.org/abs/2305.00567.

Zhu, Z., Lei, Y., Qi, G., Chai, Y., Mazur, N., An, Y. and Huang, X. 2023b. A review of the application of deep learning in intelligent fault diagnosis of rotating machinery. *Measurement* 206(June 2022), p. 112346. Available at: https://doi.org/10.1016/j.measurement.2022.112346.

Appendix 1: Pythons Codes in Jupyter Notebook

1.1. Chapter3: Thermal Images Extraction from Cardiff University File

```
import pandas as pd
  import numpy as np
  import os
  from shutil import copyfile
  data_dir = r'C:\Users\Shahd\OneDrive - Cardiff University\Cardiff\Phd\Year2\RawMotorData/'
  new_dir = r'C:\Users\Shahd\OneDrive - Cardiff University\Cardiff\Phd\Year2\MotorData26/'
  for adir in os.listdir(data_dir):
   newadir = new_dir + adir
   print(f"Processing {adir}...")
   if not os.path.exists(newadir):
    os.mkdir(newadir)
   if not adir.startswith('.'):
    bdir = data_dir + f'{adir}/'
    for cdir in os.listdir(bdir):
     if not cdir.startswith('.'):
      ddir = bdir + f'{cdir}/'
      for afile in os.listdir(ddir):
       if afile.endswith('.csv') and 'vibration' in afile:
        filepath = ddir + afile
        data = pd.read_csv(filepath)

        vib_data = data.loc[0:7000000:26, :]
        print(vib_data)
        vib_list = np.array_split(vib_data[' vibration'], 120)
        for i, v in enumerate(vib_list):

          vib_df = pd.DataFrame(data={"vibration": v})
          vib_df.to_csv(newadir + f"/{adir}_{cdir}_vibration_{i}.csv", index=False)

       if afile.endswith('.csv') and 'current' in afile:
        filepath = ddir + afile
        data2= pd.read_csv(filepath)
        cur_data = data2.loc[0:7000000:26, :]
```

```
        cur_list = np.array_split(cur_data[' phase1'], 120)

        for i, c in enumerate(cur_list):

            cur_df = pd.DataFrame(data={"current1": c})
            cur_df.to_csv(newadir + f"/{adir}_{cdir}_current_{i}.csv", index=False)

      if afile.endswith('.png'):
        filepath = ddir + afile
        new_path = newadir + f"/{adir}_{cdir}_{afile}"
        copyfile(filepath, new_path)
```

1.2. Chapter3: Thermal Images Preprocessing (Jupyter Notebook)

```
yyfrom pathlib import Path
import cv2
import os
import numpy as np
import random
from shutil import copyfile
pip install pillow
thermal_path = r'C:\Users\Shahd\OneDrive - Cardiff
University\Cardiff\Phd\Year2\MotorData26\Thermal_Resized_split'
newadir = r'C:\Users\Shahd\OneDrive - Cardiff
University\Cardiff\Phd\Year2\MotorData26\median_thermal/'
#brightness
def brightness(img, low, high):
  value = random.uniform(low, high)
  hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
  hsv = np.array(hsv, dtype = np.float64)
  hsv[:,:,1] = hsv[:,:,1]*value
  hsv[:,:,1][hsv[:,:,1]>255] = 255
  hsv[:,:,2] = hsv[:,:,2]*value
  hsv[:,:,2][hsv[:,:,2]>255] = 255
```

```
  hsv = np.array(hsv, dtype = np.uint8)
  img = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)
  return img
#rotation
def rotation(img, angle):
  angle = int(random.uniform(-angle, angle))
  h, w = img.shape[:2]
  M = cv2.getRotationMatrix2D((int(w/2), int(h/2)), angle, 1)
  img = cv2.warpAffine(img, M, (w, h))
  return img
#zoom
import cv2
import random
img = cv2.imread('arc_de_triomphe.jpg')
def fill(img, h, w):
  img = cv2.resize(img, (h, w), cv2.INTER_CUBIC)
  return img
def zoom(img, value):
  if value > 1 or value < 0:
    print('Value for zoom should be less than 1 and greater than 0')
    return img
  value = random.uniform(value, 1)
  h, w = img.shape[:2]
  h_taken = int(value*h)
  w_taken = int(value*w)
  h_start = random.randint(0, h-h_taken)
  w_start = random.randint(0, w-w_taken)
  img = img[h_start:h_start+h_taken, w_start:w_start+w_taken, :]
  img = fill(img, h, w)
  return img
for adir in os.listdir(thermal_path):
  newdir = newadir + adir

  print(f"Processing {adir}...")
```

```
###
  if not os.path.exists(newdir):
    os.mkdir(newdir)
  if not adir.startswith('.'):
    bdir = thermal_path + f'/{adir}/'
    print(bdir)
    bnewdir = newadir + f'/{adir}/'
    print(bnewdir)
##
    for cdir in os.listdir(bdir):
      cnewdir = bnewdir + cdir

      if not os.path.exists(cnewdir):
        os.mkdir(cnewdir) #print faults type
      if not cdir.startswith('.'):
        ddir = bdir + f'/{cdir}/'

        for afile in os.listdir(ddir):

            v_path = Path(afile)
            img1 = ddir + afile
            img1= cv2.imread(img1)
            median = cv2.medianBlur(img1,9 )
            median = brightness(median, 1, 2)
            median= rotation(median, 40)
            median = zoom(median, 0.4 )
            #cv2.imshow("median", median)
            filepath = ddir + afile
            new_path = cnewdir + f"/{v_path}"
            cv2.imwrite(new_path, median)
            cv2.waitKey(0)
            cv2.destroyAllWindows()
```

1.3. Chapter 3: Preprocessing For Image Fusion

```
for index in data2.index:
#Image new directory is merge_GADF_GADF_thermal
#dirname = os.path.abspath(GADF_GADF_Thermal)
#dirname= r'C:\Users\Shahd\OneDrive - Cardiff
University\Cardiff\Phd\Year2\MotorData26\GADF_GADF_Thermal'
      #assign columns using index
  c= data2['current'][index]
  v= data2['vibration'][index]
  t= data2['thermal'][index]
  fault = data2['Fault'][index]
  dest_directory = merge_GADF_GADF_thermal + f"/{fault}/"
  if not os.path.exists(dest_directory):
    os.mkdir(dest_directory)
  #reading pictures
  c_im=cv2.imread(c)
  v_im=cv2.imread(v)
  t_im=cv2.imread(t)
     #split channels for each feature
  b1, g1, r1 = cv2.split(c_im)
  b2, g2, r2 = cv2.split(v_im)
  b3, g3, r3 = cv2.split(t_im)

     #merge channels
  blue_merge = cv2.merge((b1,b2,b3))
  green_merge = cv2.merge((g1,g2,g3))
  red_merge = cv2.merge((r1,r2,r3))

     #total image
  total = blue_merge+green_merge+red_merge
     # Using cv2.imwrite() method Saving the image
  cv2.imwrite(os.path.join(dest_directory, f'{fault}_GADF_GADF_therm_{index}'+'.JPEG'),
cv2.cvtColor(total, cv2.COLOR_BGR2RGB))
```

Appendix 2: MATLAB Code

2.1 Chapter 3: MFPT Files Spitting, CSV. Files Creation, Scalograms Images (CWT images) Creation and Saving as JPG.

Scalogram of Bearing Data Visualisation

```
data_inner = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data', 'InnerRaceFault_vload_1.mat'));
% Plot bearing signal and scalogram
plotBearingSignalAndScalogram(data_inner)
% Import data with ORF
data_outer = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'test_data', 'OuterRaceFault_3.mat'));
% Plot original signal and its scalogram
plotBearingSignalAndScalogram(data_outer)

% Import normal bearing data
data_normal = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data', 'baseline_1.mat'));
% Plot original signal and its scalogram
plotBearingSignalAndScalogram(data_normal)
```

Prepare Training Data

Unzip the downloaded file.

```
if exist('RollingElementBearingFaultDiagnosis-Data-master.zip', 'file')
  unzip('RollingElementBearingFaultDiagnosis-Data-master.zip')
end
fileLocation =
'C:\Users\Shahd\Documents\MATLAB\Examples\R2022a\predmaint_deeplearning\Rol
lingElementBearingFaultDiagnosisUsingDeepLearningExample\RollingElementBearin
gFaultDiagnosis-Data-master\train_data'
ensembleTrain = fileEnsembleDatastore(fileLocation, fileExtension)
```

```
ensembleTrain.ReadFcn = @readMFPTBearing;
ensembleTrain.DataVariables = ["gs", "sr", "rate", "load", "BPFO", "BPFI", "FTF",
"BSF"];
ensembleTrain.ConditionVariables = ["Label", "FileName"];
ensembleTrain.SelectedVariables = ["gs", "sr", "rate", "load", "BPFO", "BPFI", "FTF",
"BSF", "Label", "FileName"]
% Wavelet scalograms creation
reset(ensembleTrain)
while hasdata(ensembleTrain)
 folderName = 'train_image';
 convertSignalToScalogram(ensembleTrain,folderName);
end
% Create image datastore to store all training images
path = fullfile('.', folderName);
imds = imageDatastore(path, ...
 'IncludeSubfolders',true,'LabelSource','foldernames');
% Use 20% training data as validation set
[imdsTrain,imdsValidation] = splitEachLabel(imds,0.8,'randomize');
```

Helper Functions

```
function plotBearingSignalAndScalogram(data)
% Convert 1-D bearing signals to scalograms through wavelet transform
fs = data.bearing.sr;
t_total = 0.1; % seconds
n = round(t_total*fs);
bearing = data.bearing.gs(1:n);
[cfs,frq] = cwt(bearing,'amor', fs);
% Plot the original signal and its scalogram
figure
subplot(2,1,1)
plot(0:1/fs:(n-1)/fs,bearing)
xlim([0,0.1])
```

```
title('Vibration Signal')
xlabel('Time (s)')
ylabel('Amplitude')
subplot(2,1,2)
surface(0:1/fs:(n-1)/fs,frq,abs(cfs))
shading flat
xlim([0,0.1])
ylim([0,max(frq)])
title('Scalogram')
xlabel('Time (s)')
ylabel('Frequency (Hz)')
end
function convertSignalToScalogram(ensemble,folderName)
% Convert 1-D signals to scalograms and save scalograms as images
data = read(ensemble);
fs = data.sr
x = data.gs{:};
label = char(data.Label);
fname = char(data.FileName);
ratio = 5000/97656;
interval = ratio*fs
N = floor(numel(x)/interval);
% Create folder to save images
path = fullfile('.',folderName,label);
if ~exist(path,'dir')
 mkdir(path);
end
%new added
path_numerical = fullfile('.',folderName,label, 'sub_sample_split1');
if ~exist(path_numerical,'dir')
 mkdir(path_numerical);
```

```
end
for idx = 1:N
 sig = envelope(x(interval*(idx-1)+1:interval*idx));
 file_path = fullfile('.',path_numerical,[fname '-' num2str(idx) '.csv']);%(new added)
 writematrix(sig,file_path); %new
 %like we are creating subfolders based on the interval equal data ineach
 %file not using envelop transform
 cfs = cwt(sig,'amor', seconds(1/fs));
 cfs = abs(cfs);
 img = ind2rgb(round(rescale(flip(cfs),0,255)),jet(320));
 outfname = fullfile('.',path,[fname '-' num2str(idx) '.jpg']);
 imwrite(imresize(img,[224,224]),outfname);
end
end
```

2.2 Chapter 5: Phase 1: Step1: Data Preprocessing and General Load-Dependent Feature Extraction, and Phase 2: Step 3: CLAF

Normal Dataset

```
% Import normal bearing data
data_normal = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'test_data', 'baseline_2.mat'));
```

Inner load comparison

```
IRF_50 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_inner_load', 'InnerRaceFault_vload_2.mat'));
% Plot bearing signal and scalogram
plotBearingSignalAndScalogram(IRF_50)
IRF_100 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_inner_load', 'InnerRaceFault_vload_3.mat'));
% Plot bearing signal and scalogram
plotBearingSignalAndScalogram(IRF_100)
IRF_150 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
```

```
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_inner_load', 'InnerRaceFault_vload_4.mat'));
% Plot bearing signal and scalogram
plotBearingSignalAndScalogram(IRF_150)

IRF_200 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_inner_load', 'InnerRaceFault_vload_5.mat'));
% Plot bearing signal and scalogram
plotBearingSignalAndScalogram(IRF_200)
IRF_250 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_inner_load', 'InnerRaceFault_vload_6.mat'));
% Plot bearing signal and scalogram
plotBearingSignalAndScalogram(IRF_250)
IRF_300 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_inner_load', 'InnerRaceFault_vload_7.mat'));
% Plot bearing signal and scalogram
plotBearingSignalAndScalogram(IRF_300)
```

Create Inner Load ensemble

The code calculates the minimum length of the bearing signal data across all load factors before creating the ensemble. The subsequent extraction of the bearing signal data is limited to this minimum length, ensuring that all ensembles have the same number of entries because the baseline is 6 seconds while others are only 3 seconds; now they are the same.

```
% Load dataset of different Inner load factors and corresponding variable names
load_factors = [50, 100, 150, 200, 250, 270, 300];
variable_names = {'IRF_50', 'IRF_100', 'IRF_150', 'IRF_200', 'IRF_250',
'data_normal','IRF_300'};
% Initialize an empty ensemble with two columns
ensemble = table();
% Find the minimum length of bearing signal data across all load factors
min_length = Inf; % Initialize with infinity
for i = 1:length(load_factors)
```

```
  % Construct the variable name
  variable_name = variable_names{i};

  if strcmp(variable_name, 'data_normal')
    % Extract the desired portion of the bearing signal from the loaded dataset
    fs = data_normal.bearing.sr;
    t_total = 3; % seconds
    n = round(t_total * fs);
    bearing_data = data_normal.bearing.gs(1:n);
  else
    % Extract the desired portion of the bearing signal from the loaded dataset
    fs = eval([variable_name '.bearing.sr']);
    t_total = 3; % seconds
    n = round(t_total * fs);
    bearing_data = eval([variable_name '.bearing.gs(1:n)']);
  end
  % Update the minimum length
  min_length = min(min_length, length(bearing_data));
end

for i = 1:length(load_factors)
  % Construct the variable name
  variable_name = variable_names{i};
    if strcmp(variable_name, 'data_normal')
    % Extract the desired portion of the bearing signal from the loaded dataset
    fs = data_normal.bearing.sr;
    bearing_data = data_normal.bearing.gs(1:min_length);
  else
    % Extract the desired portion of the bearing signal from the loaded dataset
    fs = eval([variable_name '.bearing.sr']);
    bearing_data = eval([variable_name '.bearing.gs(1:min_length)']);
  end

  % Create a timetable for the current variable
```

```
  timetable_data = timetable(seconds(0:1/fs:(length(bearing_data)-1)/fs)', bearing_data);
  % Add the timetable and load factor as rows to the ensemble
  ensemble = [ensemble; {timetable_data, load_factors(i)}];
end
% Rename the columns of the ensemble
ensemble.Properties.VariableNames = {'Timetable', 'LoadFactor'};
% Display the resulting ensemble
disp(ensemble);
%use the table in diagnosticFeatureDesigner the results are in the word
%file then generate features after that do the ranking
Features Ranking
% Load the provided table load (INNERONLY_feature extraction_Phase1_thesis)
% Exclude the 'LoadFactor' and 'BandPower' columns
features = FeatureTable1_1(:, 2:end-1);
% Calculate the standard deviation for each feature
stdValues = varfun(@std, features);
% Calculate the range for each feature
rangeValues = varfun(@(x) max(x) - min(x), features);

% Get the feature names
featureNames = cellfun(@(x) x(strfind(x, '/')+1:end), features.Properties.VariableNames,
'UniformOutput', false);
% Create a ranking table with feature names, std, and range
rankingTable = table(featureNames', stdValues.Variables', rangeValues.Variables',
'VariableNames', {'Feature', 'Std', 'Range'});
% Sort the ranking table based on the desired metric (e.g., std or range)
sortedTable = sortrows(rankingTable, 'Std', 'descend'); % Change 'Std' to 'Range' if you want to
rank by range
% Display the sorted table
disp(sortedTable);
%use the table in diagnosticFeatureDesigner the results are in the word
%file then generate features after that do the ranking
diagnosticFeatureDesigner
%saved in
```

```
%C:\Users\Shahd\Documents\MATLAB\Examples\R2023a\predmaint\RollingElementBearing
FaultDiagnosisExample\enessemble_inner
%enessemble as variable (enessemble_inner_Full) in the same extenstion
Use CWT "Amor" in finding the energy of each load and proof the difference
% Define the wavelet type and parameters
wavelet_type = 'amor';
num_scales = 5;
% Initialize empty cell arrays to store load factors and energies
load_factors = ensemble.LoadFactor;
energies = cell(length(load_factors), 1);
% Calculate wavelet energy for each load
for i = 1:length(load_factors)
  % Extract the timetable data for the current load
  timetable_data = ensemble.Timetable{i};
  % Get the vibration signal from the timetable data
  vibration_signal = timetable_data.Variables;
  % Adjust the length of the vibration signal for load 270 if necessary
  if strcmp(load_factors(i), 270)
    target_length = length(vibration_signal) - (length(vibration_signal) -
length(ensemble.Timetable{1}.Variables));
    vibration_signal = vibration_signal(1:target_length);
  end
  % Perform wavelet transform
  [cfs, ~] = cwt(vibration_signal, wavelet_type, num_scales);

  % Calculate wavelet energy as the squared absolute values of coefficients
  wse = sum(abs(cfs).^2, 1);
  % Store the energy
  energies{i} = wse;
end
% Create separate tables for each load factor
tables = cell(length(load_factors), 1);
for i = 1:length(load_factors)
```

```
  tables{i} = table(repmat(load_factors(i), size(energies{i})), energies{i}, 'VariableNames',
{'LoadFactor', 'WaveletEnergy'});
end
% Concatenate the tables into a single table
energy_table = vertcat(tables{:});
% Display the energy table
disp(energy_table);
% Sort the energy_table based on LoadFactor
energy_table_sorted = sortrows(energy_table, 'LoadFactor');
% Find unique load factors and create group indices
[unique_load_factors, ~, load_factor_group_indices] =
unique(energy_table_sorted.LoadFactor);
% Calculate mean energy using loop and if condition
mean_energy = NaN(size(unique_load_factors)); % Initialize mean energy vector
for i = 1:length(unique_load_factors)
  idx = (load_factor_group_indices == i);
  mean_energy(i) = mean(energy_table_sorted.WaveletEnergy(idx));
end
% Create a new table to store the results
mean_energy_table_inner = table(unique_load_factors, mean_energy, 'VariableNames',
{'LoadFactor', 'MeanEnergy'});
% Display the mean energy table
disp(mean_energy_table_inner);
% Extract data for the normal load (270)
data_normal = energy_table.WaveletEnergy(energy_table.LoadFactor == 270);
% Loop through each load and perform the t-test
for load = unique(energy_table.LoadFactor)'
  if load ~= 270
    % Extract data for the current load
    data_current = energy_table.WaveletEnergy(energy_table.LoadFactor == load);
    % Perform the t-test
    [h, p, ci, stats] = ttest2(data_normal, data_current);
    % Display results
    fprintf('\nLoad Factor %d vs. Normal (270):\n', load);
```

```
    fprintf('p-value: %f\n', p);
    % Check for significance
    if h
      fprintf('Significant difference\n');
    else
      fprintf('No significant difference\n');
    end

    % Display confidence intervals if needed
    fprintf('Confidence Interval: [%f, %f]\n', ci(1), ci(2));

    % Display t-test statistics if needed
    disp(stats);
  end
end
Significance
% Initialize variables to store results
loadFactors = unique(energy_table.LoadFactor)';
numLoads = numel(loadFactors);
meanValues = zeros(numLoads, 1);
stdValues = zeros(numLoads, 1);
seValues = zeros(numLoads, 1);
meanDiffValues = zeros(numLoads, 1);
ciValues = zeros(numLoads, 2);
tValues = zeros(numLoads, 1);
dfValues = zeros(numLoads, 1);
pValues = zeros(numLoads, 1);
% Extract data for the normal load (270)
data_normal = energy_table.WaveletEnergy(energy_table.LoadFactor == 270);

% Loop through each load and perform the t-test
for i = 1:numLoads
  load = loadFactors(i);
```

```
  if load ~= 270
    % Extract data for the current load
    data_current = energy_table.WaveletEnergy(energy_table.LoadFactor == load);
    % Perform the t-test
    [h, p, ci, stats] = ttest2(data_normal, data_current);
    % Store results
    meanValues(i) = mean(data_current);
    stdValues(i) = std(data_current);
    seValues(i) = std(data_current) / sqrt(length(data_current));
    meanDiffValues(i) = mean(data_current) - mean(data_normal);
    ciValues(i, :) = ci;
    tValues(i) = stats.tstat;
    dfValues(i) = stats.df;
    pValues(i) = p;
    % Display results
    fprintf('\nLoad Factor %d vs. Normal (270):\n', load);
    fprintf('Number of Samples: %d\n', length(data_current));
    fprintf('Mean: %f\n', meanValues(i));
    fprintf('Standard Deviation: %f\n', stdValues(i));
    fprintf('Standard Error of the Mean: %f\n', seValues(i));
    fprintf('Mean Difference: %f\n', meanDiffValues(i));
    fprintf('95%% Confidence Interval: [%f, %f]\n', ciValues(i, 1), ciValues(i, 2));
    fprintf('t-test Value: %f\n', tValues(i));
    fprintf('Degrees of Freedom: %f\n', dfValues(i));
    fprintf('p-value: %f\n', pValues(i));
    % Check for significance
    if h
      fprintf('Significant difference\n');
    else
      fprintf('No significant difference\n');
    end
  end
end
% Initialize variables to store results
```

```
loadFactors = unique(energy_table.LoadFactor)';
numLoads = numel(loadFactors);
resultsTable = table('Size', [numLoads, 11], 'VariableTypes', {'double', 'double', 'double',
'double', 'double', 'double', 'double', 'double', 'double', 'double', 'logical'}, 'VariableNames',
{'LoadFactor', 'NumSamples', 'Mean', 'StdDev', 'SEMean', 'MeanDiff', 'CI_Lower', 'CI_Upper',
'tValue', 'DF', 'Significant'});
% Extract data for the normal load (270)
data_normal = energy_table.WaveletEnergy(energy_table.LoadFactor == 270);
% Loop through each load and perform the t-test
for i = 1:numLoads
  load = loadFactors(i);

  if load ~= 270
    % Extract data for the current load
    data_current = energy_table.WaveletEnergy(energy_table.LoadFactor == load);
    % Perform the t-test
    [h, ~, ci, stats] = ttest2(data_normal, data_current);
    % Store results in the table
    resultsTable(i, :) = {load, length(data_current), mean(data_current), std(data_current),
std(data_current) / sqrt(length(data_current)), mean(data_current) - mean(data_normal), ci(1),
ci(2), stats.tstat, stats.df, h};
  end
end
% Display the results table
disp(resultsTable);
% Initialize variables to store results
loadFactors = unique(energy_table.LoadFactor)';
numLoads = numel(loadFactors);
resultsTable = table('Size', [numLoads, 12], 'VariableTypes', {'double', 'double', 'double',
'double', 'double', 'double', 'double', 'double', 'double', 'double', 'double', 'logical'},
'VariableNames', {'LoadFactor', 'NumSamples', 'Mean', 'StdDev', 'SEMean', 'MeanDiff',
'CI_Lower', 'CI_Upper', 'tValue', 'DF', 'pValue', 'Significant'});
% Extract data for the normal load (270)
data_normal = energy_table.WaveletEnergy(energy_table.LoadFactor == 270);
```

```
% Loop through each load and perform the t-test
for i = 1:numLoads
  load = loadFactors(i);

  if load ~= 270
    % Extract data for the current load
    data_current = energy_table.WaveletEnergy(energy_table.LoadFactor == load);

    % Perform the t-test
    [h, p, ci, stats] = ttest2(data_normal, data_current)

    % Store results in the table
    resultsTable(i, :) = {load, length(data_current), mean(data_current), std(data_current),
std(data_current) / sqrt(length(data_current)), mean(data_current) - mean(data_normal), ci(1),
ci(2), stats.tstat, stats.df, p, h};
  end
end
% Display the results table
disp(resultsTable);
% Bar plot of mean energy
figure;
bar(mean_energy_table_inner.LoadFactor, mean_energy_table_inner.MeanEnergy);
xlabel('Load Factor');
ylabel('Mean Energy');
title('Mean Energy vs. Load Factor (Inner Fault)');
xlim({"50","300"})
colorbar
ylim([0.0 40.0])
set(gca,"XGrid","off","YGrid","on")
colorbar
Load Index
% Calculate the energy values for each load factor
load_factors = energy_table.LoadFactor;
wavelet_energies = energy_table.WaveletEnergy;
```

```
% Normalize the energy values using min-max scaling
min_energy = min(wavelet_energies);
max_energy = max(wavelet_energies);
normalized_energies = (wavelet_energies - min_energy) / (max_energy - min_energy);
% Find the indices corresponding to the normal condition (load factor 270)
normal_indices = find(load_factors == 270);
% Calculate the deviation from the normal condition
deviation = normalized_energies;
deviation(normal_indices) = 0;
% Define severity thresholds
mild_threshold = 0.2; % Adjust according to your application
moderate_threshold = 0.5; % Adjust according to your application
% Categorize the severity based on deviation magnitude
severity = cell(size(normalized_energies));
severity(deviation < mild_threshold) = {'Mild'};
severity(deviation >= mild_threshold & deviation < moderate_threshold) = {'Moderate'};
severity(deviation >= moderate_threshold) = {'Severe'};
% Determine the number of elements to keep
num_elements = min([numel(load_factors), numel(normalized_energies), numel(deviation),
numel(severity)]);
% Print out the sizes of the arrays for debugging
fprintf('Size of load_factors: %d\n', numel(load_factors));
fprintf('Size of normalized_energies: %d\n', numel(normalized_energies));
fprintf('Size of deviation: %d\n', numel(deviation));
fprintf('Size of severity: %d\n', numel(severity));
fprintf('Number of elements to keep: %d\n', num_elements);


% Truncate the arrays to the common size
load_factors = load_factors(1:num_elements);
deviation = deviation(1:num_elements);
severity = severity(1:num_elements);
% Truncate the normalized_energies array separately
normalized_energies = normalized_energies(1:numel(load_factors));
```

```
% Reshape the arrays to have the same dimensions
load_factors = reshape(load_factors, [], 1);
normalized_energies = reshape(normalized_energies, [], 1);
deviation = reshape(deviation, [], 1);
severity = reshape(severity, [], 1);
% Create a table to store the results
degradation_table_inner = table(load_factors, mean_energy, normalized_energies, deviation,
severity, 'VariableNames', {'LoadFactor', 'MeanEnergy', 'Normalized Energy','Deviation from
Normal','Severity of Changing Load'});
% Display the degradation table
disp(degradation_table_inner);
Outer Load Comparison
ORF_50 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_outer_load', 'OuterRaceFault_vload_2.mat'));
% Plot bearing signal and scalogram
plotBearingSignalAndScalogram(ORF_50)
ORF_100 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_outer_load', 'OuterRaceFault_vload_3.mat'));
% Plot bearing signal and scalogram
plotBearingSignalAndScalogram(ORF_100)

ORF_150 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_outer_load', 'OuterRaceFault_vload_4.mat'));
% Plot bearing signal and scalogram
plotBearingSignalAndScalogram(ORF_150)
ORF_200 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_outer_load', 'OuterRaceFault_vload_5.mat'));
% Plot bearing signal and scalogram
plotBearingSignalAndScalogram(ORF_200)
ORF_250 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
```

```
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_outer_load', 'OuterRaceFault_vload_6.mat'));
% Plot bearing signal and scalogram
plotBearingSignalAndScalogram(ORF_250)
ORF_300 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_outer_load', 'OuterRaceFault_vload_7.mat'));
% Plot bearing signal and scalogram
plotBearingSignalAndScalogram(ORF_200)
Outer ensemble
% Load dataset of different Inner load factors and corresponding variable names
load_factors = [50, 100, 150, 200, 250, 270, 300];
variable_names = {'ORF_50', 'ORF_100', 'ORF_150', 'ORF_200', 'ORF_250', 'data_normal',
'ORF_300'};



% Initialize an empty ensemble with two columns
ensemble_outer = table();

% Find the minimum length of bearing signal data across all load factors
min_length = Inf; % Initialize with infinity
for i = 1:length(load_factors)
  % Construct the variable name
  variable_name = variable_names{i};

  if strcmp(variable_name, 'data_normal')
    % Extract the desired portion of the bearing signal from the loaded dataset
    fs = data_normal.bearing.sr;
    t_total = 3; % seconds
    n = round(t_total * fs);
    bearing_data = data_normal.bearing.gs(1:n);
  else
    % Extract the desired portion of the bearing signal from the loaded dataset
    fs = eval([variable_name '.bearing.sr']);
```

```
    t_total = 3; % seconds
    n = round(t_total * fs);
    bearing_data = eval([variable_name '.bearing.gs(1:n)']);
  end

  % Update the minimum length
  min_length = min(min_length, length(bearing_data));
end
for i = 1:length(load_factors)
  % Construct the variable name
  variable_name = variable_names{i};

  if strcmp(variable_name, 'data_normal')
    % Extract the desired portion of the bearing signal from the loaded dataset
    fs = data_normal.bearing.sr;
    bearing_data = data_normal.bearing.gs(1:min_length);
  else
    % Extract the desired portion of the bearing signal from the loaded dataset
    fs = eval([variable_name '.bearing.sr']);
    bearing_data = eval([variable_name '.bearing.gs(1:min_length)']);
  end
  % Create a timetable for the current variable
  timetable_data = timetable(seconds(0:1/fs:(length(bearing_data)-1)/fs)', bearing_data);
  % Add the timetable and load factor as rows to the ensemble
  ensemble_outer = [ensemble_outer; {timetable_data, load_factors(i)}];
end
% Rename the columns of the ensemble
ensemble_outer.Properties.VariableNames = {'Timetable', 'LoadFactor'};
% Display the resulting ensemble
disp(ensemble_outer);
Use CWT "Amor" in finding the energy of each load and proof the difference
% Define the wavelet type and parameters
wavelet_type = 'amor';
num_scales = 5;
```

```
% Initialize empty cell arrays to store load factors and energies
load_factors = ensemble_outer.LoadFactor;
energies = cell(length(load_factors), 1);
% Calculate wavelet energy for each load
for i = 1:length(load_factors)
  % Extract the timetable data for the current load
  timetable_data = ensemble_outer.Timetable{i};
  % Get the vibration signal from the timetable data
  vibration_signal = timetable_data.Variables;
  % Adjust the length of the vibration signal for load 270 if necessary
  if strcmp(load_factors(i), 270)
    target_length = length(vibration_signal) - (length(vibration_signal) -
length(ensemble_outer).Timetable{1}.Variables);
    vibration_signal = vibration_signal(1:target_length);
  end
  % Perform wavelet transform
  [cfs, ~] = cwt(vibration_signal, wavelet_type, num_scales);
  % Calculate wavelet energy as the squared absolute values of coefficients
  wse = sum(abs(cfs).^2, 1);

  % Store the energy
  energies{i} = wse;
end
% Create separate tables for each load factor
tables = cell(length(load_factors), 1);
for i = 1:length(load_factors)
  tables{i} = table(repmat(load_factors(i), size(energies{i})), energies{i}, 'VariableNames',
{'LoadFactor', 'WaveletEnergy'});
end
% Concatenate the tables into a single table
energy_table_outer = vertcat(tables{:});
% Display the energy table
disp(energy_table_outer);
```

```
% Find unique load factors and create group indices
[unique_load_factors, ~, load_factor_group_indices] = unique(energy_table_outer.LoadFactor);
% Calculate mean energy using loop and if condition
mean_energy = NaN(size(unique_load_factors)); % Initialize mean energy vector

for i = 1:length(unique_load_factors) % organised by numbers orders
  idx = (load_factor_group_indices == i); % Find indices for current load factor group
  mean_energy(i) = mean(energy_table_outer.WaveletEnergy(idx)); % Calculate mean energy
end
% Create a new table to store the results
mean_energy_table_outer = table(unique_load_factors, mean_energy, 'VariableNames',
{'LoadFactor', 'MeanEnergy'});
 Display the mean energy table
disp(mean_energy_table_outer);
%information
whos ensemble_outer
head(ensemble_outer)
ensemble_outer.Properties.VariableNames
properties(ensemble_outer{1, 1}{1})
ensemble_outer(1:5, :)
ensemble_outer.Properties.VariableNames
summary(ensemble_outer{1, 1}{1})



% Bar plot of mean energy
figure;
bar(mean_energy_table_outer.LoadFactor, mean_energy_table_outer.MeanEnergy);
xlabel('Load Factor');
ylabel('Mean Energy');
title('Mean Energy vs. Load Factor (Outer Fault)');
xlim({"50","300"})
colorbar
colorbar
```

```
ylim([0.0 40.0])
set(gca,"XGrid","off","YGrid","on")
colorbar
Load Index
% Calculate the energy values for each load factor
load_factors = energy_table_outer.LoadFactor;
wavelet_energies = energy_table_outer.WaveletEnergy;
% Normalize the energy values using min-max scaling
min_energy = min(wavelet_energies);
max_energy = max(wavelet_energies);
normalized_energies = (wavelet_energies - min_energy) / (max_energy - min_energy);
% Find the indices corresponding to the normal condition (load factor 270)
normal_indices = find(load_factors == 270);
% Calculate the deviation from the normal condition
deviation = normalized_energies;
deviation(normal_indices) = 0;


% Define severity thresholds
mild_threshold = 0.2; % Adjust according to your application
moderate_threshold = 0.5; % Adjust according to your application
% Categorize the severity based on deviation magnitude
severity = cell(size(normalized_energies));
severity(deviation < mild_threshold) = {'Mild'};
severity(deviation >= mild_threshold & deviation < moderate_threshold) = {'Moderate'};
severity(deviation >= moderate_threshold) = {'Severe'};
% Determine the number of elements to keep
num_elements = min([numel(load_factors), numel(normalized_energies), numel(deviation),
numel(severity)]);
% Print out the sizes of the arrays for debugging
fprintf('Size of load_factors: %d\n', numel(load_factors));
fprintf('Size of normalized_energies: %d\n', numel(normalized_energies));
fprintf('Size of deviation: %d\n', numel(deviation));
fprintf('Size of severity: %d\n', numel(severity));
```

```
fprintf('Number of elements to keep: %d\n', num_elements);
% Truncate the arrays to the common size
load_factors = load_factors(1:num_elements);
deviation = deviation(1:num_elements);
severity = severity(1:num_elements);
% Truncate the normalized_energies array separately
normalized_energies = normalized_energies(1:numel(load_factors));
% Reshape the arrays to have the same dimensions
load_factors = reshape(load_factors, [], 1);
normalized_energies = reshape(normalized_energies, [], 1);
deviation = reshape(deviation, [], 1);
severity = reshape(severity, [], 1);
% Create a table to store the results
degradation_table_outer = table(load_factors, mean_energy, normalized_energies, deviation,
severity, 'VariableNames', {'LoadFactor', 'MeanEnergy', 'Normalized
Energy','Deviation','Severity of Changing Load'});
% Display the degradation table
disp(degradation_table_outer);
Significance
% Initialize variables to store results
loadFactors = unique(degradation_table_outer.LoadFactor)';
numLoads = numel(loadFactors);
meanValues = zeros(numLoads, 1);
stdValues = zeros(numLoads, 1);
seValues = zeros(numLoads, 1);
meanDiffValues = zeros(numLoads, 1);
ciValues = zeros(numLoads, 2);
tValues = zeros(numLoads, 1);
dfValues = zeros(numLoads, 1);
pValues = zeros(numLoads, 1);
% Extract data for the normal load (270)
data_normal = energy_table_outer.WaveletEnergy(degradation_table_outer.LoadFactor ==
270);
% Loop through each load and perform the t-test
```

```
for i = 1:numLoads
  load = loadFactors(i);
   if load ~= 270
    % Extract data for the current load
    data_current = energy_table_outer.WaveletEnergy(energy_table_outer.LoadFactor == load);
    % Perform the t-test
    [h, p, ci, stats] = ttest2(data_normal, data_current);
    % Store results
    meanValues(i) = mean(data_current);
    stdValues(i) = std(data_current);
    seValues(i) = std(data_current) / sqrt(length(data_current));
    meanDiffValues(i) = mean(data_current) - mean(data_normal);
    ciValues(i, :) = ci;
    tValues(i) = stats.tstat;
    dfValues(i) = stats.df;
    pValues(i) = p;
    % Display results
    fprintf('\nLoad Factor %d vs. Normal (270):\n', load);
    fprintf('Number of Samples: %d\n', length(data_current));
    fprintf('Mean: %f\n', meanValues(i));
    fprintf('Standard Deviation: %f\n', stdValues(i));
    fprintf('Standard Error of the Mean: %f\n', seValues(i));
    fprintf('Mean Difference: %f\n', meanDiffValues(i));
    fprintf('95%% Confidence Interval: [%f, %f]\n', ciValues(i, 1), ciValues(i, 2));
    fprintf('t-test Value: %f\n', tValues(i));
    fprintf('Degrees of Freedom: %f\n', dfValues(i));
    fprintf('p-value: %f\n', pValues(i));
    % Check for significance
    if h
      fprintf('Significant difference\n');
    else
      fprintf('No significant difference\n');
    end
  end
```

```
end
% Initialize variables to store results
loadFactors = unique(energy_table_outer.LoadFactor)';
numLoads = numel(loadFactors);
resultsTable = table('Size', [numLoads, 11], 'VariableTypes', {'double', 'double', 'double',
'double', 'double', 'double', 'double', 'double', 'double', 'double', 'logical'}, 'VariableNames',
{'LoadFactor', 'NumSamples', 'Mean', 'StdDev', 'SEMean', 'MeanDiff', 'CI_Lower', 'CI_Upper',
'tValue', 'DF', 'Significant'});

% Extract data for the normal load (270)
data_normal = energy_table_outer.WaveletEnergy(energy_table_outer.LoadFactor == 270);
% Loop through each load and perform the t-test
for i = 1:numLoads
  load = loadFactors(i);

  if load ~= 270
    % Extract data for the current load
    data_current = energy_table_outer.WaveletEnergy(energy_table_outer.LoadFactor == load);

    % Perform the t-test
    [h, ~, ci, stats] = ttest2(data_normal, data_current);
    % Store results in the table
    resultsTable(i, :) = {load, length(data_current), mean(data_current), std(data_current),
std(data_current) / sqrt(length(data_current)), mean(data_current) - mean(data_normal), ci(1),
ci(2), stats.tstat, stats.df, h};
  end
end
% Display the results table
disp(resultsTable);
% Initialize variables to store results
loadFactors = unique(energy_table_outer.LoadFactor)';
numLoads = numel(loadFactors);
resultsTable = table('Size', [numLoads, 12], 'VariableTypes', {'double', 'double', 'double',
'double', 'double', 'double', 'double', 'double', 'double', 'double', 'double', 'logical'},
```

```
'VariableNames', {'LoadFactor', 'NumSamples', 'Mean', 'StdDev', 'SEMean', 'MeanDiff',
'CI_Lower', 'CI_Upper', 'tValue', 'DF', 'pValue', 'Significant'});
% Extract data for the normal load (270)
data_normal = energy_table_outer.WaveletEnergy(energy_table_outer.LoadFactor == 270);
% Loop through each load and perform the t-test
for i = 1:numLoads
 load = loadFactors(i);

 if load ~= 270
  % Extract data for the current load
  data_current = energy_table_outer.WaveletEnergy(energy_table_outer.LoadFactor == load);

  % Perform the t-test
  [h, p, ci, stats] = ttest2(data_normal, data_current)
 % Store results in the table
  resultsTable(i, :) = {load, length(data_current), mean(data_current), std(data_current),
std(data_current) / sqrt(length(data_current)), mean(data_current) - mean(data_normal), ci(1),
ci(2), stats.tstat, stats.df, p, h};
 end
end
% Display the results table
disp(resultsTable);
```

SPLITTING

diagnosticFeatureDesigner

Features Ranking

```
% Load the provided table
% Load the provided table load (OuterONLY_feature extraction_Phase1_thesis)
% Exclude the 'LoadFactor' and 'BandPower' columns
features = FeatureTable1_3(:, 2:end-1);
% Calculate the standard deviation for each feature
stdValues = varfun(@std, features);
% Calculate the range for each feature
rangeValues = varfun(@(x) max(x) - min(x), features);
```

```
% Get the feature names
featureNames = cellfun(@(x) x(strfind(x, '/')+1:end), features.Properties.VariableNames,
'UniformOutput', false);
% Create a ranking table with feature names, std, and range
rankingTable = table(featureNames', stdValues.Variables', rangeValues.Variables',
'VariableNames', {'Feature', 'Std', 'Range'});
% Sort the ranking table based on the desired metric (e.g., std or range)
sortedTable = sortrows(rankingTable, 'Std', 'descend'); % Change 'Std' to 'Range' if you want to
rank by range
% Display the sorted table
disp(sortedTable);
tableHeadings_o = FeatureTable1_3.Properties.VariableNames;
Wavelet coefficents comparison (Inner)
plotWaveletCoefficients(IRF_50);
plotWaveletCoefficients(IRF_100);
plotWaveletCoefficients(IRF_150);
plotWaveletCoefficients(IRF_200);
plotWaveletCoefficients(IRF_250);
plotWaveletCoefficients(IRF_300);
Wavelet coefficents comparison (Outer)
plotWaveletCoefficients(ORF_50);
plotWaveletCoefficients(ORF_100);
plotWaveletCoefficients(ORF_150);
plotWaveletCoefficients(ORF_200);
plotWaveletCoefficients(ORF_250);
plotWaveletCoefficients(ORF_300);
Scores (Import 2xeach heath condition then compute the average)
%im here I need to modify
wavelet_type = {'amor'};
%inner
  wse_IRF_50 = calculateWaveletSingularEntropy(IRF_50, wavelet_type)
  wse_IRF_100 = calculateWaveletSingularEntropy(IRF_100, wavelet_type)
  wse_IRF_150 = calculateWaveletSingularEntropy(IRF_150, wavelet_type)
  wse_IRF_200 = calculateWaveletSingularEntropy(IRF_200, wavelet_type)
```

```
 wse_IRF_250 = calculateWaveletSingularEntropy(IRF_250, wavelet_type)
 wse_IRF_300 = calculateWaveletSingularEntropy(IRF_300, wavelet_type)
%outer
wse_ORF_50 = calculateWaveletSingularEntropy(IRF_50, wavelet_type)
 wse_ORF_100 = calculateWaveletSingularEntropy(ORF_100, wavelet_type)
 wse_ORF_150 = calculateWaveletSingularEntropy(ORF_150, wavelet_type)
 wse_ORF_200 = calculateWaveletSingularEntropy(ORF_200, wavelet_type)
 wse_ORF_250 = calculateWaveletSingularEntropy(ORF_250, wavelet_type)
 wse_ORF_300 = calculateWaveletSingularEntropy(ORF_300, wavelet_type)

%healthy
wse_data_normal_270 = calculateWaveletSingularEntropy(data_normal, wavelet_type)
Average WSE scores
% Calculate the average WSE score Inner
avgScore_IRF_50 = calculateWaveletSingularEntropy(IRF_50, 'amor');
avgScore_IRF_100 = calculateWaveletSingularEntropy(IRF_100, 'amor');
avgScore_IRF_150 = calculateWaveletSingularEntropy(IRF_150, 'amor');
avgScore_IRF_200 = calculateWaveletSingularEntropy(IRF_200, 'amor');
avgScore_IRF_250 = calculateWaveletSingularEntropy(IRF_250, 'amor');
avgScore_IRF_300 = calculateWaveletSingularEntropy(IRF_300, 'amor');
%mean
avgScore_IRF_50 = mean(abs(avgScore_IRF_50));
avgScore_IRF_100 = mean(abs(avgScore_IRF_100));
avgScore_IRF_150 = mean(abs(avgScore_IRF_150));
avgScore_IRF_200 = mean(abs(avgScore_IRF_200));
avgScore_IRF_250 = mean(abs(avgScore_IRF_250));
avgScore_IRF_300 = mean(abs(avgScore_IRF_300));
% Display the average score with the variable name
disp(['Average score for IRF_50: ' num2str(avgScore_IRF_50)]);
disp(['Average score for IRF_100: ' num2str(avgScore_IRF_100)]);
disp(['Average score for IRF_150: ' num2str(avgScore_IRF_150)]);
disp(['Average score for IRF_200: ' num2str(avgScore_IRF_200)]);
disp(['Average score for IRF_250: ' num2str(avgScore_IRF_250)]);
disp(['Average score for IRF_300: ' num2str(avgScore_IRF_300)]);
```

```
diagnosticFeatureDesigner
% Create a table of the scores
scores_table = array2table(scores, 'VariableNames', dataset_names, 'RowNames',
wavelet_types);
disp(scores_table);
wavelet_types = {'bump', 'morse', 'amor'};
health_datasets = {'data_normal', 'data_normal2', 'data_inner', 'data_inner2', 'data_outer',
'data_outer2'};
mean_wse_values = zeros(length(wavelet_types), length(health_datasets));
for i = 1:length(wavelet_types)
  wavelet_type = wavelet_types{i};
  fprintf('Wavelet Type: %s\n', wavelet_type);

  for j = 1:length(health_datasets)
    dataset_name = health_datasets{j};
    dataset = eval(dataset_name); % Evaluate the dataset variable using its name

    wse = calculateWaveletSingularEntropy(dataset, wavelet_type);
    mean_wse = mean(abs(wse)); % Calculate mean of absolute WSE values

    mean_wse_values(i, j) = mean_wse;
  end
end
% Plot the mean absolute WSE values
figure
bar(mean_wse_values)
xticks(1:length(wavelet_types))
xticklabels(wavelet_types)
legend(health_datasets)
title('Mean Absolute WSE Values')
xlabel('Wavelet Type')
ylabel('Mean Absolute WSE')

set(gca,"XGrid","off","YGrid","on")
```

```
convertSignalToScalogram_spliiting(ensemble)
Helper Functions
function plotBearingSignalAndScalogram(data)
% Convert 1-D bearing signals to scalograms through wavelet transform
fs = data.bearing.sr;
t_total = 0.1; % seconds
n = round(t_total*fs);
bearing = data.bearing.gs(1:n);
[cfs,frq] = cwt(bearing,'amor', fs);
% Plot the original signal and its scalogram
figure
subplot(2,1,1)
plot(0:1/fs:(n-1)/fs,bearing)
xlim([0,0.1])
title(['Vibration Signal - ' inputname(1)])
xlabel('Time (s)')
ylabel('Amplitude')
subplot(2,1,2)
surface(0:1/fs:(n-1)/fs,frq,abs(cfs))
shading flat
xlim([0,0.1])
ylim([0,max(frq)])
title(['Scalogram - ' inputname(1)])
xlabel('Time (s)')
ylabel('Frequency (Hz)')
end
function convertSignalToScalogram(ensemble,folderName)
% Convert 1-D signals to scalograms and save scalograms as images
data = read(ensemble);
fs = data.sr;
x = data.gs{:};
label = char(data.Label);
fname = char(data.FileName);
```

```
ratio = 5000/97656;
interval = ratio*fs; % i want part of the data not all and morese uses fs as the space equal space but based on the frequency to capture the knowledge
N = floor(numel(x)/interval);
% Create folder to save images
path = fullfile('.',folderName,label);
if ~exist(path,'dir')
 mkdir(path);
end
path_numerical = fullfile('.',folderName,label, 'sub_sample_split1');
if ~exist(path_numerical,'dir')
 mkdir(path_numerical);
end
for idx = 1:N
 sig = envelope(x(interval*(idx-1)+1:interval*idx));
 file_path = fullfile('.',path_numerical,[fname '-' num2str(idx) '.csv']);%(new added)
 writematrix(sig,file_path); %new
 %like we are creating subfolders based on the interval equal data ineach
 %file not using envelop transform
 cfs = cwt(sig,'amor', seconds(1/fs));
 cfs = abs(cfs);
 img = ind2rgb(round(rescale(flip(cfs),0,255)),jet(320));
 outfname = fullfile('.',path,[fname '-' num2str(idx) '.jpg']);
 imwrite(imresize(img,[224,224]),outfname);
end
end
%new
function plotBearingSignalAndWaveletCoefficient(data)
  % Convert 1-D bearing signals to wavelet coefficients through wavelet transform
  fs = data.bearing.sr;
  t_total = 0.1; % seconds
  n = round(t_total * fs);
  bearing = data.bearing.gs(1:n);
```

```
  % Choose wavelet types
  wavelet_types = {'bump', 'morse', 'amor'};

  % Plot the original signal and its wavelet coefficients for each wavelet type
  figure
  for i = 1:length(wavelet_types)
    wavelet_type = wavelet_types{i};
    [cfs, ~] = cwt(bearing, wavelet_type, fs);

    subplot(length(wavelet_types), 1, i)
    plot(0:1/fs:(n-1)/fs, bearing)
    hold on
    plot(0:1/fs:(n-1)/fs, abs(cfs))
    hold off
    xlim([0, 0.1])
    title(sprintf('Wavelet Coefficient (%s)', upper(wavelet_type)))
    xlabel('Time (s)')
    ylabel('Magnitude')
    legend('Vibration Signal', 'Wavelet Coefficient')
  end
end
function wse = calculateWaveletSingularEntropy(data, wavelet_type)
  % Convert 1-D bearing signals to scalograms through wavelet transform
  fs = data.bearing.sr;
  t_total = 0.1; % seconds
  n = round(t_total * fs);
  bearing = data.bearing.gs(1:n);

  % Perform wavelet transform
  [cfs, ~] = cwt(bearing, wavelet_type, fs);
  % Calculate the WSE
  wse = sum(abs(cfs).^2 .* log(abs(cfs).^2), 1);
  % Normalize the WSE values between 0 and 1
  wse = wse / max(wse);
```

```
end
function plotWaveletCoefficients(data)
  % Convert 1-D bearing signals to scalograms through wavelet transform
  fs = data.bearing.sr;
  t_total = 0.1; % seconds
  n = round(t_total * fs);
  bearing = data.bearing.gs(1:n);
  % Choose wavelet types
  wavelet_types = {'amor'};
  % Plot the time-frequency diagrams for each wavelet type
  figure
  for i = 1:length(wavelet_types)
    wavelet_type = wavelet_types{i};
    [cfs, frq] = cwt(bearing, wavelet_type, fs);
   subplot(length(wavelet_types), 1, i)
    plot(0:1/fs:(n-1)/fs, bearing, 'b')
    hold on
    surface(0:1/fs:(n-1)/fs, frq, abs(cfs), 'FaceColor', 'texturemap', 'EdgeColor', 'none')
    colormap(jet)
    view(2)
    xlim([0, 0.1])
    ylim([0, max(frq)])
    title([upper(wavelet_type), ' Wavelet Coefficients - ', inputname(1)])
    xlabel('Time (s)')
    ylabel('Frequency (Hz)')
    colorbar
  end
  sgtitle('Scalograms with Wavelet Coefficients')
end
function plotScalogramsWithWavelets(data)
  % Convert 1-D bearing signals to scalograms through wavelet transform
  fs = data.bearing.sr;
  t_total = 0.1; % seconds
  n = round(t_total * fs);
```

```
  bearing = data.bearing.gs(1:n);

  % Choose wavelet types
  wavelet_types = {'bump', 'morse', 'amor'};

  % Plot the time-frequency diagrams for each wavelet type
  for i = 1:length(wavelet_types)
    wavelet_type = wavelet_types{i};
    [cfs, frq] = cwt(bearing, wavelet_type, fs);

    figure
    imagesc(1/fs:(n-1)/fs, frq, abs(cfs))
    set(gca, 'YDir', 'normal')
    colormap(jet)
    colorbar
    title(sprintf('2D Time-Frequency Diagram (%s)', upper(wavelet_type)))
    xlabel('Time (s)')
    ylabel('Frequency (Hz)')
  end
end


%Splitting
function convertSignalToScalogram_spliiting(ensemble, FileName)
  % Convert 1-D signals to scalograms and save scalograms as images
  data = read(ensemble);
  fs = data.sr;
  x = data.gs{:};
  label = char(data.Label);
  fname = char(data.FileName);
  ratio = 5000/97656;
  interval = ratio * fs; % Interval based on the specified ratio
  % Calculate the number of splits
  N = floor(numel(x) / interval);
```

```
  % Create folder to save images
  path = fullfile('.', folderName, label);
  if ~exist(path, 'dir')
    mkdir(path);
  end
  for idx = 1:N
    % Split the signal into subsets based on the interval
    subset_start = round(interval * (idx - 1)) + 1;
    subset_end = round(interval * idx);
    sig = x(subset_start:subset_end);
    % Perform wavelet transform
    cfs = cwt(sig, 'amor', seconds(1/fs));
    cfs = abs(cfs);
    img = ind2rgb(round(rescale(flip(cfs), 0, 255)), jet(320));

    % Save the image
    outfname = fullfile(path, [fname '-' num2str(idx) '.jpg']);
    imwrite(imresize(img, [224, 224]), outfname);
  end
end

function stats = summary_stats(x)
  stats.MeanEnergy = mean(x);
  stats.MedianEnergy = median(x);
  stats.MinEnergy = min(x);
  stats.MaxEnergy = max(x);
end
```

2.3 Chapter 5: Phase 1: Step2: Data Segmentation and Load-Dependent Subfile Creation

These folders were arranged manually after they were extractted from LoadSplitandScholograms_P4

Now we want to upload these split to diagnosis feature designer

Load DataSets (split files & add the sevierity we created before

Normal Dataset

```
% Import normal bearing data
data_normal = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data', 'baseline_1.mat'));
signal = data_normal.bearing.gs;
signal_length = length(signal)
window = hamming(signal_length)
windowed_signal = signal .* window
```

Inner load comparison

```
IRF_50 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_inner_load', 'InnerRaceFault_vload_2.mat'));
IRF_100 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_inner_load', 'InnerRaceFault_vload_3.mat'));
IRF_150 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_inner_load', 'InnerRaceFault_vload_4.mat'));
IRF_200 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_inner_load', 'InnerRaceFault_vload_5.mat'));
IRF_250 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_inner_load', 'InnerRaceFault_vload_6.mat'));
IRF_300 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_inner_load', 'InnerRaceFault_vload_7.mat'));
```

Outer Load Comparison

```
ORF_50 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_outer_load', 'OuterRaceFault_vload_2.mat'));
```

```
ORF_100 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_outer_load', 'OuterRaceFault_vload_3.mat'));
ORF_150 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_outer_load', 'OuterRaceFault_vload_4.mat'));
ORF_200 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_outer_load', 'OuterRaceFault_vload_5.mat'));
ORF_250 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_outer_load', 'OuterRaceFault_vload_6.mat'));
ORF_300 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data_outer_load', 'OuterRaceFault_vload_7.mat'));

inner_fault_datasets = {ORF_50, ORF_100, ORF_150, ORF_200, ORF_250, ORF_300};

for i = 1:length(inner_fault_datasets)
  ensemble_inner = inner_fault_datasets{i};
  ensemble_inner.DataVariables = ["gs", "sr", "rate", "load", "BPFO", "BPFI", "FTF", "BSF"];
  ensemble_inner.ConditionVariables = ["Label", "FileName"];
  ensemble_inner.SelectedVariables = ["gs", "sr", "rate", "load", "BPFO", "BPFI", "FTF",
"BSF", "Label", "FileName"];
end
outer_fault_datasets = {ORF_50, ORF_100, ORF_150, ORF_200, ORF_250, ORF_300};
for i = 1:length(outer_fault_datasets)
  ensemble_outer = outer_fault_datasets{i};
  ensemble_outer.DataVariables = ["gs", "sr", "rate", "load", "BPFO", "BPFI", "FTF", "BSF"];
  ensemble_outer.SelectedVariables = ["gs", "sr", "rate", "load", "BPFO", "BPFI", "FTF",
"BSF", "Label", "FileName"];
end
Normal_dataset = {data_normal}
for i = 1:length(Normal_dataset)
```

```
  ensemble_normal = Normal_dataset{i};
  ensemble_normal.DataVariables = ["gs", "sr", "rate", "load", "BPFO", "BPFI", "FTF",
"BSF"];
  ensemble_normal.SelectedVariables = ["gs", "sr", "rate", "load", "BPFO", "BPFI", "FTF",
"BSF", "Label", "FileName"];
end
% Process inner fault datasets
for i = 1:numel(inner_fault_datasets)
  ensemble = inner_fault_datasets{i};
  data = ensemble.bearing;
  fs = data.sr;
  x = data.gs;
  ensemble_name = sprintf('IRF_%d', i*50); % Generate ensemble name based on index
  ratio = 5000/97656;
  interval = ratio * fs; % Interval based on the specified ratio
  % Calculate the number of splits
  N = floor(numel(x) / interval);
  % Create a folder to save the tables for the current ensemble
  folder_path = fullfile('.', 'tables', ensemble_name);
  if ~exist(folder_path, 'dir')
    mkdir(folder_path);
  end
  for idx = 1:N
    % Split the signal into subsets based on the interval
    subset_start = round(interval * (idx - 1)) + 1;
    subset_end = round(interval * idx);
    sig = x(subset_start:subset_end);
    % Create a table for the current subset
    time = (subset_start:subset_end)' / fs;
    load_type = repmat(ensemble_name, numel(time), 1);
    subset_table = table(time, sig, load_type, 'VariableNames', {'Time', 'Signal', 'LoadType'});
    % Save the table as a MAT file under the ensemble folder
    filename = sprintf('subset_%d.mat', idx);
    save(fullfile(folder_path, filename), 'subset_table');
```

```
  end
end

% Process outer fault datasets
for i = 1:numel(outer_fault_datasets)
  ensemble = outer_fault_datasets{i};
  data = ensemble.bearing;
  fs = data.sr;
  x = data.gs;
  ensemble_name = sprintf('ORF_%d', i*50); % Generate ensemble name based on index
  ratio = 5000/48828;
  interval = ratio * fs; % Interval based on the specified ratio
  % Calculate the number of splits
  N = floor(numel(x) / interval);
  % Create a folder to save the tables for the current ensemble
  folder_path = fullfile('.', 'tables', ensemble_name);
  if ~exist(folder_path, 'dir')
    mkdir(folder_path);
  end
  for idx = 1:N
    % Split the signal into subsets based on the interval
    subset_start = round(interval * (idx - 1)) + 1;
    subset_end = round(interval * idx);
    sig = x(subset_start:subset_end);
    % Create a table for the current subset
    time = (subset_start:subset_end)' / fs;
    load_type = repmat(ensemble_name, numel(time), 1);
    subset_table = table(time, sig, load_type, 'VariableNames', {'Time', 'Signal', 'LoadType'});
    % Save the table as a MAT file under the ensemble folder
    filename = sprintf('subset_%d.mat', idx);
    save(fullfile(folder_path, filename), 'subset_table');
  end
end
% Process normal dataset
```

```
ensemble = Normal_dataset{1}; % Assuming there is only one ensemble in the normal dataset
data = ensemble.bearing;
fs = data.sr;
x = data.gs;
ensemble_name = 'Normal'; % Ensemble name for the normal dataset
ratio = 5000/97656;
interval = ratio * fs; % Interval based on the specified ratio
% Calculate the number of splits
N = floor(numel(x) / interval);
% Create a folder to save the tables for the normal dataset
folder_path = fullfile('.', 'tables', ensemble_name);
if ~exist(folder_path, 'dir')
  mkdir(folder_path);
end
for idx = 1:N
  % Split the signal into subsets based on the interval
  subset_start = round(interval * (idx - 1)) + 1;
  subset_end = round(interval * idx);
  sig = x(subset_start:subset_end);

  % Create a table for the current subset
  time = (subset_start:subset_end)' / fs;
  load_type = repmat(ensemble_name, numel(time), 1);
  subset_table = table(time, sig, load_type, 'VariableNames', {'Time', 'Signal', 'LoadType'});

  % Save the table as a MAT file under the ensemble folder
  filename = sprintf('subset_%d.mat', idx);
  save(fullfile(folder_path, filename), 'subset_table');
end
Create splits and save as ensemble for all loads
% Create an empty cell array to store the ensemble tables
ensemble_tables = cell(0, 2);

% Process inner fault datasets
```

```
for i = 1:numel(inner_fault_datasets)
  ensemble = inner_fault_datasets{i};
  data = ensemble.bearing;
  fs = data.sr;
  x = data.gs;
  ensemble_name = sprintf('IRF_%d', i*50); % Generate ensemble name based on index
  ratio = 5000/97656;
  interval = ratio * fs; % Interval based on the specified ratio
  % Calculate the number of splits
  N = floor(numel(x) / interval);
  for idx = 1:N
    % Split the signal into subsets based on the interval
    subset_start = round(interval * (idx - 1)) + 1;
    subset_end = round(interval * idx);
    sig = x(subset_start:subset_end);
    % Create a table for the subset with vibration signal
    subset_table = table(sig, 'VariableNames', {'Signal'});
    % Append the subset table and ensemble name to the ensemble tables
    ensemble_tables = [ensemble_tables; {subset_table, ensemble_name}];
  end
end
% Process outer fault datasets
for i = 1:numel(outer_fault_datasets)
  ensemble = outer_fault_datasets{i};
  data = ensemble.bearing;
  fs = data.sr;
  x = data.gs;
  ensemble_name = sprintf('ORF_%d', i*50); % Generate ensemble name based on index
  ratio = 5000/97656;
  interval = ratio * fs; % Interval based on the specified ratio
  % Calculate the number of splits
  N = floor(numel(x) / interval);
  for idx = 1:N
    % Split the signal into subsets based on the interval
```

```
    subset_start = round(interval * (idx - 1)) + 1;
    subset_end = round(interval * idx);
    sig = x(subset_start:subset_end);
    % Create a table for the subset with vibration signal
    subset_table = table(sig, 'VariableNames', {'Signal'});
    % Append the subset table and ensemble name to the ensemble tables
    ensemble_tables = [ensemble_tables; {subset_table, ensemble_name}];
  end
end
% Process normal dataset
ensemble_name = 'Normal';
data = data_normal.bearing;
fs = data.sr;
x = data.gs;
ratio = 5000/97656;
interval = ratio * fs; % Interval based on the specified ratio
% Calculate the number of splits
N = floor(numel(x) / interval);
for idx = 1:N
  % Split the signal into subsets based on the interval
  subset_start = round(interval * (idx - 1)) + 1;
  subset_end = round(interval * idx);
  sig = x(subset_start:subset_end);
  % Create a table for the subset with vibration signal
  subset_table = table(sig, 'VariableNames', {'Signal'});
  % Append the subset table and ensemble name to the ensemble tables
  ensemble_tables = [ensemble_tables; {subset_table, ensemble_name}];
end
% Create the ensemble table with headers
ensemble = table(ensemble_tables(:,1), ensemble_tables(:,2), 'VariableNames', {'Timetable',
'LoadFactor'});
% Display the resulting ensemble table (here it is a table)
disp(ensemble);
% Create the ensemble table with headers
```

```
ensemble = table(ensemble_tables(:,1), ensemble_tables(:,2), 'VariableNames', {'Timetable',
'LoadFactor'});

% Calculate the row counts for each load factor
row_counts = grpstats(ensemble, 'LoadFactor', 'numel');
% Display the row counts
disp(row_counts);
% Assuming you have a table called "ensemble" with 813 rows and 2 columns
for i = 1:size(ensemble, 1)
  % Get the table from the cell in the first column
  cellData = ensemble{i, 1};
  % Access the table within the cell
  tableData = cellData{1};
  % Get the signal data from the table
  signalData = tableData.Signal;
  % Get the load factor from the second column of the ensemble
  loadFactor = ensemble{i, 2};
  % Set the time step based on the load factor
  if strcmp(loadFactor, 'Normal')
    % Normal load with time step 97656 Hz
    timeStep = seconds(1 / 97656);
  else
    % Inner or outer load with time step 48828 Hz
    timeStep = seconds(1 / 48828);
  end
  % Create a time vector for the signal
  time = (0:length(signalData)-1) * timeStep;

  % Convert the signal to a time series
  timeSeriesData = timetable(time', signalData);
  % Replace the cell in the first column with the time series
  ensemble{i, 1} = {timeSeriesData};
end
diagnosticFeatureDesigner
```

Load Index Severity (4 classes)

Create a severity array corresponding to the load types: (6 load types per fault)

```
inner_faults = {'IRF_50', 'IRF_100', 'IRF_150', 'IRF_200', 'IRF_250', 'IRF_300'};
outer_faults = {'ORF_50', 'ORF_100', 'ORF_150', 'ORF_200', 'ORF_250', 'ORF_300'};
normal_load = {'Normal'};
inner_severity = ["Mild", "Mild", "Mild", "Moderate", "Moderate", "Moderate"];
outer_severity = ["Mild", "Mild", "Mild", "Mild", "Mild", "Severe", "Mild"];
normal_severity = ["Helthy"];
```

2. Iterate over the ensemble and add the severity column based on the load type:

```
% Initialize an empty array to store the severity values
severity = strings(size(ensemble, 1), 1);
for i = 1:size(ensemble, 1)
  % Get the type of fault and load from the ensemble
  fault_load = ensemble{i, 2};
  % Check if it belongs to inner faults
  if ismember(fault_load, inner_faults)
    severity(i) = inner_severity(ismember(inner_faults, fault_load));
  % Check if it belongs to outer faults
  elseif ismember(fault_load, outer_faults)
    severity(i) = outer_severity(ismember(outer_faults, fault_load));
  % Otherwise, it belongs to normal
  else
    severity(i) = normal_severity;
  end
end

% Add the severity column to the ensemble
ensemble.Severity = severity;
disp(ensemble)
```

Normal Fault classification 3 classes

```
inner_faults = {'IRF_50', 'IRF_100', 'IRF_150', 'IRF_200', 'IRF_250', 'IRF_300'};
outer_faults = {'ORF_50', 'ORF_100', 'ORF_150', 'ORF_200', 'ORF_250', 'ORF_300'};
normal_load = {'Normal'};
```

```
inner = ["Inner", "Inner", "Inner", "Inner", "Inner", "Inner"];
outer = ["Outer", "Outer", "Outer", "Outer", "Outer", "Outer", "Outer"];
normal_severity = ["Helthy"];
for i = 1:size(ensemble, 1)
  % Get the type of fault and load from the ensemble
  fault_load = ensemble{i, 2};
  % Check if it belongs to inner faults
  if ismember(fault_load, inner_faults)
    severity(i) = inner_severity(ismember(inner_faults, fault_load));
  % Check if it belongs to outer faults
  elseif ismember(fault_load, outer_faults)
    severity(i) = outer_severity(ismember(outer_faults, fault_load));
  % Otherwise, it belongs to normal
  else
    severity(i) = normal_severity;
  end
end

% Add the severity column to the ensemble
ensemble.Severity = severity;
disp(ensemble)
```

2.4 Chapter5 : Phase2: Step1: CWT Signal Encoding and Optimal Technique Selection and Phase 2: Step 2: CWT Energy Assessment for Each Load Factor

Scalogram of Bearing Data

The two dimensions in a scalogram image represent time and frequency. To visualise the relationship between a scalogram and its original vibration signal, plot the vibration signal with an IRF against its scalogram.

```
% Import data with IRF
data_inner = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data', 'InnerRaceFault_vload_1.mat'));
% Plot bearing signal and scalogram
plotBearingSignalAndScalogram(data_inner)
```

```
% for WSE
data_inner2 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data', 'InnerRaceFault_vload_2.mat'));
% Plot bearing signal and scalogram
plotBearingSignalAndScalogram(data_inner2)
```

During the 0.1 seconds shown in the plot, the vibration signal contains 12 impulses because the tested bearing's BPFI is 118.875 Hz. Accordingly, the scalogram shows 12 distinct peaks that align with the impulses in the vibration signal. Next, visualise scalograms for the ORF.

```
% Import data with ORF
data_outer = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'test_data', 'OuterRaceFault_3.mat'));
% Plot original signal and its scalogram
plotBearingSignalAndScalogram(data_outer)
```

The scalogram of the ORF fault shows 8 distinct peaks during the first 0.1 seconds, which is consistent with the ballpass frequencies. Because the impulses in the time-domain signal is not as dominant as in the IRF case, the distinct peaks in the scalogram show less contrast with the background. The scalogram of the normal condition does not show dominant distinct peaks.

```
% Import normal bearing data
data_normal = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data', 'baseline_1.mat'));
% Plot original signal and its scalogram
plotBearingSignalAndScalogram(data_normal)
% Import normal bearing data
data_normal2 = load(fullfile(MATLABroot, 'toolbox', 'predmaint', ...
  'predmaintdemos', 'bearingFaultDiagnosis', ...
  'train_data', 'baseline_2.mat'));
% Plot original signal and its scalogram
plotBearingSignalAndScalogram(data_normal2)
```

The number of distinct peaks is a good feature to differentiate between IRFs, ORFs, and normal conditions. Therefore, a scalogram can be a good candidate for classifying bearing faults. In

this example, all bearing signal measurements come from tests using the same shaft speed. To apply this example to bearing signals under different shaft speeds, the data needs to be normalized by shaft speed. Otherwise, the number of "pillars" in the scalogram will be wrong.

CWT Signal Encoding and Optimal Technique Selection

```
plotWaveletCoefficients(data_inner);
plotWaveletCoefficients(data_normal);
plotWaveletCoefficients(data_outer);
% Plot for IRF condition
plotBearingSignalAndWaveletCoefficient(data_normal);
```

WSE Analysis for Appropriate CWT Selection

```
%im here I need to modify
wavelet_types = {'bump', 'morse', 'amor'};
for i = 1:length(wavelet_types)
  wavelet_type = wavelet_types{i};
  wse_inner = calculateWaveletSingularEntropy(data_inner, wavelet_type)
  wse_inner2 = calculateWaveletSingularEntropy(data_inner2, wavelet_type)
  wse_outer = calculateWaveletSingularEntropy(data_outer, wavelet_type)
  wse_outer2 = calculateWaveletSingularEntropy(data_outer2, wavelet_type)
  wse_normal = calculateWaveletSingularEntropy(data_normal, wavelet_type)
  wse_normal2 = calculateWaveletSingularEntropy(data_normal2, wavelet_type)
    % Do further analysis or visualization with the WSE values
  % ...
end
wavelet_types = {'bump', 'morse', 'amor'};
for i = 1:length(wavelet_types)
  wavelet_type = wavelet_types{i};
  wse_inner = calculateWaveletSingularEntropy(data_inner, wavelet_type);
  wse_inner_mean = mean(abs(wse_inner)); % Calculate mean of absolute WSE values for
data_inner
  wse_inner2 = calculateWaveletSingularEntropy(data_inner2, wavelet_type);
  wse_inner2_mean = mean(abs(wse_inner2)); % Calculate mean of absolute WSE values for
data_inner2
```

```
  wse_outer = calculateWaveletSingularEntropy(data_outer, wavelet_type);
  wse_outer_mean = mean(abs(wse_outer)); % Calculate mean of absolute WSE values for
data_outer

  wse_outer2 = calculateWaveletSingularEntropy(data_outer2, wavelet_type);
  wse_outer2_mean = mean(abs(wse_outer2)); % Calculate mean of absolute WSE values for
data_outer2

  wse_normal = calculateWaveletSingularEntropy(data_normal2, wavelet_type);
  wse_normal_mean = mean(abs(wse_normal)); % Calculate mean of absolute WSE values for
data_normal

  wse_normal2 = calculateWaveletSingularEntropy(data_normal2, wavelet_type);
  wse_normal2_mean = mean(abs(wse_normal2)); % Calculate mean of absolute WSE values
for data_normal2

end
Scores
wavelet_types = {'bump', 'morse', 'amor'};
for i = 1:length(wavelet_types)
  wavelet_type = wavelet_types{i};
  fprintf('Wavelet Type: %s\n', wavelet_type);
  wse_inner = calculateWaveletSingularEntropy(data_inner, wavelet_type);
  wse_inner_mean = mean(abs(wse_inner)) % Calculate mean of absolute WSE values for
data_inner
  wse_inner2 = calculateWaveletSingularEntropy(data_inner2, wavelet_type);
  wse_inner2_mean = mean(abs(wse_inner2)) % Calculate mean of absolute WSE values for
data_inner2
  wse_outer = calculateWaveletSingularEntropy(data_outer, wavelet_type);
  wse_outer_mean = mean(abs(wse_outer)) % Calculate mean of absolute WSE values for
data_outer

  wse_outer2 = calculateWaveletSingularEntropy(data_outer2, wavelet_type);
```

```
  wse_outer2_mean = mean(abs(wse_outer2)) % Calculate mean of absolute WSE values for
data_outer2
  wse_normal = calculateWaveletSingularEntropy(data_normal, wavelet_type);
  wse_normal_mean = mean(abs(wse_normal)) % Calculate mean of absolute WSE values for
data_normal
  wse_normal2 = calculateWaveletSingularEntropy(data_normal2, wavelet_type);
  wse_normal2_mean = mean(abs(wse_normal2)) % Calculate mean of absolute WSE values
for data_normal2
  % Do further analysis or visualization with the mean absolute WSE values
  % ...
end
% Create a table of the scores
scores_table = array2table(scores, 'VariableNames', dataset_names, 'RowNames',
wavelet_types);
disp(scores_table);
wavelet_types = {'bump', 'morse', 'amor'};
health_datasets = {'data_normal', 'data_normal2', 'data_inner', 'data_inner2', 'data_outer',
'data_outer2'};
mean_wse_values = zeros(length(wavelet_types), length(health_datasets));
for i = 1:length(wavelet_types)
  wavelet_type = wavelet_types{i};
  fprintf('Wavelet Type: %s\n', wavelet_type);

  for j = 1:length(health_datasets)
    dataset_name = health_datasets{j};
    dataset = eval(dataset_name); % Evaluate the dataset variable using its name

    wse = calculateWaveletSingularEntropy(dataset, wavelet_type);
    mean_wse = mean(abs(wse)); % Calculate mean of absolute WSE values

    mean_wse_values(i, j) = mean_wse;
  end
end
% Plot the mean absolute WSE values
```

```
figure
bar(mean_wse_values)
xticks(1:length(wavelet_types))
xticklabels(wavelet_types)
legend(health_datasets)
title('Mean Absolute WSE Values')
xlabel('Wavelet Type')
ylabel('Mean Absolute WSE')
set(gca,"XGrid","off","YGrid","on")
Helper Functions
function plotBearingSignalAndScalogram(data)
% Convert 1-D bearing signals to scalograms through wavelet transform
fs = data.bearing.sr;
t_total = 0.1; % seconds
n = round(t_total*fs);
bearing = data.bearing.gs(1:n);
[cfs,frq] = cwt(bearing,'amor', fs);
% Plot the original signal and its scalogram
figure
subplot(2,1,1)
plot(0:1/fs:(n-1)/fs,bearing)
xlim([0,0.1])
title('Vibration Signal')
xlabel('Time (s)')
ylabel('Amplitude')
subplot(2,1,2)
surface(0:1/fs:(n-1)/fs,frq,abs(cfs))
shading flat
xlim([0,0.1])
ylim([0,max(frq)])
title('Scalogram')
xlabel('Time (s)')
ylabel('Frequency (Hz)')
end
```

```
function convertSignalToScalogram(ensemble,folderName)
% Convert 1-D signals to scalograms and save scalograms as images
data = read(ensemble);
fs = data.sr;
x = data.gs{:};
label = char(data.Label);
fname = char(data.FileName);
ratio = 5000/97656;
interval = ratio*fs;
N = floor(numel(x)/interval);
% Create folder to save images
path = fullfile('.',folderName,label);
if ~exist(path,'dir')
 mkdir(path);
end
path_numerical = fullfile('.',folderName,label, 'sub_sample_split1');
if ~exist(path_numerical,'dir')
 mkdir(path_numerical);
end
for idx = 1:N
 sig = envelope(x(interval*(idx-1)+1:interval*idx));
 file_path = fullfile('.',path_numerical,[fname '-' num2str(idx) '.csv']);%(new added)
 writematrix(sig,file_path); %new
 %like we are creating subfolders based on the interval equal data ineach
 %file not using envelop transform
 cfs = cwt(sig,'amor', seconds(1/fs));
 cfs = abs(cfs);
 img = ind2rgb(round(rescale(flip(cfs),0,255)),jet(320));
 outfname = fullfile('.',path,[fname '-' num2str(idx) '.jpg']);
 imwrite(imresize(img,[224,224]),outfname);
end
end
%new
function plotBearingSignalAndWaveletCoefficient(data)
```

```
  % Convert 1-D bearing signals to wavelet coefficients through wavelet transform
  fs = data.bearing.sr;
  t_total = 0.1; % seconds
  n = round(t_total * fs);
  bearing = data.bearing.gs(1:n);
  % Choose wavelet types
  wavelet_types = {'bump', 'morse', 'amor'};
  % Plot the original signal and its wavelet coefficients for each wavelet type
  figure
  for i = 1:length(wavelet_types)
    wavelet_type = wavelet_types{i};
    [cfs, ~] = cwt(bearing, wavelet_type, fs);

    subplot(length(wavelet_types), 1, i)
    plot(0:1/fs:(n-1)/fs, bearing)
    hold on
    plot(0:1/fs:(n-1)/fs, abs(cfs))
    hold off
    xlim([0, 0.1])
    title(sprintf('Wavelet Coefficient (%s)', upper(wavelet_type)))
    xlabel('Time (s)')
    ylabel('Magnitude')
    legend('Vibration Signal', 'Wavelet Coefficient')
  end
end
function wse = calculateWaveletSingularEntropy(data, wavelet_type)
  % Convert 1-D bearing signals to scalograms through wavelet transform
  fs = data.bearing.sr;
  t_total = 0.1; % seconds
  n = round(t_total * fs);
  bearing = data.bearing.gs(1:n);
   % Perform wavelet transform
  [cfs, ~] = cwt(bearing, wavelet_type, fs);
  % Calculate the WSE
```

```
  wse = sum(abs(cfs).^2 .* log(abs(cfs).^2), 1); % Updated to calculate along columns
  % Normalize the WSE values between 0 and 1
  wse = wse / max(wse);
end
function plotWaveletCoefficients(data)
  % Convert 1-D bearing signals to scalograms through wavelet transform
  fs = data.bearing.sr;
  t_total = 0.1; % seconds
  n = round(t_total * fs);
  bearing = data.bearing.gs(1:n);
    % Choose wavelet types
  wavelet_types = {'bump', 'morse', 'amor'};
  % Plot the time-frequency diagrams for each wavelet type
  figure
  for i = 1:length(wavelet_types)
    wavelet_type = wavelet_types{i};
    [cfs, frq] = cwt(bearing, wavelet_type, fs);
    subplot(length(wavelet_types), 1, i)
    plot(0:1/fs:(n-1)/fs, bearing, 'b')
    hold on
    surface(0:1/fs:(n-1)/fs, frq, abs(cfs), 'FaceColor', 'texturemap', 'EdgeColor', 'none')
    colormap(jet)
    view(2)
    xlim([0, 0.1])
    ylim([0, max(frq)])
    title(upper(wavelet_type))
    xlabel('Time (s)')
    ylabel('Frequency (Hz)')
    colorbar
  end
  sgtitle('Scalograms with Wavelet Coefficients')
end
function plotScalogramsWithWavelets(data)
  % Convert 1-D bearing signals to scalograms through wavelet transform
```

```
fs = data.bearing.sr;
t_total = 0.1; % seconds
n = round(t_total * fs);
bearing = data.bearing.gs(1:n);

% Choose wavelet types
wavelet_types = {'bump', 'morse', 'amor'};
 % Plot the time-frequency diagrams for each wavelet type
for i = 1:length(wavelet_types)
 wavelet_type = wavelet_types{i};
 [cfs, frq] = cwt(bearing, wavelet_type, fs);

 figure
 imagesc(1/fs:(n-1)/fs, frq, abs(cfs))
 set(gca, 'YDir', 'normal')
 colormap(jet)
 colorbar
 title(sprintf('2D Time-Frequency Diagram (%s)', upper(wavelet_type)))
 xlabel('Time (s)')
 ylabel('Frequency (Hz)')
 end
end
```

2.5 Chapter 6: Full Code

1. Same dataset used previously for training the ALexNet (CWT)

```
rng(15); % Set the random seed (you change every time you repeat the experiment ) use
(1,3,6,9,12,15,21,24,27,30) for reproducibility
% Specify the path to your images folder
ImagesPath_CWTFLI =
'C:\Users\Shahd\Documents\MATLAB\Examples\R2023a\predmaint\RollingElementBearingF
aultDiagnosisExample\CWT_LoadIndexSplit_1';
% Load and preprocess your dataset with resized images (according to the
% CNN size)
inputSize = [227, 227, 3];
generatedImages = imageDatastore(ImagesPath_CWTFLI, ...
```

```
  'IncludeSubfolders', true, ...
  'LabelSource', 'foldernames', ...
  'ReadFcn', @(filename) imresize(imread(filename), inputSize(1:2)));
% Split the dataset into training, validation, and testing datastores
rng(30); % Set the random seed for reproducibility
[trainImages_CWTLI, valImages_CWTLI, testImages_CWTLI] =
splitEachLabel(generatedImages, 0.6, 0.2, 0.2, 'randomized');
% Display the number of images in each split
numTrainImagesCWTLI = numel(trainImages_CWTLI.Files);
numValImagesCWTLI = numel(valImages_CWTLI.Files);
numTestImagesCWTLI = numel(testImages_CWTLI.Files);
disp(['Number of training images: ', num2str(numTrainImagesCWTLI)]);
disp(['Number of validation images: ', num2str(numValImagesCWTLI)]);
disp(['Number of testing images: ', num2str(numTestImagesCWTLI)]);
trainingLabels_CWTLI = countEachLabel(trainImages_CWTLI)
validationLabels_CWTLI= countEachLabel(valImages_CWTLI)
testLabels_CWTLI= countEachLabel(testImages_CWTLI)
% Specify the path to your images folder
ImagesPath_GADFLI =
'C:\Users\Shahd\Documents\MATLAB\Examples\R2023a\predmaint\RollingElementBearingF
aultDiagnosisExample\GADF_LoadIndexSplit_1';
% Load and preprocess your dataset with resized images (according to the
% CNN size)
inputSize = [227, 227, 3];
generatedImages_GADF = imageDatastore(ImagesPath_GADFLI, ...
  'IncludeSubfolders', true, ...
  'LabelSource', 'foldernames', ...
  'ReadFcn', @(filename) imresize(imread(filename), inputSize(1:2)));

% Split the dataset into training, validation, and testing datastores
% (images label stats with healthy, mild, moderate then severe)
rng(30); % Set the random seed for reproducibility
[trainImages_GADFLI, valImages_GADFLI, testImages_GADFLI] =
splitEachLabel(generatedImages_GADF, 0.6, 0.2, 0.2, 'randomized');
```

```
% Display the number of images in each split
numTrainImagesGADFLI = numel(trainImages_GADFLI.Files);
numValImagesGADFLI = numel(valImages_GADFLI.Files);
numTestImagesGADFLI = numel(testImages_GADFLI.Files);
disp(['Number of training images: ', num2str(numTrainImagesGADFLI)]);
disp(['Number of validation images: ', num2str(numValImagesGADFLI)]);
disp(['Number of testing images: ', num2str(numTestImagesGADFLI)]);
trainingLabels_GADFLI = countEachLabel(trainImages_GADFLI)
validationLabels_GADFLI= countEachLabel(valImages_GADFLI)
testLabels_GADFLI= countEachLabel(testImages_GADFLI)
```

2. Load(ensemble _MFPT_subfiles_SeverityAdded)just click will show ensemble

```
% Assuming your ensemble is stored in a table or a timetable called ' ensemble '
%add an index column
ensemble.Index = (1:size(ensemble, 1))';
%change this because I have spilling mistake
% Assuming your ensemble is stored in a table or a timetable called 'ensemble'
% Correct the spelling mistake in the 'Severity' column
ensemble.Severity = strrep(ensemble.Severity, 'Helthy', 'Healthy');
%sort the ensemble by severity (open the ensemble variable and sort it
%manually ascending order) this is because the FeatureTable1_2 was but
%based on the original order but we will convert it this way to march CWT
%and GADF order
%in the following section we will add the image path column similar to the
%datastore
Load (LI_20Features_table_p4.mat)just click will show FeatureTable1_2
fix spelling mistake in healthy
FeatureTable1_2.Severity = strrep(ensemble.Severity, 'Helthy', 'Healthy');
```

3. upload the ensemble with LI severity and add images path

```
% Specify the path to your images folder
CWTImagesPath =
'C:\Users\Shahd\Documents\MATLAB\Examples\R2023a\predmaint\RollingElementBearingF
aultDiagnosisExample\CWT_Images';
% Assuming 'ensemble' contains the ensemble with the added LoadFactor column
num_samples = height(ensemble);
```

```
imagePaths = cell(num_samples, 1);
% Create the image paths and add them to the ensemble table
folder_indices = containers.Map(); % To keep track of the current index for each folder
for i = 1:num_samples
  load_factor = ensemble.LoadFactor{i};
  if ~isKey(folder_indices, load_factor)
    folder_indices(load_factor) = 1;
  end
  current_index = folder_indices(load_factor);
    img_folder = fullfile(CWTImagesPath, load_factor);
  img_name = [load_factor, '-', num2str(current_index), '.jpg'];
  img_path = fullfile(img_folder, img_name);
  imagePaths{i} = img_path;
    % Update the index for the current folder
  folder_indices(load_factor) = current_index + 1;
end
% Add the 'ImagePath' column to the ensemble table (on the enesemble)
ensemble.ImagePath = imagePaths;
```

4. Splitting Consistensy

```
% because we splitting from the datastore arranges as (healthy, mild,
% moderate and severe) now we are taking the exact paths
trainImagePaths_CWTLI = trainImages_CWTLI.Files;
valImagePaths_CWTLI = valImages_CWTLI.Files;
testImagePaths_CWTLI = testImages_CWTLI.Files;
```

- Adding the 'ImagePath' column from the ensemble table to FeatureTable1_2 and then using it for splitting will ensure that you're matching the images consistently between your CWT dataset and the feature dataset.

```
% Create a list of image file names for training, validation, and testing
% datasets of CWT images
%here you can see the images names used for train valid and test
trainImageNames_CWTLI = cellfun(@(x) getFileNameFromPath(x),
trainImages_CWTLI.Files, 'UniformOutput', false);
valImageNames_CWTLI = cellfun(@(x) getFileNameFromPath(x), valImages_CWTLI.Files,
'UniformOutput', false);
```

```
testImageNames_CWTLI = cellfun(@(x) getFileNameFromPath(x), testImages_CWTLI.Files,
'UniformOutput', false);
% Add the 'ImageName' column from ensemble to FeatureTable1_2
FeatureTable1_2.ImageName = cellfun(@(x) getFileNameFromPath(x), ensemble.ImagePath,
'UniformOutput', false);
% Split FeatureTable1_2 into matching datasets based on image file names
% 3 columns were added here (image full path, index and image name)
matchingTrainData_FeatureTable1_2 =
FeatureTable1_2(ismember(FeatureTable1_2.ImageName, trainImageNames_CWTLI), :);
matchingValData_FeatureTable1_2 =
FeatureTable1_2(ismember(FeatureTable1_2.ImageName, valImageNames_CWTLI), :);
matchingTestData_FeatureTable1_2 =
FeatureTable1_2(ismember(FeatureTable1_2.ImageName, testImageNames_CWTLI), :);

%sort ascending (important)
Now you have to make sure that the order in time and frequency domain features lables match
the CWT and GADF image datastore as they are organised byfolder name which means (all
healthy files first, all mild, all moderate, aalll severe) grouped by LI
testing_withGADF = horzcat(testImages_GADFLI.Labels,
categorical(matchingTestData_FeatureTable1_2.Severity))
testing_withCWT = horzcat(testImages_CWTLI.Labels,
categorical(matchingTestData_FeatureTable1_2.Severity))
%for classification learner combine training and validation
WNNTrainValData_FeatureTable1_2 = [matchingTrainData_FeatureTable1_2;
matchingValData_FeatureTable1_2];
% i got these two functions CubicSVM_4.12 and WideNeuralNetwork_4.29
5.  Load pre trained models
%seed 1
%load functions from classification learner
load('CubicSVM_4.12.mat') %function named CubicSVM_412 was traned on all 20 features
load('WideNeuralNetwork_4.29.mat')%function named WNN_429 was traned on all 10
features
% Step 1: Load the pre-trained AlexNet network and the WNN model (if not already loaded)
load('LI_ALEXNET_CWT.mat'); % will load trainedNetwork_3
```

```
%load GADF Alexnet (low accuracy do not load)
load('LI_ALEXNET_GADF.mat'); % will load trainedNetwork_4
%load GADF Alexnet (7/3/2024) will load trainedNetwork_AlexNEWGADF
load('LI_trained_ALexNet_March24.mat')
%load CWT AlexNet (12/2/2024)will load trainedNetwork_AlexNEWCWT
load('LI_ALexNet_CWT_March24.mat')
%seed 3
load('WNN_seed3.mat')
load('CubicSVM_seed3.mat')
%Alexnet
load('AlexNet_CWT_seed3.mat') % will load AlexNet_CWT_seed3
load('AlexNet_GADF_seed3.mat') % will load AlexNet_GADF_seed3
% seed 6
load('WNN_seed6.mat')
load('CubicSVM_seed6.mat')
%Alexnet
load('AlexNet_CWT_seed6.mat') % will load AlexNet_CWT_seed6
load('AlexNet_GADF_seed6.mat') % will load AlexNet_GADF_seed6
% seed 9
load('WNN_seed9.mat')
load('CubicSVM_seed9.mat')
%I'm here training the CWT
load('AlexNet_CWT_seed9.mat') % will load AlexNet_CWT_seed9
load('AlexNet_GADF_seed9.mat') % will load AlexNet_GADF_seed9
% seed 12
load('WNN_seed12.mat')
load('CubicSVM_seed12.mat')
%I'm here training the CWT
load('AlexNet_CWT_seed12.mat') % will load AlexNet_CWT_seed12
load('AlexNet_GADF_seed12.mat') % will load AlexNet_GADF_seed12
% seed 15
load('WNN_seed15.mat')
load('CubicSVM_seed15.mat')
%I'm here training the CWT
```

```
load('AlexNet_CWT_seed15.mat') % will load AlexNet_CWT_seed15
load('AlexNet_GADF_seed15.mat') % will load AlexNet_GADF_seed15
%seed 18 (here)I dud cnn not good do not include
load('WNN_seed18.mat')
load('CubicSVM_seed18.mat')
%I'm here training the CWT
load('AlexNet_CWT_seed18.mat') % will load AlexNet_CWT_seed15
load('AlexNet_GADF_seed18.mat') % will load AlexNet_GADF_seed15
%seed 21
load('WNN_seed21.mat')
load('CubicSVM_seed21.mat')
%I'm here training the CWT
load('AlexNet_CWT_seed21.mat') % will load AlexNet_CWT_seed15
load('AlexNet_GADF_seed21.mat') % will load AlexNet_GADF_seed15
%seed 24
load('WNN_seed24.mat')
load('CubicSVM_seed24.mat')
%I'm here training the CWT
load('AlexNet_CWT_seed24.mat') % will load AlexNet_CWT_seed15
load('AlexNet_GADF_seed24.mat') % will load AlexNet_GADF_seed15
%seed 27
load('WNN_seed27.mat')
load('CubicSVM_seed27.mat')

%I'm here training the CWT
load('AlexNet_CWT_seed27.mat') % will load AlexNet_CWT_seed15
load('AlexNet_GADF_seed27.mat') % will load AlexNet_GADF_seed15
%seed 30
load('WNN_seed30.mat')
load('CubicSVM_seed30.mat')
%I'm here training the CWT
load('AlexNet_CWT_seed30.mat') % will load AlexNet_CWT_seed15
load('AlexNet_GADF_seed30.mat') % will load AlexNet_GADF_seed15
```

6. Predections (single chanel)

```
%CWT (AlexNet)(change function name per seed function)
cwt_predictions_train = classify(AlexNet_CWT_seed30, trainImages_CWTLI);
cwt_predictions_val = classify(AlexNet_CWT_seed30, valImages_CWTLI);
cwt_predictions_test = classify(AlexNet_CWT_seed30, testImages_CWTLI);
cwt_scores_train = predict(AlexNet_CWT_seed30, trainImages_CWTLI);
cwt_scores_val = predict(AlexNet_CWT_seed30, valImages_CWTLI);
cwt_scores_test = predict(AlexNet_CWT_seed30, testImages_CWTLI);
%GADF (AlexNet)
GADF_predictions_train = classify(AlexNet_GADF_seed30, trainImages_GADFLI);
GADF_predictions_val = classify(AlexNet_GADF_seed30, valImages_GADFLI);
GADF_predictions_test = classify(AlexNet_GADF_seed30, testImages_GADFLI);
GADF_scores_train = predict(AlexNet_GADF_seed30,trainImages_GADFLI);
GADF_scores_val = predict(AlexNet_GADF_seed30, valImages_GADFLI);
GADF_scores_test = predict(AlexNet_GADF_seed30, testImages_GADFLI)
%For time and frequency domain features (CubicSVM) (change function name per seed
function):
[yfit_CubicSVM_train, CubicSVM_scores_train] =
CubicSVM_seed30.predictFcn(matchingTrainData_FeatureTable1_2(:, 1:end-1));
[yfit_CubicSVM_val, CubicSVM_scores_val] =
CubicSVM_seed30.predictFcn(matchingValData_FeatureTable1_2(:, 1:end-1));
[yfit_CubicSVM_test, CubicSVM_scores_test] =
CubicSVM_seed30.predictFcn(matchingTestData_FeatureTable1_2(:, 1:end-1));
%% For time and frequency domain features (WNN):(change function name per seed function):
% because it should matches the classification learner input)
%exclude the last added one columns for linking
[yfit_WNN_train, WNN_scores_train] =
WNN_seed30.predictFcn(matchingTrainData_FeatureTable1_2(:, 1:end-1));
[yfit_WNN_val, WNN_scores_val] =
WNN_seed30.predictFcn(matchingValData_FeatureTable1_2(:, 1:end-1));
[yfit_WNN_test, WNN_scores_test] =
WNN_seed30.predictFcn(matchingTestData_FeatureTable1_2(:, 1:end-1));
diagnosticFeatureDesigner
```

7. Testing (single Channel)

- CWT (AlexNet) single on testing dataset

```
YPred = cwt_predictions_test;
YTest = testImages_CWTLI.Labels;
% Calculate overall accuracy
accuracy = sum(YPred == YTest) / numel(YTest) * 100;
fprintf('Overall Accuracy: %.2f%%\n', accuracy);
% Calculate accuracy per fault type
faultTypes = unique(YTest);
numFaultTypes = numel(faultTypes);
accuracyPerFaultType = zeros(numFaultTypes, 1);

for i = 1:numFaultTypes
  currentFaultType = faultTypes(i);
  indices = YTest == currentFaultType;
  accuracyPerFaultType(i) = sum(YPred(indices) == currentFaultType) / sum(indices) * 100;

  fprintf('Accuracy for Fault Type %s: %.2f%%\n', currentFaultType,
accuracyPerFaultType(i));
end
% Calculate precision, recall, and F1-score
cm = confusionchart(YTest, YPred);
cm.ColumnSummary = 'column-normalized';
cm.RowSummary = 'row-normalized';
cm.Title = 'Confusion Matrix CWT_AlexNet_seed30)';
```

- GADF (AlexNet)

```
YPred = GADF_predictions_test;
YTest = testImages_GADFLI.Labels;
% Correct the spelling mistake in the predicted labels(essential)
Ypred_CubicSVM_corrected = strrep(YPred, 'Helthy', 'Healthy');
% Calculate overall accuracy
accuracy = sum(Ypred_CubicSVM_corrected == YTest) / numel(YTest) * 100;
fprintf('Overall Accuracy: %.2f%%\n', accuracy);
% Calculate accuracy per fault type
faultTypes = unique(YTest);
numFaultTypes = numel(faultTypes);
```

```
accuracyPerFaultType = zeros(numFaultTypes, 1);
for i = 1:numFaultTypes
  currentFaultType = faultTypes(i);
  indices = YTest == currentFaultType;
  accuracyPerFaultType(i) = sum(Ypred_CubicSVM_corrected(indices) == currentFaultType) /
sum(indices) * 100;

  fprintf('Accuracy for Fault Type %s: %.2f%%\n', currentFaultType,
accuracyPerFaultType(i));
end

% Calculate precision, recall, and F1-score
cm = confusionchart(YTest, Ypred_CubicSVM_corrected);
cm.ColumnSummary = 'column-normalized';
cm.RowSummary = 'row-normalized';
cm.Title = 'Confusion Matrix GADF_ AlexNet_seed30';
```

- WNN Features

```
Ypred_WNN= yfit_WNN_test;
% Correct the spelling mistake in the predicted labels(essential)
Ypred_WNN_corrected = strrep(Ypred_WNN, 'Helthy', 'Healthy');
YTest_WNN = matchingTestData_FeatureTable1_2.Severity; %arranged with healthy first
% Calculate overall accuracy
accuracy = sum(Ypred_WNN_corrected == YTest_WNN) / numel(YTest_WNN) * 100;
fprintf('Overall Accuracy WNN: %.2f%%\n', accuracy);
% Calculate accuracy per fault type
faultTypes = unique(YTest_WNN);
numFaultTypes = numel(faultTypes);
accuracyPerFaultType = zeros(numFaultTypes, 1);
for i = 1:numFaultTypes
  currentFaultType = faultTypes(i);
  indices = YTest_WNN == currentFaultType;
  accuracyPerFaultType(i) = sum(Ypred_WNN_corrected(indices) == currentFaultType) /
sum(indices) * 100;
```

```
  fprintf('Accuracy for Fault Type WNN %s: %.2f%%\n', currentFaultType,
accuracyPerFaultType(i));
end
%here i want to make sure the testing label of the images from the datastore
%and for WNN is the same
testing = horzcat(YTest, categorical(YTest_WNN))
```

- CubicSVM Features

```
Ypred_CubicSVM= yfit_CubicSVM_test;
Ypred_WNN_corrected = strrep(Ypred_CubicSVM, 'Helthy', 'Healthy');
YTest_CubicSVM = matchingTestData_FeatureTable1_2.Severity;
% Calculate overall accuracy
accuracy = sum(Ypred_WNN_corrected == YTest_CubicSVM) / numel(YTest_CubicSVM) *
100;
fprintf('Overall Accuracy CubicSVM: %.2f%%\n', accuracy);
% Calculate accuracy per fault type
faultTypes = unique(YTest_CubicSVM);
numFaultTypes = numel(faultTypes);
accuracyPerFaultType = zeros(numFaultTypes, 1);
for i = 1:numFaultTypes
  currentFaultType = faultTypes(i);
  indices = YTest_CubicSVM == currentFaultType;
  accuracyPerFaultType(i) = sum(Ypred_WNN_corrected(indices) == currentFaultType) /
sum(indices) * 100;
   fprintf('Accuracy for Fault Type CubicSVM %s: %.2f%%\n', currentFaultType,
accuracyPerFaultType(i));
end
%testing (should match)
testing = horzcat(YTest, categorical(YTest_CubicSVM))
```

8. Decsion Fusion Altenatives

1. Alternative 1: CWT(AlexNet)-WNN4.29 (change this)

```
%1.1% Define the weight factors for each model's predictions(1.2)
time_freq_weight_mild = 0.1;
time_freq_weight_moderate = 0.9; % Adjust this weight as needed
```

```
time_freq_weight_severe = 0.1; % Adjust this weight as needed
CWT_weight_mild = 0.9;  % Weight for CWT AlexNet's predictions for Mild Load Index
CWT_weight_moderate = 0.1; % Adjust this weight as needed
CWT_weight_severe = 0.9; % Weight for CWT AlexNet's predictions for Severe Load Index
num_classes = 4;
% Get the labels for the testing dataset
test_labels = matchingTestData_FeatureTable1_2.Severity;
% Initialize an array to store the fused scores
fused_scores_test = zeros(length(test_labels), num_classes); % num_classes is the number of
classes
% Iterate through each sample and apply fusion based on class
for i = 1:length(test_labels)
  % Get the Load Index label for the current sample
  load_index_label = char(test_labels(i));

  % Assign weights based on the Load Index label
  if strcmp(load_index_label, 'Mild')
    time_freq_weight = time_freq_weight_mild;
    CWT_weight = CWT_weight_mild;

  elseif strcmp(load_index_label, 'Moderate')
    time_freq_weight = time_freq_weight_moderate;
    CWT_weight = CWT_weight_moderate;

  elseif strcmp(load_index_label, 'Severe')
    time_freq_weight = time_freq_weight_severe;
    CWT_weight = CWT_weight_severe;

  else % Healthy named in the enesemble helthy that is why it is zero
    time_freq_weight = 1;
    CWT_weight = 0; % No contribution from CWT for Healthy

  end
```

```
  % Calculate the fused score using adjusted weights
  fused_scores_test(i, :) = time_freq_weight * WNN_scores_test(i, :) ...
            + CWT_weight * cwt_scores_test(i, :);

end
% Get the final predicted labels based on the highest score for each sample
[~, final_predictions_test] = max(fused_scores_test, [], 2);
%1.2% Define the weight factors for each model's predictions(1.2)
time_freq_weight_mild = 0.5;
time_freq_weight_moderate = 0.5; % Adjust this weight as needed
time_freq_weight_severe = 0.5; % Adjust this weight as needed
CWT_weight_mild = 0.5;  % Weight for CWT AlexNet's predictions for Mild Load Index
CWT_weight_moderate = 0.5; % Adjust this weight as needed
CWT_weight_severe = 0.5; % Weight for CWT AlexNet's predictions for Severe Load Index
num_classes = 4;
% Get the labels for the testing dataset
test_labels = matchingTestData_FeatureTable1_2.Severity;
% Initialize an array to store the fused scores
fused_scores_test = zeros(length(test_labels), num_classes); % num_classes is the number of
classes
% Iterate through each sample and apply fusion based on class
for i = 1:length(test_labels)
  % Get the Load Index label for the current sample
  load_index_label = char(test_labels(i));

  % Assign weights based on the Load Index label
  if strcmp(load_index_label, 'Mild')
    time_freq_weight = time_freq_weight_mild;
    CWT_weight = CWT_weight_mild;

  elseif strcmp(load_index_label, 'Moderate')
    time_freq_weight = time_freq_weight_moderate;
    CWT_weight = CWT_weight_moderate;
```

```
 elseif strcmp(load_index_label, 'Severe')
  time_freq_weight = time_freq_weight_severe;
  CWT_weight = CWT_weight_severe;

 else % Healthy named in the enesemble helthy that is why it is zero
  time_freq_weight = 1;
  WNN_weight = 0; % No contribution from CWT for Healthy

 end
 % Calculate the fused score using adjusted weights
 fused_scores_test(i, :) = time_freq_weight * WNN_scores_test(i, :) ...
          + CWT_weight * cwt_scores_test(i, :);

end
% Get the final predicted labels based on the highest score for each sample
[~, final_predictions_test] = max(fused_scores_test, [], 2);
2.Alternative 2: CWT(AlexNet)-CubicSVM4.29 (change this )
%2.1 Define the weight factors for each model's predictions
time_freq_weight_mild = 0.1;
time_freq_weight_moderate = 0.1; % Adjust this weight as needed
time_freq_weight_severe = 0.9; % Adjust this weight as needed
cwt_weight_mild = 0.9;  % Weight for CWT AlexNet's predictions for Mild Load Index
cwt_weight_moderate = 0.9; % Adjust this weight as needed
cwt_weight_severe = 0.1; % Weight for CWT AlexNet's predictions for Severe Load Index
num_classes = 4;
% Get the labels for the testing dataset
test_labels = matchingTestData_FeatureTable1_2.Severity;
% Initialize an array to store the fused scores
fused_scores_test = zeros(length(test_labels), num_classes); % num_classes is the number of
classes
% Iterate through each sample and apply fusion based on class
for i = 1:length(test_labels)
 % Get the Load Index label for the current sample
 load_index_label = char(test_labels(i));
```

```
 % Assign weights based on the Load Index label
 if strcmp(load_index_label, 'Mild')
   time_freq_weight = time_freq_weight_mild;
   cwt_weight = cwt_weight_mild;

 elseif strcmp(load_index_label, 'Moderate')
   time_freq_weight = time_freq_weight_moderate;
   cwt_weight = cwt_weight_moderate;

 elseif strcmp(load_index_label, 'Severe')
   time_freq_weight = time_freq_weight_severe;
   cwt_weight = cwt_weight_severe;

 else % Healthy named in the enesemble helthy that is why it is zero
   time_freq_weight = 1;
   cwt_weight = 0; % No contribution from CWT for Healthy

 end
   % Calculate the fused score using adjusted weights
 fused_scores_test(i, :) = time_freq_weight * CubicSVM_scores_test(i, :) ...
           + cwt_weight * cwt_scores_test(i, :);

end
% Get the final predicted labels based on the highest score for each sample
[~, final_predictions_test] = max(fused_scores_test, [], 2);
% 2.2Define the weight factors for each model's predictions
time_freq_weight_mild = 0.5;
time_freq_weight_moderate = 0.5; % Adjust this weight as needed
time_freq_weight_severe = 0.5; % Adjust this weight as needed
cwt_weight_mild = 0.5;  % Weight for CWT AlexNet's predictions for Mild Load Index
cwt_weight_moderate = 0.5; % Adjust this weight as needed
cwt_weight_severe = 0.5; % Weight for CWT AlexNet's predictions for Severe Load Index
num_classes = 4;
```

```
% Get the labels for the testing dataset
test_labels = matchingTestData_FeatureTable1_2.Severity;
% Initialize an array to store the fused scores
fused_scores_test = zeros(length(test_labels), num_classes); % num_classes is the number of
classes
% Iterate through each sample and apply fusion based on class
for i = 1:length(test_labels)
  % Get the Load Index label for the current sample
  load_index_label = char(test_labels(i));
    % Assign weights based on the Load Index label
  if strcmp(load_index_label, 'Mild')
    time_freq_weight = time_freq_weight_mild;
    cwt_weight = cwt_weight_mild;

  elseif strcmp(load_index_label, 'Moderate')
    time_freq_weight = time_freq_weight_moderate;
    cwt_weight = cwt_weight_moderate;

  elseif strcmp(load_index_label, 'Severe')
    time_freq_weight = time_freq_weight_severe;
    cwt_weight = cwt_weight_severe;

  else % Healthy named in the enesemble helthy that is why it is zero
    time_freq_weight = 1;
    cwt_weight = 0; % No contribution from CWT for Healthy

  end
    % Calculate the fused score using adjusted weights
  fused_scores_test(i, :) = time_freq_weight * CubicSVM_scores_test(i, :) ...
            + cwt_weight * cwt_scores_test(i, :);

end
% Get the final predicted labels based on the highest score for each sample
[~, final_predictions_test] = max(fused_scores_test, [], 2);
```

```
%Extra to handel severe
% Define the weight factors for each model's predictions
time_freq_weight_mild = 0.1;
time_freq_weight_moderate = 0.1; % Adjust this weight as needed
time_freq_weight_severe = 0.5; % Adjust this weight as needed
cwt_weight_mild = 0.9;  % Weight for CWT AlexNet's predictions for Mild Load Index
cwt_weight_moderate = 0.9; % Adjust this weight as needed
cwt_weight_severe = 0.5; % Weight for CWT AlexNet's predictions for Severe Load Index

num_classes = 4;
% Get the labels for the testing dataset
test_labels = matchingTestData_FeatureTable1_2.Severity;
% Initialize an array to store the fused scores
fused_scores_test = zeros(length(test_labels), num_classes); % num_classes is the number of
classes
% Iterate through each sample and apply fusion based on class
for i = 1:length(test_labels)
  % Get the Load Index label for the current sample
  load_index_label = char(test_labels(i));

  % Assign weights based on the Load Index label
  if strcmp(load_index_label, 'Mild')
    time_freq_weight = time_freq_weight_mild;
    cwt_weight = cwt_weight_mild;
    gadf_weight = gadf_weight_mild;
  elseif strcmp(load_index_label, 'Moderate')
    time_freq_weight = time_freq_weight_moderate;
    cwt_weight = cwt_weight_moderate;
    gadf_weight = gadf_weight_moderate;
  elseif strcmp(load_index_label, 'Severe')
    time_freq_weight = time_freq_weight_severe;
    cwt_weight = cwt_weight_severe;
    gadf_weight = gadf_weight_severe;
  else % Healthy named in the enesemble helthy that is why it is zero
```

```
   time_freq_weight = 1;
   cwt_weight = 0; % No contribution from CWT for Healthy
   gadf_weight = 0; % No contribution from GADF for Healthy
  end
   % Calculate the fused score using adjusted weights
  fused_scores_test(i, :) = time_freq_weight * CubicSVM_scores_test(i, :) ...
            + cwt_weight * cwt_scores_test(i, :) ...
           ;
end
% Get the final predicted labels based on the highest score for each sample
[~, final_predictions_test] = max(fused_scores_test, [], 2);

3. GADF, CWT and CubicSVM
%3.1
% Define the weight factors for each model's predictions
time_freq_weight_mild = 0.1;
time_freq_weight_moderate = 0.1; % Adjust this weight as needed
time_freq_weight_severe = 0.8; % Adjust this weight as needed
cwt_weight_mild = 0.8;  % Weight for CWT AlexNet's predictions for Mild Load Index
cwt_weight_moderate = 0.1; % Adjust this weight as needed
cwt_weight_severe = 0.1; % Weight for CWT AlexNet's predictions for Severe Load Index

gadf_weight_mild = 0.1;  % Weight for GADF AlexNet's predictions for Mild Load Index
gadf_weight_moderate =0.8; % Adjust this weight as needed
gadf_weight_severe =0.1; % Weight for GADF AlexNet's predictions for Severe Load Index
num_classes = 4;
% Get the labels for the testing dataset
test_labels = matchingTestData_FeatureTable1_2.Severity;
% Initialize an array to store the fused scores
fused_scores_test = zeros(length(test_labels), num_classes); % num_classes is the number of
classes
% Iterate through each sample and apply fusion based on class
for i = 1:length(test_labels)
  % Get the Load Index label for the current sample
```

```
 load_index_label = char(test_labels(i));
   % Assign weights based on the Load Index label
 if strcmp(load_index_label, 'Mild')
   time_freq_weight = time_freq_weight_mild;
   cwt_weight = cwt_weight_mild;
   gadf_weight = gadf_weight_mild;
 elseif strcmp(load_index_label, 'Moderate')
   time_freq_weight = time_freq_weight_moderate;
   cwt_weight = cwt_weight_moderate;
   gadf_weight = gadf_weight_moderate;
 elseif strcmp(load_index_label, 'Severe')
   time_freq_weight = time_freq_weight_severe;
   cwt_weight = cwt_weight_severe;
   gadf_weight = gadf_weight_severe;
 else % Healthy named in the enesemble helthy that is why it is zero
   time_freq_weight = 1;
   cwt_weight = 0; % No contribution from CWT for Healthy
   gadf_weight = 0; % No contribution from GADF for Healthy
 end
   % Calculate the fused score using adjusted weights
 fused_scores_test(i, :) = time_freq_weight * CubicSVM_scores_test(i, :) ...
           + cwt_weight * cwt_scores_test(i, :) ...
           + gadf_weight * GADF_scores_test(i, :);
end
% Get the final predicted labels based on the highest score for each sample
[~, final_predictions_test] = max(fused_scores_test, [], 2);
%3.2
% Define the weight factors for each model's predictions
time_freq_weight_mild = 0.33;
time_freq_weight_moderate = 0.33; % Adjust this weight as needed
time_freq_weight_severe = 0.33; % Adjust this weight as needed
cwt_weight_mild = 0.33;  % Weight for CWT AlexNet's predictions for Mild Load Index
cwt_weight_moderate = 0.33; % Adjust this weight as needed
cwt_weight_severe = 0.33; % Weight for CWT AlexNet's predictions for Severe Load Index
```

```
gadf_weight_mild = 0.33;  % Weight for GADF AlexNet's predictions for Mild Load Index
gadf_weight_moderate =0.33; % Adjust this weight as needed
gadf_weight_severe =0.33; % Weight for GADF AlexNet's predictions for Severe Load Index
num_classes = 4;
% Get the labels for the testing dataset
test_labels = matchingTestData_FeatureTable1_2.Severity;
% Initialize an array to store the fused scores
fused_scores_test = zeros(length(test_labels), num_classes); % num_classes is the number of
classes
% Iterate through each sample and apply fusion based on class
for i = 1:length(test_labels)
  % Get the Load Index label for the current sample
  load_index_label = char(test_labels(i));

  % Assign weights based on the Load Index label
  if strcmp(load_index_label, 'Mild')
    time_freq_weight = time_freq_weight_mild;
    cwt_weight = cwt_weight_mild;
    gadf_weight = gadf_weight_mild;
  elseif strcmp(load_index_label, 'Moderate')
    time_freq_weight = time_freq_weight_moderate;
    cwt_weight = cwt_weight_moderate;
    gadf_weight = gadf_weight_moderate;
  elseif strcmp(load_index_label, 'Severe')
    time_freq_weight = time_freq_weight_severe;
    cwt_weight = cwt_weight_severe;
    gadf_weight = gadf_weight_severe;
  else % Healthy named in the enesemble helthy that is why it is zero
    time_freq_weight = 1;
    cwt_weight = 0; % No contribution from CWT for Healthy
    gadf_weight = 0; % No contribution from GADF for Healthy
  end

  % Calculate the fused score using adjusted weights
```

```
  fused_scores_test(i, :) = time_freq_weight * CubicSVM_scores_test(i, :) ...
            + cwt_weight * cwt_scores_test(i, :) ...
            + gadf_weight * GADF_scores_test(i, :);
end

% Get the final predicted labels based on the highest score for each sample
[~, final_predictions_test] = max(fused_scores_test, [], 2);
YPred_fused = final_predictions_test; % Use the fused predictions here
YTest = categorical(matchingTestData_FeatureTable1_2.Severity); % Convert YTest to
categorical
% Convert indices to categorical labels
YPred_fused = categorical(faultTypes(YPred_fused), faultTypes);
% Calculate overall accuracy
accuracy_fused = sum(YPred_fused == YTest) / numel(YTest) * 100;
fprintf('Overall Accuracy with Decision Fusion: %.2f%%\n', accuracy_fused);
% Calculate accuracy per fault type
faultTypes = unique(YTest);
numFaultTypes = numel(faultTypes);
accuracyPerFaultType_fused = zeros(numFaultTypes, 1);
for i = 1:numFaultTypes
  currentFaultType = faultTypes(i);
  indices = YTest == currentFaultType;
  accuracyPerFaultType_fused(i) = sum(YPred_fused(indices) == currentFaultType) /
sum(indices) * 100;
    fprintf('Accuracy for Fault Type %s with Decision Fusion: %.2f%%\n', currentFaultType,
accuracyPerFaultType_fused(i));
end
% Create a confusion chart
cm = confusionchart(YTest, YPred_fused);
cm.Normalization = 'row-normalized'; % Set the normalization to row-normalized
cm.Title = 'Confusion Matrix with Decision Fusion 3.2';
Functions
% Function to extract file name from full path (because in the enesemble
% the full path not similar it is better to use image name)
```

```
function fileName = getFileNameFromPath(fullPath)
  [~, fileName, ~] = fileparts(fullPath);
end
```

Appendix 3: Google Colab Codes

3.1 Chapter 4: DCGAN

```
from google.colab import drive
drive.mount('/content/drive')
Mounted at /content/drive
#to know the used GPU
import torch
if torch.cuda.is_available():
  device = torch.cuda.get_device_name(0)
  print(f'Using GPU: {device}')
else:
  print('GPU is not available. Using CPU instead.')
Using GPU: Tesla T4
##Import libraries + dataLoaders
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
import numpy as np
from tensorflow.keras.models import Sequential
# Define the data generator for preprocessing and data augmentation
data_generator = ImageDataGenerator(
  rescale=1./255, # normalize pixel values between 0 and 1
)
# Load the images from the directory
train_dir =
'/content/drive/MyDrive/MyResearch/GAN_Trials_March23/MotorImage_train'
train_generator = data_generator.flow_from_directory(
  train_dir,
  target_size=(224, 224),
  batch_size=32,
  classes=['inner'],
  class_mode='binary',
```

```
)
print(train_generator[0][0].shape)
Found 288 images belonging to 1 classes.
(32, 224, 224, 3)
from keras.layers import Input, Dense, Reshape, Flatten, Dropout,
BatchNormalization, Activation, ZeroPadding2D
from keras.layers.convolutional import UpSampling2D, Conv2D
from keras.models import Sequential, Model
from keras.optimisers import Adam
from keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
from keras.layers import LeakyReLU
from keras import layers
import numpy as np
import os
import matplotlib.pyplot as plt
import argparse
##Build Models
# Define the Generator Model
def build_generator():
  model = Sequential()
  model.add(Dense(128 * 56 * 56, activation="relu", input_dim=100))
  model.add(Reshape((56, 56, 128)))
  model.add(UpSampling2D())
  model.add(Conv2D(64, kernel_size=3, padding="same"))
  model.add(BatchNormalization(momentum=0.8))
  model.add(Activation("relu"))
  model.add(UpSampling2D())
  model.add(Conv2D(32, kernel_size=3, padding="same"))
  model.add(BatchNormalization(momentum=0.8))
  model.add(Activation("relu"))
```

```
  model.add(Conv2D(3, kernel_size=3, padding="same"))
  model.add(Activation("tanh"))
  noise = Input(shape=(100,))
  img = model(noise)
  return Model(noise, img)
# Define the Discriminator Model
def build_discriminator():
  model = Sequential()
  model.add(Conv2D(32, kernel_size=3, strides=2, input_shape=(224, 224, 3),
padding="same"))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Dropout(0.25))
  model.add(Conv2D(64, kernel_size=3, strides=2, padding="same"))
  model.add(ZeroPadding2D(padding=((0,1),(0,1))))
  model.add(BatchNormalization(momentum=0.8))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Dropout(0.25))
  model.add(Conv2D(128, kernel_size=3, strides=2, padding="same"))
  model.add(BatchNormalization(momentum=0.8))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Dropout(0.25))
  model.add(Conv2D(256, kernel_size=3, strides=1, padding="same"))
  model.add(BatchNormalization(momentum=0.8))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Dropout(0.25))
  model.add(Flatten())
  model.add(Dense(1, activation='sigmoid'))
  img = Input(shape=(224, 224, 3))
  validity = model(img)
  return Model(img, validity)
```

```
# Define the Adversarial Model
def build_adversarial(generator, discriminator):
  optimiser = Adam(0.0001, 0.5)
  discriminator.trainable = False
  gan_input = Input(shape=(100,))
  fake_image = generator(gan_input)
  validity = discriminator(fake_image)
  gan = Model(gan_input, validity)
  metrics = ['accuracy']
  gan.compile(loss='binary_crossentropy', optimiser=optimiser, metrics = metrics)
  return gan
# Set up the DCGAN
generator = build_generator()
discriminator = build_discriminator()
adversarial = build_adversarial(generator, discriminator)
discriminator.trainable = False
#The first thing to do is set trainable to false on our discriminator. This will prevent it
from updating its weights independently. This does not mean that the discriminator will
not learn though. We are going to add the discriminator and generator to another
network as components. Our discriminator will be able to update its weights in the
context of the GAN network, while the generator will be updated independently. This
will allow us to use the generator outside the context of the GAN to actually produce
synthetic samples.
generator.compile(loss='binary_crossentropy',
optimiser=tf.keras.optimisers.Adam(0.0001, 0.5))
# Compile the model discr
from keras.optimisers import Adam
optimiser = Adam(learning_rate=0.0001, beta_1=0.5)
loss_function = 'binary_crossentropy'
discriminator.compile(optimiser=optimiser, loss=loss_function, metrics=['accuracy'])
```

```
#compile
# Compile the model with the custom metric
from tensorflow.keras.optimisers import Adam
optimiser = Adam(learning_rate=0.0001, beta_1=0.5)
metrics = ['accuracy']
adversarial.compile(loss='binary_crossentropy', optimiser= optimiser, metrics =
metrics)
#summary
generator.summary()
discriminator.summary()
adversarial.summary()
##Model training
#where to save generated image s
save_path =
'/content/drive/MyDrive/MyResearch/GAN_Trials_March23/Inner_L0.0001'
import os
os.makedirs(save_path, exist_ok=True)
import numpy as np
import matplotlib.pyplot as plt
epochs = 50 # (changable)
batch_size = 32 #better to match the datagenerator batch_size
num_training_images = 288
save_interval = 9 #In this example, the generator model is saved and the generated
images are displayed every 27 epochs.
#288 // 32 = 9. So, a good choice for save_interval would be a multiple of 9, such as 9,
18, 27, 36, etc.
gen_loss = []
disc_loss = []
channels = 3
for epoch in range(epochs):
```

```
  for i in range(int(num_training_images/batch_size)):
    # Get batch of real images
    real_images, _ = train_generator.next()
    # Generate batch of fake images
    fake_images = generator.predict(np.random.normal(0, 1, (batch_size, 100)))
    # Train the discriminator
    discriminator_loss_real = discriminator.train_on_batch(real_images,
np.ones((batch_size, 1)))
    discriminator_loss_fake = discriminator.train_on_batch(fake_images,
np.zeros((batch_size, 1)))
    discriminator_loss = 0.5 * np.add(discriminator_loss_real, discriminator_loss_fake)
    # Train the generator
    gan_loss = adversarial.train_on_batch(np.random.normal(0, 1, (batch_size, 100)),
np.ones((batch_size, 1)))
  # Print the progress
  print("Epoch %d/%d [D loss: %s] [G loss: %s]" % (epoch+1, epochs,
discriminator_loss, gan_loss))
  # Append the losses to the corresponding lists
  gen_loss.append(gan_loss)
  disc_loss.append(discriminator_loss)
   # Show and save generated images every "save_interval" epochs
  if (epoch + 1) % save_interval == 0:
   save = True
  else:
   save = False
  show_images(generator, noise, epoch=epoch+1, save=save, save_path=save_path,
channels=channels)
# Save the generator model
generator.save(os.path.join(save_path,'Exp1_generator_model.h5'))
discriminator.save(os.path.join(save_path,'Exp1_discriminator_model.h5'))
```

3.2 Chapter 4: WGAN-GP and cWGAN-GP:

In thesis supplementary material file. complex file with nested structure.

3.3 Chapter 7: Full Code

```
from google.colab import drive
drive.mount('/content/drive')
!pip install torch-geometric
# Import necessary libraries
from google.colab import drive
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Dense, Conv1D, MaxPooling1D, Flatten
import numpy as np
from scipy.spatial.distance import cdist
import networkx as nx
import torch
from torch_geometric.utils.convert import from_networkx
# Mount Google Drive
drive.mount('/content/drive')
# Load dataset
dataset_path = '/content/drive/MyDrive/GNN Classification Task/FeatureTable1_2.xlsx'
df = pd.read_excel(dataset_path)
# Preprocess dataset
df.fillna(0, inplace=True)
label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(df.iloc[:, 0].values)
```

```
y = to_categorical(encoded_labels)

X = df.iloc[:, 1:].values

# balance the dataset classes per class without biase, then reassign variable (encoded_labels, y and X)

from imblearn.over_sampling import SMOTE

# Apply SMOTE to balance the dataset

smote = SMOTE(random_state=42)

X, encoded_labels = smote.fit_resample(X, encoded_labels)

# Convert labels to categorical

y = to_categorical(encoded_labels)

print(pd.Series(encoded_labels).value_counts())


#Create a new DataFrame with the balanced data

df = pd.DataFrame(data=X, columns=df.columns[1:])

df['Label'] = [np.argmax(row) for row in y]

# Get the encoded labels from the balanced dataset

encoded_labels = df['Label'].values

print("New encoded labels after balancing:", encoded_labels)


# 2. Set Random Seeds

import random

import numpy as np

import torch

# Set the random seed for reproducibility

seed_value = 42

random.seed(seed_value)

np.random.seed(seed_value)

torch.manual_seed(seed_value)

if torch.cuda.is_available():
```

```
  torch.cuda.manual_seed_all(seed_value)
  torch.backends.cudnn.deterministic = True
  torch.backends.cudnn.benchmark = False
print(df.columns)
!pip install --upgrade networkx
```

Dataset Preperation for CNN

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.optim import Adam
from sklearn.model_selection import train_test_split
import numpy as np
```

Dataset Splitting

```
from sklearn.model_selection import train_test_split
# Set random seeds for reproducibility
seed = 42
np.random.seed(seed)
torch.manual_seed(seed)
if torch.cuda.is_available():
  torch.cuda.manual_seed_all(seed)
```

 # Assuming 'encoded_labels' is your array of labels for the dataset (Same indeces will be used in GNN and CNN for train, valid and test)

```
# Split indices to maintain stratification
train_indices, temp_indices, y_train, y_temp = train_test_split(
  range(len(encoded_labels)), encoded_labels, stratify=encoded_labels, test_size=0.4, random_state=seed)
val_indices, test_indices, y_val, y_test = train_test_split(
  temp_indices, y_temp, stratify=y_temp, test_size=0.5, random_state=seed)
from collections import Counter
```

 # Check the distribution of classes in each set

print('Training set class distribution:', Counter(y_train))

print('Validation set class distribution:', Counter(y_val))

print('Test set class distribution:', Counter(y_test))

print(y_test)

print(test_indices)

import numpy as np

Assuming y_test is a numpy array of your test labels

unique, counts = np.unique(y_test, return_counts=True)

test_class_distribution = dict(zip(unique, counts))

print(test_class_distribution)

Dataset Preparation for CNN

#Step 1: Create Tensors for CNN

#First, ensure your feature matrix X and encoded_labels are converted to tensors, just like you did previously. Then, use the indices to create tensors for training, validation, and testing sets:

#here we add train dataset because we are not using graph and masks

X_tensor = torch.tensor(X, dtype=torch.float)

y_tensor = torch.tensor(encoded_labels, dtype=torch.long)

X_train, y_train = X_tensor[train_indices], y_tensor[train_indices]

X_val, y_val = X_tensor[val_indices], y_tensor[val_indices]

X_test, y_test = X_tensor[test_indices], y_tensor[test_indices]

#Step 2: Create DataLoaders for CNN

X_tensor = torch.tensor(X, dtype=torch.float)

y_tensor = torch.tensor(encoded_labels, dtype=torch.long)

X_train, y_train = X_tensor[train_indices], y_tensor[train_indices]

X_val, y_val = X_tensor[val_indices], y_tensor[val_indices]

X_test, y_test = X_tensor[test_indices], y_tensor[test_indices]

print(y_test)

how testing dataset is arranged for the cnn? is it arranged by test_indices?

Yes, the testing dataset for the CNN is arranged by the test_indices.

This ensures that the same data points are used for testing in both CNN and GNN models.

Here's how the test dataset is created:

X_test, y_test = X_tensor[test_indices], y_tensor[test_indices]

X_test contains the features for the test dataset,

and y_test contains the corresponding labels.

The order of data points in X_test and y_test is determined by the test_indices.

#GNN (K=3):

Dataset preperation for GNN

We need to create fraph and masks

import matplotlib.pyplot as plt

import networkx as nx

from torch_geometric.utils import to_networkx

from sklearn.neighbours import kneighbours_graph

import torch

from torch_geometric.data import Data

 # Necessary imports for converting sparse matrix

from scipy.sparse import coo_matrix

 # Step 1: Graph Construction

 # Convert features X into a graph

A = kneighbours_graph(X, n_neighbours=3, include_self=True) # Adjust n_neighbours based on your dataset

 # Convert the adjacency matrix to COO format

A_coo = coo_matrix(A)

 # Now, use the row and col attributes from the COO matrix

edge_index = torch.tensor([A_coo.row, A_coo.col], dtype=torch.long)

 # Create a PyTorch Geometric data object

```
data = Data(x=torch.tensor(X, dtype=torch.float), edge_index=edge_index,
y=torch.tensor(encoded_labels, dtype=torch.long))

from torch_geometric.utils import to_networkx

# Assuming 'data' is your PyTorch Geometric data object and it has 'y' for labels

G = to_networkx(data, to_undirected=True)

plt.figure(figsize=(15, 15)) # Increase figure size for better visibility

pos = nx.spring_layout(G, seed=42) # Layout for better node distribution

# Get unique classes and assign a distinct color to each class

classes = np.unique(data.y.cpu().numpy())

colors = plt.cm.rainbow(np.linspace(0, 1, len(classes)))

# Create a color map for nodes

class_color_map = {cls: colors[i] for i, cls in enumerate(classes)}

node_colors = [class_color_map[data.y[i].item()] for i in range(len(G))]

# Draw nodes with class-based color

nx.draw_networkx_nodes(G, pos, node_size=50, node_color=node_colors, alpha=0.8)

# Draw edges

nx.draw_networkx_edges(G, pos, alpha=0.1, edge_color="gray")

# Optionally, draw node labels for a subset or specific nodes for clarity

# For better clarity, consider labeling nodes of interest only

# subset_labels = {i: str(i) for i in subset_nodes}

# nx.draw_networkx_labels(G, pos, labels=subset_labels, font_size=8)

plt.title("Graph Visualization with Class Colors (KNN =3)")

plt.axis('off') # Turn off the axis

plt.show()
```

#Create Masks

How Masks Work

Masks are typically boolean arrays (or tensors) where each element corresponds to a node in your dataset:

True (1): If the element is True, the corresponding node is included in the operation (like training or evaluation).

False (0): If the element is False, the corresponding node is excluded from the operation.

For example, if you have a dataset with 100 nodes, and you want to train on the first 80 and test on the remaining 20, your training mask would be an array with the first 80 elements set to True and the rest set to False. The test mask would be the opposite.

#Masks creation For GNN

 # Initialize all masks to False initially

train_mask = torch.zeros(len(encoded_labels), dtype=torch.bool)

val_mask = torch.zeros(len(encoded_labels), dtype=torch.bool)

test_mask = torch.zeros(len(encoded_labels), dtype=torch.bool)

 # Set True for indices belonging to each split

train_mask[train_indices] = True

val_mask[val_indices] = True

test_mask[test_indices] = True

print(test_indices)

 # Attach masks to your data object

data.train_mask = train_mask

data.val_mask = val_mask

data.test_mask = test_mask

print(f"Training mask count: {train_mask.sum().item()}, Expected: {len(train_indices)}")

print(f"Validation mask count: {val_mask.sum().item()}, Expected: {len(val_indices)}")

print(f"Test mask count: {test_mask.sum().item()}, Expected: {len(test_indices)}")

data.y[data.test_mask]

print(data.y[data.test_mask])

 # Assuming 'label_encoder' is your LabelEncoder instance

original_class_labels = label_encoder.classes_

 # Print the mapping of encoded labels to original class labels

```
for encoded_label, original_label in enumerate(original_class_labels):
  print(f"Encoded label {encoded_label} stands for {original_label}")
#Define the GNN Model
import torch.nn.functional as F
from torch_geometric.nn import GCNConv
# Set a fixed random seed for reproducibility
seed = 42
np.random.seed(seed)
torch.manual_seed(seed)
if torch.cuda.is_available():
  torch.cuda.manual_seed_all(seed)
class GCN(torch.nn.Module):
  def __init__(self, num_features, num_classes):
    super(GCN, self).__init__()
    self.conv1 = GCNConv(num_features, 16)
    self.conv2 = GCNConv(16, num_classes)
  def forward(self, data):
    x, edge_index = data.x, data.edge_index
    x = F.relu(self.conv1(x, edge_index))
    x = F.dropout(x, training=self.training)
    x = self.conv2(x, edge_index)
    return F.log_softmax(x, dim=1)
model = GCN(num_features=data.num_features, num_classes=4)
# Assigning to a new variable name
GNN_k3 = model
#Train the Model
Actual Trainig With 5 cross validation (200 epochs, LR = 0.001, Adam optimiser)
import matplotlib.pyplot as plt
```

```
import torch
from torch_geometric.data import Data
from torch_geometric.nn import GCNConv
import torch.nn.functional as F
from torch.optim import Adam
import numpy as np
from sklearn.model_selection import train_test_split
#same for both
#Here Just check on the masks and ensure initial training results
from torch.optim import Adam
# Set a fixed random seed for reproducibility
seed = 40
np.random.seed(seed)
torch.manual_seed(seed)
if torch.cuda.is_available():
  torch.cuda.manual_seed_all(seed)


optimiser = Adam(model.parameters(), lr=0.0009, weight_decay=5e-4)
criterion = torch.nn.CrossEntropyLoss()
##GNN Training (K=3)
# Within the evaluate function, log the number of correct predictions and total number of predictions (Same for GNN k =3 and k =4) Before training each I will change its name and train it according to the K and save it again
def evaluate(mask):
  model.eval()
  with torch.no_grad():
    logits = model(data)
    preds = logits[mask].max(1)[1]
    correct = preds.eq(data.y[mask]).sum().item()
```

```
    total = mask.sum().item()
    acc = correct / total
    print(f"Correct: {correct}, Total: {total}, Mask sum: {mask.sum().item()}")
  return acc
#Training
from sklearn.model_selection import KFold
import random
def set_seed(seed_value):
    random.seed(seed_value)  # Python's built-in random module
    np.random.seed(seed_value)  # NumPy's random module
    torch.manual_seed(seed_value)  # PyTorch's random number generator
    if torch.cuda.is_available():
        torch.cuda.manual_seed(seed_value)  # CUDA's random number generator
        torch.cuda.manual_seed_all(seed_value)  # If you are using multi-GPU
        torch.backends.cudnn.deterministic = True  # To ensure that CUDA's convolution operations are deterministic
        torch.backends.cudnn.benchmark = False  # If your input sizes do not vary, setting this to False can improve reproducibility
# Set a seed for reproducibility
set_seed(seed_value=39)
# Define the number of folds for cross-validation
num_folds = 5
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
# Convert the dataset to a PyTorch tensor
X_tensor = torch.tensor(X, dtype=torch.float)
y_tensor = torch.tensor(encoded_labels, dtype=torch.long)
# Lists to store performance metrics
train_losses = []
train_accuracies = []
```

```
val_accuracies = []
num_epochs = len(train_accuracies)  # The number of epochs training was run
# Lists to store performance metrics for each fold
fold_train_losses = []
fold_train_accuracies = []
fold_val_accuracies = []
for fold, (train_index, val_index) in enumerate(kf.split(X_tensor)):
    # Split the data into training and validation sets for the current fold
    X_train, X_val = X_tensor[train_index], X_tensor[val_index]
    y_train, y_val = y_tensor[train_index], y_tensor[val_index]
    # Create PyTorch Geometric data objects for training and validation
    train_data = Data(x=X_train, y=y_train)
    val_data = Data(x=X_val, y=y_val)
    # Initialize your GNN model (Change here)
    optimizer = Adam(GNN_k3.parameters(), lr=0.0009, weight_decay=5e-4)
     # Train the model using the training data for the current fold
    for epoch in range(400):
        GNN_k3.train()
        optimizer.zero_grad()
        out = GNN_k3(data)
        loss = F.nll_loss(out[data.train_mask], data.y[data.train_mask])
        loss.backward()
        optimizer.step()
        # Evaluate the model using the validation data for the current fold
        train_acc = evaluate(data.train_mask)
        val_acc = evaluate(data.val_mask)
        # Store the results
        # Append training and validation accuracy to their respective lists
```

```
        train_accuracies.append(train_acc)
        val_accuracies.append(val_acc)
        fold_train_losses.append(loss.item())
        fold_train_accuracies.append(train_acc)
        fold_val_accuracies.append(val_acc)
        print(f'Fold: {fold}, Epoch: {epoch}, Loss: {loss.item():.4f}, Train Acc: {train_acc:.4f}, Val Acc: {val_acc:.4f}')
    # Here you can add code to reset the model, if needed, for the next fold
# Calculate and print the average performance across all folds
average_train_loss = sum(fold_train_losses) / len(fold_train_losses)
average_train_accuracy = sum(fold_train_accuracies) / len(fold_train_accuracies)
average_val_accuracy = sum(fold_val_accuracies) / len(fold_val_accuracies)
print(f'Average Train Loss: {average_train_loss:.4f}')
print(f'Average Train Accuracy: {average_train_accuracy:.4f}')
print(f'Average Validation Accuracy: {average_val_accuracy:.4f}')
# Plotting training and validation accuracy
plt.figure(figsize=(10, 6))
# Make sure to use the correct range based on the number of epochs trained
plt.plot(range(1, len(train_accuracies) + 1), train_accuracies, label='Training Accuracy')
plt.plot(range(1, len(val_accuracies) + 1), val_accuracies, label='Validation Accuracy')
plt.title('Training and Validation Accuracy over Epochs ')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
test_acc = evaluate(data.test_mask)
test_acc = evaluate(data.test_mask)
print(f'Test Accuracy: {test_acc:.4f}')
print(data.y[data.test_mask])
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
GNN_k3.eval()
total_correct = 0
total_samples = 0
with torch.no_grad():
    logits = GNN_k3(data)  # Forward pass
    predictions = logits[data.test_mask].max(1)[1]  # Get predicted classes
    true_labels = data.y[data.test_mask]  # True labels
    total_correct = (predictions == true_labels).sum().item()
    total_samples = data.test_mask.sum().item()
# Initialize a dictionary to store accuracy for each class
accuracy_per_class = {}
# Loop through each class
for class_index, class_name in enumerate(["Healthy", "Mild", "Moderate", "Severe"]):
    # Indices of true labels for the current class
    true_class_indices = (true_labels == class_index)
    # Total number of samples in the current class
    total_class_samples = true_class_indices.sum().item()
    if total_class_samples > 0:
        # Correct predictions for the current class
        correct_class_predictions = (predictions[true_class_indices] ==
true_labels[true_class_indices]).sum().item()
        class_accuracy = correct_class_predictions / total_class_samples
        accuracy_per_class[class_name] = class_accuracy
    else:
        accuracy_per_class[class_name] = None  # No samples for this class in the test set
```

```
# Calculate overall accuracy
overall_accuracy = total_correct / total_samples
# Print the accuracy for each class
for class_name, class_accuracy in accuracy_per_class.items():
    if class_accuracy is not None:
        print(f"Accuracy for {class_name}: {class_accuracy:.4f}")
    else:
        print(f"No samples for class {class_name} in the test set.")
# Print overall accuracy
print(f"Overall Accuracy: {overall_accuracy:.4f}")
#Saving the Best performing model after testing
model_save_path = '/content/drive/MyDrive/GNN Classification Task/GNN_k3.pth'
# Save the model state dictionary
torch.save(GNN_k3.state_dict(), 'GNN_k3.pt')
#verify it was saved
# Load the state dictionary
GNN_k3.load_state_dict(torch.load('/content/drive/MyDrive/GNN Classification
Task/GNN_k3.pth'))
#Same training for GNN (K=4 and K=5)


# 1D-CNN:
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.optim import Adam
from torch.utils.data import DataLoader, TensorDataset
from sklearn.model_selection import KFold
import numpy as np
import matplotlib.pyplot as plt
```

```
import random
from sklearn.model_selection import KFold
from sklearn.model_selection import KFold
# Define the number of folds and create the KFold instance
num_folds = 5
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
# Set random seeds for reproducibility
seed = 42
np.random.seed(seed)
torch.manual_seed(seed)
if torch.cuda.is_available():
  torch.cuda.manual_seed_all(seed)
# Define the model
class Simple1DCNN(nn.Module):
  def __init__(self, num_features, num_classes):
    super(Simple1DCNN, self).__init__()
    self.conv1 = nn.Conv1d(in_channels=1, out_channels=16, kernel_size=3, stride=1,
padding=1)
    self.pool = nn.MaxPool1d(kernel_size=2, stride=2)
    self.conv2 = nn.Conv1d(in_channels=16, out_channels=32, kernel_size=3, stride=1,
padding=1)
    self.flatten = nn.Flatten()
    # Temporarily set a placeholder value for the number of input features to fc1
    self.fc1 = nn.Linear(1, 120) # Placeholder, will be updated dynamically
    self.fc2 = nn.Linear(120, num_classes)

    # Dynamically calculate the correct input size for fc1
    self._init_fc1(num_features)
  def _init_fc1(self, num_features):
```

```
    dummy_input = torch.zeros((1, num_features), dtype=torch.float)
    output = self.pool(F.relu(self.conv1(dummy_input.unsqueeze(1))))
    output = self.pool(F.relu(self.conv2(output)))
    output_size = output.view(-1).size(0)
    self.fc1 = nn.Linear(output_size, 120)
  def forward(self, x):
    x = x.unsqueeze(1) # Add a channel dimension
    x = F.relu(self.conv1(x))
    x = self.pool(x)
    x = F.relu(self.conv2(x))
    x = self.pool(x)
    x = self.flatten(x)
    x = F.relu(self.fc1(x))
    x = self.fc2(x)
    return x
# Function to reset model weights
def reset_weights(m):
  if isinstance(m, nn.Conv1d) or isinstance(m, nn.Linear):
    m.reset_parameters()
# Prepare for training
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
num_features = X_tensor.shape[1]
num_classes = 4
CNN_model = Simple1DCNN(num_features=num_features, num_classes=num_classes)
CNN_model.to(device)


# Load the saved model state dictionary
model_state_dict = torch.load('/content/drive/MyDrive/GNN Classification
Task/CNN_model.pth')# Load the state dictionary into the model
```

```
CNN_model.load_state_dict(model_state_dict)


import torch
import torch.nn as nn
# Assuming CNN_model is your model instance
CNN_model = Simple1DCNN(num_features=100, num_classes=4) # Adjust num_features accordingly
def print_model_summary(CNN_model):
  print("Model Summary:\n")
  print("{:<25} {:<25} {:<15}".format("Layer Type", "Output Shape", "Param #"))
  print("="*65)
  total_params = 0
  for layer in CNN_model.modules():
    # Skip the overall model container
    if isinstance(layer, nn.Module) and not isinstance(layer, nn.Sequential) and not isinstance(layer, Simple1DCNN):
      layer_str = str(layer)
      layer_type = layer_str.split('(')[0]
      param_count = sum([p.numel() for p in layer.parameters()])
      total_params += param_count
      # For Conv1d and Linear layers, you can directly calculate the output shape
      # For other types of layers, you may need to adjust this logic
      if hasattr(layer, 'out_channels') and hasattr(layer, 'kernel_size'):
        output_shape = f"{layer.out_channels}, L" # L needs to be calculated based on your model's architecture
      elif isinstance(layer, nn.Linear):
        output_shape = str(layer.out_features)
      else:
        output_shape = "Variable"
```

```
    print("{:<25} {:<25} {:<15}".format(layer_type, output_shape, param_count))
  print("="*65)
  print(f"Total Params: {total_params}")
print_model_summary(CNN_model)

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
# Ensure that your test data is a torch Tensor
X_test = torch.tensor(X[test_indices], dtype=torch.float)
y_test = torch.tensor(encoded_labels[test_indices], dtype=torch.long)
# Now use X_test and y_test to create your test_dataset and test_loader
# Convert your test set to a TensorDataset
test_dataset = TensorDataset(X_test, y_test)
# Create a DataLoader for your test set
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
print(y_test)

import matplotlib.pyplot as plt
import numpy as np
# Initialize lists to hold the accuracies for all folds and epochs
all_folds_train_accuracies = []
all_folds_val_accuracies = []
# Cross-validation loop
for fold, (train_index, val_index) in enumerate(kf.split(X_tensor)):
  print(f'Starting fold {fold+1}')
  CNN_model.apply(reset_weights) # Reset model weights
```

```
  # Create separate datasets for training and validation
  X_train_fold, X_val_fold = X_tensor[train_index], X_tensor[val_index]
  y_train_fold, y_val_fold = y_tensor[train_index], y_tensor[val_index]
  # DataLoader setup for training and validation
  train_loader = DataLoader(TensorDataset(X_train_fold, y_train_fold), batch_size=64, shuffle=True)
  val_loader = DataLoader(TensorDataset(X_val_fold, y_val_fold), batch_size=64)
  # Prepare for training
  device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
  num_features = X_tensor.shape[1]
  num_classes = 4
  CNN_model= Simple1DCNN(num_features=num_features, num_classes=num_classes)
  CNN_model.to(device)


import numpy as np
import matplotlib.pyplot as plt
from torch.optim import Adam
import torch.nn as nn
# Assuming CNN_model, train_loader, val_loader, and device are already defined
# Optimiser and criterion
optimiser = Adam(CNN_model.parameters(), lr=0.005)
criterion = nn.CrossEntropyLoss()
# Lists to store metrics for all folds
all_folds_train_accuracies = []
all_folds_val_accuracies = []
# Assuming num_folds and fold are defined, and the training/validation loop is part of a larger cross-validation loop
for fold in range(num_folds):
  # Lists to store metrics for the current fold
```

```
  fold_train_accuracies = []
  fold_val_accuracies = []


  # Training loop for current fold
  for epoch in range(300): # Adjusted to match the loop range with the defined number of epochs
    # Training phase
    CNN_model.train()
    train_loss, train_correct, train_total = 0, 0, 0
    for X_batch, y_batch in train_loader:
      X_batch, y_batch = X_batch.to(device), y_batch.to(device)
      optimiser.zero_grad()
      outputs = CNN_model(X_batch)
      loss = criterion(outputs, y_batch)
      loss.backward()
      optimiser.step()
      _, predicted = torch.max(outputs.data, 1)
      train_total += y_batch.size(0)
      train_correct += (predicted == y_batch).sum().item()
      train_loss += loss.item()
    # Calculate training accuracy for the current epoch
    train_accuracy = train_correct / train_total
    fold_train_accuracies.append(train_accuracy)
    # Validation phase
    CNN_model.eval()
    val_loss, val_correct, val_total = 0, 0, 0
    with torch.no_grad():
      for X_batch, y_batch in val_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
```

```
        outputs = CNN_model(X_batch)
        _, predicted = torch.max(outputs, 1)
        val_total += y_batch.size(0)
        val_correct += (predicted == y_batch).sum().item()

    # Calculate validation accuracy for the current epoch
    val_accuracy = val_correct / val_total
    fold_val_accuracies.append(val_accuracy)
    # Output the results
    print(f'Fold: {fold+1}, Epoch: {epoch+1}, Loss: {train_loss / len(train_loader):.4f}, Train Acc: {train_accuracy:.4f}, Val Acc: {val_accuracy:.4f}')
  # Store the accuracies for the current fold
  all_folds_train_accuracies.append(fold_train_accuracies)
  all_folds_val_accuracies.append(fold_val_accuracies)
# Convert lists of lists to a NumPy array for easier handling
all_folds_train_accuracies = np.array(all_folds_train_accuracies)
all_folds_val_accuracies = np.array(all_folds_val_accuracies)
# Plot training and validation accuracies for each epoch and fold
for fold in range(num_folds):
  plt.figure(figsize=(10, 6))
  epochs_range = range(1, 301) # Corrected to match the actual range of epochs
  plt.plot(epochs_range, all_folds_train_accuracies[fold], label='Training Accuracy - Fold {}'.format(fold+1))
  plt.plot(epochs_range, all_folds_val_accuracies[fold], label='Validation Accuracy - Fold {}'.format(fold+1))
  plt.title(f'Training and Validation Accuracy over Epochs - Fold {fold+1}')
  plt.xlabel('Epoch')
  plt.ylabel('Accuracy')
  plt.legend()
```

```
  plt.show()



# Convert lists of lists to a NumPy array for easier handling
all_folds_train_accuracies = np.array(all_folds_train_accuracies)
all_folds_val_accuracies = np.array(all_folds_val_accuracies)
# Calculate the mean accuracies across folds for each epoch
mean_train_accuracies = all_folds_train_accuracies.mean(axis=0)
mean_val_accuracies = all_folds_val_accuracies.mean(axis=0)
# Calculate the overall average validation accuracy
overall_average_val_accuracy = mean_val_accuracies.mean()
# Print the overall average validation accuracy
print(f"Overall Average Validation Accuracy: {overall_average_val_accuracy:.4f}")

import matplotlib.pyplot as plt
import numpy as np
# Assuming 'all_folds_train_accuracies' and 'all_folds_val_accuracies' are lists of lists,
# where each sublist contains accuracies for one fold, and each fold has data for 300 epochs.
# Convert lists of lists to a NumPy array for easier handling
all_folds_train_accuracies = np.array(all_folds_train_accuracies)
all_folds_val_accuracies = np.array(all_folds_val_accuracies)
# Calculate the mean accuracies across folds for each epoch
mean_train_accuracies = all_folds_train_accuracies.mean(axis=0)
mean_val_accuracies = all_folds_val_accuracies.mean(axis=0)
# Plot training and validation accuracies for each epoch
plt.figure(figsize=(12, 6))
# Adjusting the range to 301 because Python ranges are exclusive at the upper bound, and we're counting from 1
```

```
plt.plot(range(1, 301), mean_train_accuracies, label='Average Training Accuracy')
plt.plot(range(1, 301), mean_val_accuracies, label='Average Validation Accuracy')
plt.title('Average Training and Validation Accuracy over Epochs for All Folds')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()


# testing dataset accuracy arranged by test_indeces similar to GNN and accuracy per class# Assuming CNN_model, X_test, y_test, and test_loader are already defined# Evaluate the model on the test set
CNN_model.eval()
with torch.no_grad():
    test_loss = 0
    test_correct = 0
    test_total = 0
    for X_batch, y_batch in test_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
        outputs = CNN_model(X_batch)
        loss = criterion(outputs, y_batch)
        test_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        test_total += y_batch.size(0)
        test_correct += (predicted == y_batch).sum().item()

    # Calculate test accuracy
    test_accuracy = test_correct / test_total
    print(f'Test Accuracy: {test_accuracy:.4f}')
```

```
    # Get predictions for the entire test set
    outputs = CNN_model(X_test.to(device))
    _, predicted = torch.max(outputs.data, 1)

    # Calculate confusion matrix
    cm = confusion_matrix(y_test.cpu().numpy(), predicted.cpu().numpy())
    sns.heatmap(cm, annot=True, fmt='d')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix')
    plt.show()

    # Calculate accuracy per class
    accuracy_per_class = {}
    for class_index, class_name in enumerate(["Healthy", "Mild", "Moderate", "Severe"]):
        true_class_indices = (y_test == class_index)
        total_class_samples = true_class_indices.sum().item()
        if total_class_samples > 0:
            correct_class_predictions = predicted[true_class_indices] ==
y_test[true_class_indices]
            class_accuracy = correct_class_predictions.sum().item() / total_class_samples
            accuracy_per_class[class_name] = class_accuracy
        else:
            accuracy_per_class[class_name] = None  # No samples for this class in the test set



model_save_path_cnn = '/content/drive/MyDrive/GNN Classification
Task/CNN_model.pth'
```

```
# Save the model state dictionary
#torch.save(CNN_model.state_dict(), model_save_path_cnn)

# Load the state dictionary to verify it was saved correctly
#CNN_model.load_state_dict(torch.load(model_save_path_cnn))
#print("Model saved and loaded successfully.")
```

#Hybrid CNN and GNN with optimal GNN Configuration using Taguchi

```
# prompt: Now on testing dataset compare GCN output (the model name is model) and CNN (the model name is CNN_model) output

# Assuming you have your GCN model and CNN model loaded and ready to use

# Get predictions from the GCN model
model.eval()
with torch.no_grad():
  gcn_logits = model(data)
  gcn_test_logits = gcn_logits[data.test_mask]
  gcn_preds = gcn_test_logits.max(1)[1]


# Get predictions from the CNN model
CNN_model.eval()
with torch.no_grad():
  cnn_outputs = CNN_model(X_test.to(device))
  _, cnn_predicted = torch.max(cnn_outputs.data, 1)

# Compare the predictions
print("GCN Predictions:", gcn_preds)
print("CNN Predictions:", cnn_predicted)
```

```
# Calculate the number of matching predictions
matching_predictions = (gcn_preds.cpu().numpy() == cnn_predicted.cpu().numpy()).sum()
print("Number of matching predictions:", matching_predictions)

# Calculate the number of differing predictions
differing_predictions = (gcn_preds.cpu().numpy() != cnn_predicted.cpu().numpy()).sum()
print("Number of differing predictions:", differing_predictions)
print("True labels:", true_labels)
# Assuming CNN_model, model, X_test, y_test, and test_loader are already defined

# Evaluate the model on the test set
CNN_model.eval()
model.eval()
with torch.no_grad():
    test_loss = 0
    test_correct = 0
    test_total = 0
    for X_batch, y_batch in test_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
        outputs = CNN_model(X_batch)
        loss = criterion(outputs, y_batch)
        test_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        test_total += y_batch.size(0)
        test_correct += (predicted == y_batch).sum().item()
```

```
    # Get predictions for the entire test set
    cnn_outputs = CNN_model(X_test.to(device))
    _, cnn_predicted = torch.max(cnn_outputs.data, 1)

    # Get predictions from the GCN model
    gcn_logits = model(data)
    gcn_test_logits = gcn_logits[data.test_mask]
    gcn_preds = gcn_test_logits.max(1)[1]

    # Calculate the weighted sum of predictions
    # Define weights for each class in CNN and GCN models
    gcn_weights_per_class = [0.5, 0.5, 0.5, 0.5]  # Example weights for each class in GCN
    cnn_weights_per_class = [0.5, 0.5, 0.5, 0.5]  # Example weights for each class in CNN


    # Apply weights to the logits
    weighted_cnn_outputs = cnn_outputs * torch.tensor(cnn_weights_per_class).to(device)
    weighted_gcn_test_logits = gcn_test_logits *
torch.tensor(gcn_weights_per_class).to(device)

    # Fuse the logits
    fused_logits = weighted_cnn_outputs + weighted_gcn_test_logits

    # Get the predicted classes from the fused logits
    _, fused_predictions = torch.max(fused_logits.data, 1)

    # Calculate the confusion matrix
    cm = confusion_matrix(y_test.cpu().numpy(), fused_predictions.cpu().numpy())
```

```
    sns.heatmap(cm, annot=True, fmt='d')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix (Decision Fusion-Equal weights)')
    plt.show()

    # Calculate accuracy per class
    accuracy_per_class = {}
    for class_index, class_name in enumerate(["Healthy", "Mild", "Moderate", "Severe"]):
        true_class_indices = (y_test == class_index)
        total_class_samples = true_class_indices.sum().item()
        if total_class_samples > 0:
            correct_class_predictions = fused_predictions[true_class_indices] ==
y_test[true_class_indices]
            class_accuracy = correct_class_predictions.sum().item() / total_class_samples
            accuracy_per_class[class_name] = class_accuracy
        else:
            accuracy_per_class[class_name] = None  # No samples for this class in the test set

    # Print the accuracy for each class
    for class_name, class_accuracy in accuracy_per_class.items():
        if class_accuracy is not None:
            print(f"Accuracy for {class_name}: {class_accuracy:.4f}")
        else:
            print(f"No samples for class {class_name} in the test set.")
    # Calculate the average test accuracy
total_accuracy = 0
for class_name, class_accuracy in accuracy_per_class.items():
    if class_accuracy is not None:
```

```
        total_accuracy += class_accuracy
average_test_accuracy = total_accuracy / len(accuracy_per_class)

print(f"Average Test Accuracy: {average_test_accuracy:.4f}")
# Evaluate the model on the test set
CNN_model.eval()
model.eval()
with torch.no_grad():
    test_loss = 0
    test_correct = 0
    test_total = 0
    for X_batch, y_batch in test_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
        outputs = CNN_model(X_batch)
        loss = criterion(outputs, y_batch)
        test_loss += loss.item()
        _, predicted = torch.max(outputs.data, 1)
        test_total += y_batch.size(0)
        test_correct += (predicted == y_batch).sum().item()


    # Get predictions for the entire test set
    cnn_outputs = CNN_model(X_test.to(device))
    _, cnn_predicted = torch.max(cnn_outputs.data, 1)

    # Get predictions from the GCN model
    gcn_logits = model(data)
    gcn_test_logits = gcn_logits[data.test_mask]
```

```
    gcn_preds = gcn_test_logits.max(1)[1]

    # Calculate the weighted sum of predictions
    # Define weights for each class in CNN and GCN models
    gcn_weights_per_class = [0.3, 0.6, 0.5, 0.5]  # Example weights for each class in GCN
    cnn_weights_per_class = [0.7, 0.4, 0.5, 0.5]  # Example weights for each class in CNN


    # Apply weights to the logits
    weighted_cnn_outputs = cnn_outputs * torch.tensor(cnn_weights_per_class).to(device)
    weighted_gcn_test_logits = gcn_test_logits *
torch.tensor(gcn_weights_per_class).to(device)

    # Fuse the logits
    fused_logits = weighted_cnn_outputs + weighted_gcn_test_logits

    # Get the predicted classes from the fused logits
    _, fused_predictions = torch.max(fused_logits.data, 1)

    # Calculate the confusion matrix
    cm = confusion_matrix(y_test.cpu().numpy(), fused_predictions.cpu().numpy())
    sns.heatmap(cm, annot=True, fmt='d')
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.title('Confusion Matrix (Decision Fusion)')
    plt.show()

    # Calculate accuracy per class
    accuracy_per_class = {}
```

```
    for class_index, class_name in enumerate(["Healthy", "Mild", "Moderate", "Severe"]):
        true_class_indices = (y_test == class_index)
        total_class_samples = true_class_indices.sum().item()
        if total_class_samples > 0:
            correct_class_predictions = fused_predictions[true_class_indices] ==
y_test[true_class_indices]
            class_accuracy = correct_class_predictions.sum().item() / total_class_samples
            accuracy_per_class[class_name] = class_accuracy
        else:
            accuracy_per_class[class_name] = None  # No samples for this class in the test set

    # Print the accuracy for each class
    for class_name, class_accuracy in accuracy_per_class.items():
        if class_accuracy is not None:
            print(f"Accuracy for {class_name}: {class_accuracy:.4f}")
        else:
            print(f"No samples for class {class_name} in the test set.")

# Calculate the average test accuracy
total_accuracy = 0
for class_name, class_accuracy in accuracy_per_class.items():
    if class_accuracy is not None:
        total_accuracy += class_accuracy
average_test_accuracy = total_accuracy / len(accuracy_per_class)

print(f"Average Test Accuracy: {average_test_accuracy:.4f}")
```

Appendix 4: Chapter 6 Extra Results

4.1 Chapter 6: Results of Decision Fusion Experiments Over the Five Runs.

Alternatives	No. of Channels	Training Time: TFD a.CubicSVM b.WNN (S)	Training Time: CWT AlexNet (min)	Training Time: GADF AlexNet (min)	Healthy	Mild	Moderate	Severe	Overall Accuracy	Total Training Time
Run 1 Seed 1										
1.1 (TFDb -CWT)	2	48.63	7.2	0	100.00%	94.68%	100.00%	100.00%	98.67%	0:08:01
1.2 (TFDb-CWT)	2	48.63	7.2	0	100.00%	95.74%	100.00%	100.00%	98.94%	0:08:01
2.1 (TFDa -CWT)	2	26.71	7.2	0	100.00%	95.74%	97.87%	100.00%	98.40%	0:07:39
2.2 (TFDa -CWT)	2	26.71	7.2	0	100.00%	96.81%	97.87%	100.00%	98.67%	0:07:39
3.1 (TFDa -CWT-GADF)	3	26.71	7.2	7.53	100.00%	95.74%	98.94%	100.00%	98.67%	0:15:11
3.2 (TFDa -CWT-GADF)	3	26.71	7.2	7.53	100.00%	95.74%	97.87%	100.00%	98.40%	0:15:11
Run 2 Seed 3										
1.1 (TFDb -CWT)	2	18.74	10.32	0	100.00%	95.74%	100.00%	100.00%	98.94%	0:10:38
1.2 (TFDb-CWT)	2	18.74	10.32	0	100.00%	90.43%	97.87%	100.00%	97.08%	0:10:38
2.1 (TFDa -CWT)	2	12.87	10.32	0	100.00%	96.81%	97.87%	100.00%	98.67%	0:10:32
2.2 (TFDa -CWT)	2	12.87	10.32	0	100.00%	96.81%	98.94%	100.00%	98.94%	0:10:32
3.1 (TFDa -CWT-GADF)	3	12.87	10.32	11.49	100.00%	95.74%	100.00%	100.00%	98.94%	0:22:01
3.2 (TFDa -CWT-GADF)	3	12.87	10.32	11.49	100.00%	95.74%	94.29%	100.00%	97.51%	0:22:01
Run 3 Seed 6										
1.1 (TFDb -CWT)	2	16.75	7.54	0	100.00%	95.74%	100.00%	100.00%	98.94%	0:07:49
1.2 (TFDb-CWT)	2	16.75	7.54	0	100.00%	94.68%	100.00%	100.00%	98.67%	0:07:49
2.1 (TFDa -CWT)	2	11.56	7.54	0	100.00%	90.43%	95.74%	100.00%	96.54%	0:07:44
2.2 (TFDa -CWT)	2	11.56	7.54	0	100.00%	92.55%	95.74%	100.00%	97.07%	0:07:44
3.1 (TFDa -CWT-GADF)	3	11.56	7.54	7.56	100.00%	98.94%	98.94%	100.00%	99.47%	0:15:18
3.2 (TFDa -CWT-GADF)	3	11.56	7.54	7.56	100.00%	96.81%	98.94%	100.00%	98.94%	0:15:18
Run 4 Seed 9										
1.1 (TFDb -CWT)	2	19.75	7.34	0	100.00%	94.68%	100.00%	100.00%	98.67%	0:07:40
1.2 (TFDb-CWT)	2	19.75	7.34	0	100.00%	94.68%	100.00%	100.00%	98.67%	0:07:40
2.1 (TFDa -CWT)	2	10.85	7.34	0	100.00%	95.74%	98.94%	100.00%	98.67%	0:07:31
2.2 (TFDa -CWT)	2	10.85	7.34	0	100.00%	95.74%	98.94%	100.00%	98.67%	0:07:31
3.1 (TFDa -CWT-GADF)	3	10.85	7.34	8.2	100.00%	95.74%	100.00%	100.00%	98.94%	0:15:43
3.2 (TFDa -CWT-GADF)	3	10.85	7.34	8.2	100.00%	95.74%	100.00%	100.00%	98.94%	0:15:43
Run 5 Seed 12										
1.1 (TFDb -CWT)	2	22.07	12.41	0	100.00%	96.81%	100.00%	100.00%	99.07%	0:12:47
1.2 (TFDb-CWT)	2	22.07	12.41	0	100.00%	95.74%	100.00%	100.00%	98.67%	0:12:47
2.1 (TFDa -CWT)	2	16.41	12.41	0	100.00%	93.62%	100.00%	100.00%	98.54%	0:12:41
2.2 (TFDa -CWT)	2	16.41	12.41	0	100.00%	94.68%	100.00%	100.00%	98.94%	0:12:41
3.1 (TFDa -CWT-GADF)	3	16.41	12.41	11.59	100.00%	98.94%	97.87%	100.00%	99.20%	0:24:16
3.2 (TFDa -CWT-GADF)	3	16.41	12.41	11.59	100.00%	97.87%	98.94%	100.00%	99.20%	0:24:16

4.2 Chapter 7: Feature Extraction and Selection for Data Points

Severity	CF	Crest Factor	Impulse Factor	Kurtosis	Mean	Peak Value	RMS	SINAD	S/N	Shape Factor	Skewness	Std	Band Power	Peak Amp1	Peak Amp2	Peak Amp3	Peak Amp4	Peak Amp5	Peak Freq1	Peak Freq3
Mild	22.99	8.66	16.1	19.29	-0.23	17.24	1.99	-10.3	-10.27	1.86	-0.02	1.98	1.81	4E-04	3E-04	3E-04	1E-04	1E-05	1.E+04	4476.56
Mild	27.36	10.84	19.75	27.36	-0.23	18.94	1.75	-14.25	-14.14	1.82	0.28	1.73	1.21	5E-04	3E-04	2E-04	2E-05	1E-05	4.E+03	9289.82
Mild	28.49	10.72	20.45	29.5	-0.22	19	1.77	-8.17	-8.17	1.91	0.35	1.76	1.37	8E-04	3E-04	1E-04	1E-05		9.E+03	14127.26
Mild	25.31	10.17	18.29	22.75	-0.22	16.28	1.6	-13.02	-12.79	1.8	0.36	1.59	1.18	3E-04	2E-04	1E-04	2E-05	9E-06	1.E+04	4509.91
Mild	19.36	7.71	13.77	17.41	-0.22	14.65	1.9	-8.77	-8.77	1.79	0.73	1.89	1.5	7E-04	3E-04	2E-04	1E-05	6E-06	4.E+03	13891.02
Mild	23.14	9.95	17.19	23.94	-0.24	15.53	1.56	-12.38	-12.34	1.73	0.96	1.54	0.95	4E-04	1E-04	7E-05	9E-06	7E-06	4.E+03	9298.51
Mild	25.72	9.88	18.47	29.52	-0.23	17.18	1.74	-10.79	-10.79	1.87	0.69	1.72	1.39	5E-04	3E-04	2E-04	1E-05	1E-05	9.E+03	4249
Mild	18.07	7.81	13.33	16.37	-0.22	11.81	1.51	-12.5	-12.5	1.71	-0.01	1.5	1.04	3E-04	2E-04	1E-04	2E-05	1E-05	1.E+04	8894.99
Mild	22.59	9.02	16.12	18.72	-0.25	17.81	1.97	-11.54	-11.5	1.79	0.57	1.96	1.49	8E-04	3E-04	1E-04	1E-05	1E-05	4.E+03	9355.02
Mild	29.84	11.87	21.71	34.27	-0.22	19.35	1.63	-12.36	-12.35	1.83	1.93	1.62	1.17	4E-04	3E-04	2E-04	1E-05	9E-06	1.E+04	9509.94
Mild	27.02	9.87	19.1	26.99	-0.23	18.41	1.86	-11.74	-11.74	1.93	0.18	1.85	1.65	7E-04	4E-04	8E-05	3E-05	7E-06	1.E+04	4654.49
Mild	19.12	8.4	14.25	18.78	-0.22	12.45	1.48	-12.46	-12.46	1.7	0.38	1.47	0.95	6E-04	1E-04	1E-04	1E-05	8E-06	5.E+03	14070.07
Mild	25.35	9.04	17.48	21.74	-0.22	20.02	2.21	-11.29	-11.25	1.93	0.59	2.2	2	7E-04	5E-04	2E-04	3E-05		4.E+03	14024.68
Mild	36.16	13.96	26.1	39.14	-0.23	23.1	1.65	-10.25	-10.25	1.87	-0.12	1.64	1.27	4E-04	2E-04	1E-04	1E-05	9E-06	9.E+03	3990.37
Mild	22.76	9.77	16.78	19.55	-0.22	14.15	1.45	-10.22	-10.22	1.72	0.48	1.43	0.93	2E-04	2E-04	8E-05	1E-05	8E-06	5.E+03	9666.18

Severity	CF	Crest Factor	Impulse Factor	Kurtosis	Mean	Peak Value	RMS	SINAD	S/N	Shape Factor	Skewness	Std	Band Power	Peak Amp1	Peak Amp2	Peak Amp3	Peak Amp4	Peak Amp5	Peak Freq1	Peak Freq3
Mild	23.94	10.29	17.62	22.18	-0.23	15.91	1.55	-11.59	-11.49	1.71	0.82	1.53	0.94	4E-04	1E-04	9E-05	1E-05	9E-06	4.E+03	9210.33
Mild	24.63	9.19	17.28	23.15	-0.23	18.39	2	-11.29	-11.22	1.88	1.08	1.99	1.67	7E-04	4E-04	1E-04	1E-05	9E-06	1.E+04	9521.39
Mild	29.71	10.84	20.85	31.2	-0.21	21.19	1.95	-11.83	-11.83	1.92	0.09	1.94	1.83	7E-04	2E-04	2E-04	1E-04	1E-05	1.E+04	4793.69
Mild	18.52	8.25	13.8	17.17	-0.19	12.25	1.48	-11.94	-11.94	1.67	0.65	1.47	0.91	4E-04	2E-04	5E-05	1E-05		4.E+03	9163.66
Mild	29.54	10.17	20.27	28.5	-0.2	21.21	2.09	-8.06	-8.05	1.99	0.66	2.08	1.87	7E-04	5E-04	1E-04	2E-05	1E-05	4.E+03	13820.42
Mild	23.29	9.07	16.58	23.34	-0.24	16.74	1.85	-11.45	-11.45	1.83	0.44	1.83	1.29	5E-04	2E-04	1E-04	2E-05	9E-06	1.E+04	4676.62
Mild	24.77	9.78	17.9	25.46	-0.22	15.84	1.62	-3.85	-3.78	1.83	0.1	1.6	1.16	5E-04	3E-04	1E-04	2E-05		5.E+03	8703.19
Mild	22.13	8.65	15.92	23.18	-0.23	15.11	1.75	-12.53	-12.44	1.84	0.33	1.73	1.25	4E-04	3E-04	2E-04	1E-05		4.E+03	14050.28
Mild	25.21	9.11	17.75	26.74	-0.23	19.01	2.09	-10.06	-10.02	1.95	0.19	2.07	1.91	5E-04	4E-04	3E-04	3E-05	2E-05	1.E+04	3879.38
Mild	26.49	11.07	19.28	20.82	-0.22	17.4	1.57	-13.28	-13.22	1.74	-0.64	1.56	1.12	2E-04	2E-04	2E-04	3E-05	2E-05	9.E+03	4603.16
Mild	21.1	9.39	15.84	22.37	-0.21	13.57	1.45	-5.13	-5.11	1.69	0.89	1.43	0.81	4E-04	2E-04	3E-05	8E-06	8E-06	4.E+03	9761.09
Mild	26.03	9.61	18.29	24.74	-0.21	17.85	1.86	-10.93	-10.91	1.9	0.47	1.85	1.46	6E-04	3E-04	1E-04	2E-05	2E-05	4.E+03	8671.56
Mild	36.16	12.74	24.93	31.5	-0.21	27.5	2.16	-12.24	-12.06	1.96	1.24	2.15	2.16	8E-04	3E-04	3E-04	1E-04	1E-05	1.E+04	9298
Mild	19.52	8.77	14.72	19.67	-0.22	11.79	1.34	-10.45	-10.44	1.68	0.76	1.33	0.74	4E-04	2E-04	3E-05	1E-05	7E-06	4.E+03	9021.95
Mild	23.88	9.63	17.59	30.59	-0.22	15.24	1.58	-10.3	-10.27	1.83	0.75	1.57	0.95	4E-04	2E-04	6E-05	1E-05		4.E+03	9562.65
Mild	30.78	11.34	21.49	28.31	-0.24	22.02	1.94	-12.25	-12.17	1.89	0.71	1.93	1.7	6E-04	4E-04	2E-04	2E-05	2E-05	1.E+04	4059.22
Mild	22.79	8.52	15.95	21.87	-0.23	17.17	2.02	-11.76	-11.76	1.87	0.48	2	1.87	8E-04	5E-04	1E-04	2E-05	2E-05	1.E+04	9054.48
Mild	23.24	9.55	17.03	26.79	-0.22	14.76	1.54	-11.21	-11.17	1.78	0.66	1.53	0.92	4E-04	3E-04	4E-05	1E-05	9E-06	4.E+03	9543.52

Severity	CF	Crest Factor	Impulse Factor	Kurtosis	Mean	Peak Value	RMS	SINAD	S/N	Shape Factor	Skewness	Std	Band Power	Peak Amp1	Peak Amp2	Peak Amp3	Peak Amp4	Peak Amp5	Peak Freq1	Peak Freq3
Mild	27.79	10.75	20.11	30.49	-0.23	18.13	1.69	-11.96	-11.94	1.87	1.03	1.67	1.18	4E-04	3E-04	3E-04	3E-05	2E-05	1.E+04	4207.65
Mild	28.93	10.44	20.11	27.7	-0.22	21.27	2.04	-11.56	-11.51	1.93	-0.05	2.03	1.94	3E-04	3E-04	3E-04	5E-05	1E-05	4.E+03	9925.67
Mild	24.53	10.36	18.01	22.97	-0.24	16.93	1.63	-10.22	-10.22	1.74	0.84	1.62	1.11	3E-04	3E-04	9E-05	2E-05	2E-05	4.E+03	8659.79
Mild	25.77	10.21	18.71	30.41	-0.24	17.4	1.7	-10.19	-10.1	1.83	0.82	1.69	1.16	6E-04	3E-04	1E-04	1E-05	1E-05	4.E+03	9417.1
Mild	29.89	10.53	20.88	31.73	-0.23	19.38	1.84	-9.05	-9	1.98	1.24	1.83	1.56	3E-04	3E-04	3E-04	1E-05	1E-05	1.E+04	4276.45
Mild	23.55	8.58	16.32	21.36	-0.23	16.62	1.94	-11.25	-11.25	1.9	0.28	1.92	1.69	4E-04	4E-04	2E-04	1E-04	1E-05	1.E+04	9180.71
Mild	24.85	9.56	17.89	27.82	-0.23	17.04	1.78	-10.4	-10.36	1.87	0.8	1.77	1.29	6E-04	2E-04	9E-05	2E-05		4.E+03	9675.36
Mild	27.15	9.62	19.19	32.83	-0.21	17.83	1.85	-12.82	-12.76	1.99	0.35	1.84	1.51	3E-04	3E-04	3E-04	5E-05	1E-05	1.E+04	9085.96
Mild	27.79	11.29	20.21	27.15	-0.23	18.18	1.61	-11.75	-11.75	1.79	0	1.59	1.19	5E-04	1E-04	6E-05	2E-05	1E-05	1.E+04	9110.88
Mild	24.49	9.81	17.65	23.45	-0.24	18.11	1.85	-12.65	-12.64	1.8	0.93	1.83	1.32	7E-04	3E-04	6E-05	2E-05		4.E+03	9429.12
Mild	29.01	11.49	21.17	33.4	-0.23	19.62	1.71	-6.89	-6.89	1.84	1.5	1.69	1.19	4E-04	3E-04	3E-04	2E-05	1E-05	1.E+04	9453.65
Mild	37.95	14.88	27.6	45.79	-0.22	24.4	1.64	-10.89	-10.84	1.86	2.16	1.63	1.21	3E-04	2E-04	2E-04	1E-05	6E-06	9.E+03	13476.58
Mild	26.95	9.81	18.74	23.22	-0.23	20.66	2.11	-9.21	-9.21	1.91	0.12	2.09	2.02	5E-04	4E-04	3E-04	4E-05	1E-05	1.E+04	9731.73
Mild	27.48	11.5	20.25	26.42	-0.23	20.91	1.82	-14.34	-14.1	1.76	0.82	1.8	1.29	6E-04	3E-04	1E-04	2E-05	2E-05	4.E+03	9363.83
Mild	36.17	13.34	25.95	45.8	-0.22	22.71	1.7	-12.12	-12.12	1.95	0.88	1.69	1.29	3E-04	3E-04	2E-04	2E-05	9E-06	9.E+03	13789.04
Mild	31.65	11.87	22.43	28.38	-0.2	20.26	1.71	-9.42	-9.42	1.89	0.46	1.69	1.36	3E-04	3E-04	1E-04	1E-04	9E-06	9.E+03	18776.66
Mild	22.84	8.67	15.99	18.75	-0.22	16.86	1.94	-10.3	-10.26	1.84	0.58	1.93	1.54	8E-04	2E-04	8E-05	2E-05		4.E+03	9417.52
Mild	27.52	10.03	19.52	32.67	-0.23	18.89	1.88	-14.66	-14.57	1.95	0.34	1.87	1.53	3E-04	3E-04	3E-04	4E-05	2E-05	9.E+03	4076.22

Severity	CF	Crest Factor	Impulse Factor	Kurtosis	Mean	Peak Value	RMS	SINAD	S/N	Shape Factor	Skewness	Std	Band Power	Peak Amp1	Peak Amp2	Peak Amp3	Peak Amp4	Peak Amp5	Peak Freq1	Peak Freq3
Mild	25.92	11	19.12	26.24	-0.21	16.95	1.54	-15.82	-15.82	1.74	0.19	1.53	1.08	2E-04	2E-04	8E-05	2E-05	1E-05	1.E+04	9087.04
Mild	22.37	9.82	16.62	18.67	-0.23	14.31	1.46	-11.22	-11.22	1.69	0.33	1.44	0.95	3E-04	2E-04	5E-05	5E-05	1E-05	1.E+04	11383.15
Mild	26.36	9.53	18.25	23.8	-0.22	20.66	2.17	-10.2	-10.19	1.92	1.17	2.16	1.89	9E-04	3E-04	2E-04	3E-05		4.E+03	9255.58
Mild	39.69	14.76	28.56	47.9	-0.22	26.94	1.83	-12.16	-12.16	1.94	1.98	1.81	1.55	4E-04	3E-04	2E-04	1E-05	9E-06	9.E+03	4080.93
Mild	22.64	9.16	16.37	20.34	-0.22	14.88	1.62	-11.88	-11.88	1.79	-0.11	1.61	1.23	5E-04	1E-04	1E-04	1E-04	2E-05	1.E+04	18973.78
Mild	23.79	10.91	18.1	23.77	-0.22	15.57	1.43	-13.13	-13.06	1.66	0.92	1.41	0.83	3E-04	2E-04	6E-05	1E-05	1E-05	4.E+03	9376.53
Mild	23.12	8.23	16	23.57	-0.21	17.64	2.14	-12.92	-12.88	1.94	0.78	2.13	1.89	9E-04	3E-04	2E-04	3E-05	2E-05	4.E+03	9403.18
Mild	24.01	9.22	17.13	24.33	-0.2	19.83	2.15	-10.18	-10.11	1.86	1.3	2.14	1.27	7E-04	9E-05	6E-05	9E-06	6E-06	4.E+03	13985.58
Mild	25.53	10.33	18.82	30.6	-0.22	17.37	1.68	-4.64	-4.58	1.82	0.09	1.67	1.08	4E-04	2E-04	4E-05	2E-05		5.E+03	13615.64
Mild	22.95	9.18	16.58	22.72	-0.22	15.67	1.71	-13.21	-13.2	1.81	0.3	1.69	1.34	4E-04	2E-04	1E-04	9E-05		4.E+03	9683.75
Mild	24.91	9.2	17.57	26.04	-0.19	18.64	2.03	-4.65	-4.62	1.91	1.46	2.02	1.39	8E-04	1E-04	9E-05	1E-05		4.E+03	9677.05
Mild	23.21	10.3	17.49	23.04	-0.26	18.61	1.81	-8.79	-8.69	1.7	0.47	1.79	0.87	4E-04	6E-05	3E-05	2E-05	1E-05	4.E+03	13046.08
Mild	25.93	10.62	19.07	28.15	-0.22	16.95	1.6	-10.88	-10.83	1.79	0.66	1.58	1.12	4E-04	2E-04	7E-05	3E-05		4.E+03	10034.1
Mild	35.73	15.13	26.62	38.45	-0.24	22.93	1.52	-8.8	-8.77	1.76	-1.16	1.5	0.95	3E-04	1E-04	5E-05	2E-05		4.E+03	9013.55
Mild	22.71	8.4	15.99	23.87	-0.22	18.11	2.16	-8.38	-8.33	1.9	1.24	2.15	1.31	7E-04	1E-04	4E-05	3E-05	2E-05	4.E+03	13777.56
Mild	28.36	12.67	21.39	28.23	-0.22	19.53	1.54	-12.11	-11.99	1.69	1.32	1.53	0.94	3E-04	9E-05	5E-05	4E-05		5.E+03	9731.03
Mild	40.58	16	29.8	50.12	-0.21	26.59	1.66	-11.07	-10.96	1.86	-1.12	1.65	1.25	4E-04	2E-04	1E-04	5E-05		4.E+03	10114.97

Severity	CF	Crest Factor	Impulse Factor	Kurtosis	Mean	Peak Value	RMS	SINAD	S/N	Shape Factor	Skewness	Std	Band Power	Peak Amp1	Peak Amp2	Peak Amp3	Peak Amp4	Peak Amp5	Peak Freq1	Peak Freq3
Mild	28.25	11.22	20.69	34.61	-0.21	19.75	1.76	-11.09	-11.07	1.84	1.42	1.75	1.09	4E-04	1E-04	8E-05	1E-05	6E-06	4.E+03	14060.37
Mild	23.49	8.45	16.22	22.11	-0.22	18.88	2.23	-8.35	-8.33	1.92	0.78	2.22	1.63	8E-04	2E-04	7E-05	1E-05		4.E+03	13836.48
Mild	29.18	12.3	21.68	28.55	-0.22	20.06	1.63	-12.02	-12.01	1.76	-0.66	1.62	1.2	4E-04	2E-04	1E-04	2E-05		4.E+03	10072.31
Mild	23.58	10.06	17.57	26.79	-0.24	15.6	1.55	-6.98	-6.98	1.75	0.96	1.53	0.85	3E-04	7E-05	6E-05	1E-05		4.E+03	13974.33
Mild	27.18	10.72	19.66	29.77	-0.21	19.75	1.84	-8.02	-7.94	1.83	1.42	1.83	1	4E-04	8E-05	4E-05	2E-05	7E-06	4.E+03	13745.25
Mild	28.11	10.42	19.71	26.04	-0.21	22.97	2.2	-8.29	-8.24	1.89	0.75	2.19	1.89	4E-04	3E-04	2E-04	1E-05	8E-06	4.E+03	9488.63
Mild	36.51	15.22	27.16	40.13	-0.21	24.29	1.6	-11.14	-11.11	1.78	-0.86	1.58	1.08	3E-04	2E-04	6E-05	1E-05		4.E+03	9190.11
Mild	29.21	11.26	21.22	36.86	-0.23	19.07	1.69	-7.8	-7.77	1.88	1.62	1.68	0.83	5E-04	6E-05	2E-05	7E-06	7E-06	4.E+03	14143.46
Mild	26.02	10.4	19.02	29.78	-0.22	18.61	1.79	-11.2	-11.19	1.83	1.42	1.78	1.1	4E-04	7E-05	5E-05	1E-05		4.E+03	9806.64
Mild	33.69	12.32	23.63	32.81	-0.21	25.61	2.08	-9.97	-9.96	1.92	0.24	2.07	1.92	5E-04	3E-04	2E-04	4E-05		4.E+03	10141.35
Mild	27.77	10.46	20.04	36	-0.21	18.36	1.75	-7.82	-7.66	1.92	1.5	1.74	1.09	5E-04	1E-04	1E-04	1E-05		4.E+03	14030.82
Mild	26.83	10.35	19.49	33.14	-0.2	17.74	1.71	-6.19	-6.09	1.88	0.76	1.7	0.78	4E-04	3E-05	3E-05	1E-05	1E-05	4.E+03	13781.5
Mild	25.91	9.69	18.24	24.15	-0.22	19.28	1.99	-11.53	-11.33	1.88	0.43	1.98	1.68	7E-04	2E-04	2E-04	2E-05		4.E+03	14242.86
Mild	26.52	10.52	19.04	22.74	-0.22	18.63	1.77	-10.17	-10.14	1.81	1.14	1.76	1.19	4E-04	1E-04	9E-05	3E-05		4.E+03	14053
Mild	27.34	10.57	19.9	36.15	-0.23	18.34	1.73	-1.33	-1.09	1.88	1.24	1.72	0.8	5E-04	4E-05	3E-05	1E-05	6E-06	4.E+03	10255.49
Mild	23.8	9.83	17.53	27.23	-0.22	16.89	1.72	-4.91	-4.83	1.78	1.14	1.7	1.01	5E-04	6E-05	5E-05	3E-05	1E-05	4.E+03	9942.34
Mild	22.9	8.78	16.19	21.08	-0.21	18.29	2.08	-10.85	-10.78	1.84	0.09	2.07	1.85	4E-04	2E-04	2E-04	7E-05		4.E+03	14232.88
Mild	25.99	10.56	19.06	31.83	-0.24	18.26	1.73	-14.5	-14.4	1.8	1.47	1.71	0.88	4E-04	6E-05	5E-05	9E-06	7E-06	4.E+03	14050.82

Severity	CF	Crest Factor	Impulse Factor	Kurtosis	Mean	Peak Value	RMS	SINAD	S/N	Shape Factor	Skewness	Std	Band Power	Peak Amp1	Peak Amp2	Peak Amp3	Peak Amp4	Peak Amp5	Peak Freq1	Peak Freq3
Mild	27.44	11.12	20.3	35.63	-0.23	18.4	1.66	-2.93	-2.33	1.83	1.47	1.64	0.8	4E-04	6E-05	5E-05	3E-05	2E-05	4.E+03	9575.19
Mild	32.95	12.06	23.2	30.6	-0.22	24.79	2.06	-9.7	-9.68	1.92	0.59	2.04	1.9	6E-04	4E-04	8E-05	5E-05		4.E+03	10244.51
Mild	25.18	10.25	18.59	34.46	-0.21	17.97	1.75	-7.12	-6.99	1.81	1.53	1.74	1.05	4E-04	9E-05	7E-05	9E-06	9E-06	4.E+03	13971.07
Mild	26.48	10.28	19.45	37.69	-0.21	17.28	1.68	-13.93	-13.9	1.89	1.25	1.67	0.76	4E-04	5E-05	4E-05	1E-05	9E-06	4.E+03	9880.1
Mild	24.67	10.3	18.25	25.81	-0.24	16.43	1.6	-8.94	-8.93	1.77	0.87	1.58	1	4E-04	1E-04	4E-05	3E-05		4.E+03	9752.1
Mild	27.12	9.85	18.86	25.04	-0.22	20.96	2.13	-7.67	-7.59	1.91	0.54	2.12	1.91	8E-04	3E-04	2E-04	3E-05		4.E+03	9630.74
Mild	23.63	9.59	17.39	29.02	-0.23	18	1.88	-9.14	-8.9	1.81	0.9	1.86	0.95	5E-04	5E-05	4E-05	1E-05	7E-06	4.E+03	14126.93
Mild	26.16	10.14	18.96	29.35	-0.22	17.96	1.77	-10.77	-10.73	1.87	0.83	1.76	1.18	5E-04	1E-04	7E-05	4E-05		4.E+03	13786.4
Mild	27.8	9.88	19.31	27.99	-0.23	20.32	2.06	-13.58	-13.55	1.95	0.73	2.04	1.99	5E-04	5E-04	2E-04	3E-05		4.E+03	9968.38
Mild	22.87	8.51	16.1	23.45	-0.23	17.65	2.07	-8.43	-8.41	1.89	1.07	2.06	1.42	7E-04	1E-04	1E-04	1E-05		4.E+03	9574.28
Mild	26.35	11.12	19.71	31.38	-0.2	18.72	1.68	-10.94	-10.81	1.77	1.36	1.67	0.83	4E-04	6E-05	3E-05	2E-05	1E-05	4.E+03	13796.14
Mild	33.67	13.17	24.38	35.44	-0.24	22.71	1.72	-13.34	-13.3	1.85	0.92	1.71	1.29	3E-04	2E-04	8E-05	2E-05		4.E+03	9659.96
Mild	33.81	13.76	24.8	36.03	-0.23	22.32	1.62	-10.47	-10.43	1.8	-0.53	1.61	1.17	3E-04	2E-04	8E-05	2E-05		4.E+03	9407.36
Mild	27.8	9.96	19.39	26.67	-0.22	22.2	2.23	-9.4	-9.37	1.95	1.44	2.22	1.41	9E-04	1E-04	4E-05	1E-05	1E-05	4.E+03	13837.04
Mild	29.66	12.73	22.19	32.49	-0.21	20.1	1.58	-7.63	-7.62	1.74	1.68	1.57	0.98	3E-04	1E-04	5E-05	2E-05		4.E+03	10315.4
Mild	30.95	12.16	22.55	37.47	-0.23	20.7	1.7	-11.72	-11.59	1.85	-0.35	1.69	1.32	5E-04	3E-04	1E-04	2E-05		4.E+03	9797.73
Mild	31.89	12.58	23.39	41.88	-0.21	20.96	1.67	-11.06	-10.93	1.86	1.48	1.65	1.04	4E-04	1E-04	7E-05	1E-05	7E-06	4.E+03	14103.82

Severity	CF	Crest Factor	Impulse Factor	Kurtosis	Mean	Peak Value	RMS	SINAD	S/N	Shape Factor	Skewness	Std	Band Power	Peak Amp1	Peak Amp2	Peak Amp3	Peak Amp4	Peak Amp5	Peak Freq1	Peak Freq3
Mild	21.45	7.96	15.09	23.27	-0.23	17.97	2.26	-9.34	-9.28	1.9	1.2	2.25	1.44	8E-04	7E-05	3E-05	2E-05	1E-05	4.E+03	13657.55
Mild	27.36	10.82	19.89	29.04	-0.24	18.47	1.71	-11.77	-11.75	1.84	0.05	1.69	1.31	3E-04	2E-04	1E-04	2E-05		4.E+03	9714.83
Mild	29.37	12.29	21.93	34.7	-0.22	18.79	1.53	-10.3	-10.15	1.78	0.78	1.51	0.93	4E-04	1E-04	7E-05	1E-05	7E-06	4.E+03	9667.49
Mild	25.31	9.88	18.37	31.02	-0.23	18.2	1.84	-11.58	-11.5	1.86	1.2	1.83	1.01	5E-04	1E-04	5E-05	1E-05	1E-05	4.E+03	13217.09
Mild	28.06	10.29	19.62	23.7	-0.2	22.3	2.17	-6.58	-6.57	1.91	0.86	2.16	1.72	6E-04	2E-04	1E-04	1E-05		4.E+03	9702.59
Mild	28.72	12.33	21.33	26.93	-0.22	18.98	1.54	-9.81	-9.76	1.73	-0.26	1.52	1.03	4E-04	2E-04	5E-05	2E-05		4.E+03	9920.3
Mild	26.75	10.4	19.54	35.48	-0.25	17.68	1.7	-6.05	-5.95	1.88	1.48	1.68	0.88	4E-04	7E-05	6E-05	7E-06	6E-06	4.E+03	14390.74
Mild	29.45	11.05	21.07	28.79	-0.19	22.98	2.08	-4.35	-4.32	1.91	1.71	2.07	1.36	5E-04	2E-04	8E-05	1E-05		4.E+03	9844.67
Mild	32.72	13.65	24.18	32.68	-0.26	23.66	1.73	-6.34	-6.33	1.77	-1.01	1.71	1.27	5E-04	2E-04	8E-05	5E-05		4.E+03	10055.4
Mild	30.35	11.88	22.28	44.91	-0.25	19.29	1.62	-7.37	-7.36	1.88	1.7	1.6	0.91	4E-04	7E-05	6E-05	8E-06	6E-06	4.E+03	13952.63
Mild	24.92	9.42	17.9	28.34	-0.22	18.48	1.96	-7.66	-7.58	1.9	1	1.95	1.18	5E-04	2E-04	5E-05	2E-05	2E-05	4.E+03	13502.67
Mild	26.66	10	18.83	23.18	-0.23	21.24	2.12	-11.61	-11.56	1.88	0.85	2.11	1.88	7E-04	2E-04	2E-04	4E-05		4.E+03	9893.51
Mild	25.63	10.59	18.86	26.58	-0.21	17.32	1.64	-10.86	-10.69	1.78	0.68	1.62	1.04	4E-04	2E-04	6E-05	1E-05		4.E+03	9195.72
Mild	26.44	11.14	19.73	34.83	-0.21	17.61	1.58	-6.94	-6.93	1.77	1.7	1.57	0.85	4E-04	3E-05	3E-05	2E-05	1E-05	4.E+03	7694.86
Mild	27.65	11.93	20.71	27.08	-0.22	17.49	1.47	-13.22	-13.22	1.74	-0.67	1.45	0.89	4E-04	9E-05	6E-05	1E-05		4.E+03	14324.11
Mild	22.87	9.83	17.08	26.68	-0.26	14.74	1.5	-10.1	-10.08	1.74	-0.64	1.48	0.74	2E-04	8E-05	6E-05	8E-06	6E-06	4.E+03	14276.16
Mild	25.9	9.33	18.3	30.51	-0.19	20.73	2.22	-5.42	-5.39	1.96	2.05	2.22	1.15	8E-04	4E-05	3E-05	2E-05	8E-06	4.E+03	10370.56
Mild	21.16	9.13	15.74	20.84	-0.22	14.16	1.55	-10.19	-10.19	1.72	0.02	1.53	0.97	2E-04	1E-04	5E-05	5E-05	3E-05	4.E+03	13806.27

Severity	CF	Crest Factor	Impulse Factor	Kurtosis	Mean	Peak Value	RMS	SINAD	S/N	Shape Factor	Skewness	Std	Band Power	Peak Amp1	Peak Amp2	Peak Amp3	Peak Amp4	Peak Amp5	Peak Freq1	Peak Freq3
Mild	25.05	11.24	18.94	26.38	-0.21	16.25	1.45	-14.69	-14.62	1.69	0.82	1.43	0.83	2E-04	6E-05	5E-05	4E-05		4.E+03	14351.87
Mild	28.18	9.67	19.47	29.27	-0.24	21.59	2.23	-11.53	-11.53	2.01	1.81	2.22	1.35	0.001	6E-05	5E-05	2E-05	1E-05	4.E+03	13972.65
Mild	20.1	9.28	15.2	19.07	-0.19	14.22	1.53	-10.47	-10.11	1.64	0.72	1.52	0.83	2E-04	6E-05	3E-05	1E-05	1E-05	4.E+03	10852.39
Mild	26.99	11.82	20.14	24.12	-0.21	17.05	1.44	-8.77	-8.72	1.7	-0.47	1.43	0.87	3E-04	7E-05	4E-05	1E-05		4.E+03	10554.66
Mild	25.53	10.75	18.9	28.56	-0.22	16.46	1.53	-11.16	-10.89	1.76	0.62	1.51	0.69	2E-04	5E-05	5E-05	4E-05	1E-05	4.E+03	14293.73
Mild	25.8	9.23	18.01	28.9	-0.24	20.44	2.22	-12.84	-12.83	1.95	1.73	2.2	1.44	0.001	7E-05	5E-05	2E-05	1E-05	4.E+03	13628.64
Mild	22.18	9.44	16.43	21.91	-0.22	15.2	1.61	-8.73	-8.72	1.74	-0.2	1.6	1.11	7E-04	7E-05	6E-05	2E-05		4.E+03	10777.2
Mild	25.53	11.71	19.49	26.53	-0.21	16.33	1.39	-7.49	-7.47	1.66	0.09	1.38	0.61	2E-04	7E-05	6E-05	3E-05	1E-05	4.E+03	10556.12
Mild	30.74	11.44	22.07	38.7	-0.21	21.22	1.86	-3.87	-3.78	1.93	1.83	1.84	0.91	6E-04	3E-05	2E-05	2E-05	9E-06	4.E+03	13813.08
Mild	27.33	10.38	19.26	25.14	-0.22	20.15	1.94	-14.38	-14.37	1.86	1.39	1.93	1.33	9E-04	6E-05	4E-05	2E-05		4.E+03	10169.82
Mild	26.35	11.69	19.98	31.77	-0.24	18.17	1.55	-11.13	-11.11	1.71	0.83	1.54	0.94	2E-04	1E-04	1E-04	9E-06	8E-06	4.E+03	14376.16
Mild	33.07	12.49	24	41.94	-0.22	21.43	1.72	-8.43	-8.33	1.92	1.85	1.7	0.78	6E-04	3E-05	2E-05	9E-06	6E-06	4.E+03	14220.81
Mild	30.1	11.61	21.86	38.2	-0.21	20.92	1.8	-13.54	-13.48	1.88	1.98	1.79	1.05	5E-04	5E-05	3E-05	3E-05	1E-05	4.E+03	10918.77
Mild	20.19	8.48	14.69	18.18	-0.23	15.81	1.86	-8.41	-8.39	1.73	0.34	1.85	1.39	4E-04	2E-04	1E-04	3E-05		4.E+03	13849.06
Mild	29.95	12.15	22.14	33.9	-0.22	19.07	1.57	-5.97	-5.7	1.82	0.95	1.56	0.69	3E-04	5E-05	5E-05	3E-05	1E-05	4.E+03	14220.85
Mild	31.75	11.6	22.9	44.77	-0.22	20.71	1.79	-10.04	-9.85	1.98	2.26	1.77	0.92	7E-04	3E-05	3E-05	1E-05	7E-06	4.E+03	10149.67
Mild	27.42	11.26	20.09	25.86	-0.2	20.25	1.8	-16.13	-16.13	1.78	0.11	1.79	1.26	6E-04	1E-04	1E-04	4E-05	1E-05	4.E+03	10755.87

Severity	CF	Crest Factor	Impulse Factor	Kurtosis	Mean	Peak Value	RMS	SINAD	S/N	Shape Factor	Skewness	Std	Band Power	Peak Amp1	Peak Amp2	Peak Amp3	Peak Amp4	Peak Amp5	Peak Freq1	Peak Freq3
Mild	22.14	9.66	16.4	20.66	-0.22	15.49	1.6	-10.25	-10.25	1.7	0.17	1.59	0.82	3E-04	5E-05	5E-05	4E-05	1E-05	4.E+03	14239.18
Mild	31.26	11.12	22.28	43.18	-0.22	20.11	1.81	-3.35	-3.15	2	1.98	1.8	0.88	6E-04	5E-05	3E-05	1E-05		4.E+03	10187.44
Mild	26.42	10.69	19.42	36.13	-0.22	16.97	1.59	-8.08	-8	1.82	1.88	1.57	0.83	4E-04	3E-05	3E-05	2E-05	1E-05	4.E+03	7591.86
Mild	22.74	9.16	16.46	21.56	-0.21	17.22	1.88	-13.46	-13.4	1.8	0.54	1.87	1.39	6E-04	2E-04	1E-04	5E-05		4.E+03	10314.21
Mild	30.46	11.63	22.01	38.55	-0.21	20.94	1.8	-13.09	-13.07	1.89	1.89	1.79	0.91	5E-04	5E-05	3E-05	3E-05	1E-05	4.E+03	14126.95
Mild	32.35	12.23	23.66	50.71	-0.21	20.19	1.65	-1.07	-1.07	1.93	2.83	1.64	0.84	5E-04	3E-05	3E-05	1E-05	8E-06	4.E+03	10387.05
Mild	17.97	7.62	13.14	16.54	-0.22	13.18	1.73	-12.15	-12.14	1.73	-0.19	1.72	1.23	3E-04	1E-04	9E-05	7E-05		4.E+03	11004.11
Mild	25.31	10.11	18.36	26.47	-0.21	19.66	1.94	-11.67	-11.67	1.82	1.46	1.93	1.1	5E-04	5E-05	5E-05	4E-05		4.E+03	14125.27
Mild	29.63	11.13	21.46	40.31	-0.22	20.14	1.81	-13.73	-13.7	1.93	2.09	1.8	0.95	6E-04	4E-05	4E-05	8E-06	7E-06	4.E+03	10366.47
Mild	22.72	10.2	17.17	25.31	-0.22	14.5	1.42	-3.8	-3.34	1.68	1.35	1.4	0.67	2E-04	5E-05	3E-05	2E-05	2E-05	4.E+03	11086.8
Mild	22.76	8.84	16.21	22.71	-0.23	16.2	1.83	-8.69	-8.6	1.83	0.01	1.82	1.17	5E-04	1E-04	5E-05	2E-05		4.E+03	10266.61
Mild	27.43	10.92	20.03	34.74	-0.22	21.26	1.95	-7.87	-7.83	1.83	2.04	1.93	1.07	5E-04	6E-05	4E-05	4E-05	1E-05	4.E+03	14011.75
Mild	26.09	11.17	19.54	29.33	-0.21	17.26	1.55	-9.33	-9.33	1.75	1.4	1.53	0.76	3E-04	4E-05	3E-05	2E-05	1E-05	4.E+03	7163.69
Mild	29.06	12.13	21.58	32.1	-0.23	18.94	1.56	-10.52	-10.52	1.78	-0.73	1.54	1.07	5E-04	1E-04	7E-05	2E-05		4.E+03	14329.17
Mild	26.84	9.67	18.77	29.79	-0.23	20.17	2.09	-8.97	-8.93	1.94	1.53	2.07	1.21	8E-04	9E-05	3E-05	1E-05	1E-05	4.E+03	14229.56
Mild	27.07	11.49	20.36	35.63	-0.23	19.48	1.7	-8.08	-8.07	1.77	1.67	1.68	0.89	5E-04	4E-05	3E-05	3E-05	1E-05	4.E+03	10455.12
Mild	22.39	9.77	16.77	23.39	-0.22	14.4	1.47	-10.36	-10.34	1.72	-0.22	1.46	0.88	3E-04	1E-04	8E-05	3E-05	2E-05	4.E+03	11044.96

Severity	CF	Crest Factor	Impulse Factor	Kurtosis	Mean	Peak Value	RMS	SINAD	S/N	Shape Factor	Skewness	Std	Band Power	Peak Amp1	Peak Amp2	Peak Amp3	Peak Amp4	Peak Amp5	Peak Freq1	Peak Freq3
Mild	27.31	11.87	20.66	32.28	-0.23	17.39	1.47	-12.47	-12.33	1.74	1.05	1.45	0.8	2E-04	9E-05	6E-05	9E-06	7E-06	4.E+03	14193.32
Mild	26.54	9.3	18.43	29.35	-0.21	21.44	2.31	-8.34	-8.15	1.98	2	2.3	1.44	0.001	5E-05	3E-05	1E-05	1E-05	4.E+03	13864.18
Mild	20.82	10.05	16.06	21.2	-0.23	14.01	1.39	-11.25	-11.24	1.6	1.14	1.38	0.65	2E-04	5E-05	3E-05	1E-05	9E-06	4.E+03	11591.56
Mild	22.6	9.89	16.89	20.82	-0.21	14.47	1.46	-13.93	-13.91	1.71	-0.14	1.45	0.88	2E-04	1E-04	8E-05	2E-05		4.E+03	13778.8
Mild	26.71	11.11	19.8	31.52	-0.23	17.77	1.6	-8.84	-8.77	1.78	0.94	1.58	0.74	3E-04	6E-05	5E-05	3E-05	8E-06	4.E+03	14439.77
Mild	25.93	9.54	18.26	26.32	-0.22	21.32	2.23	-11.19	-11.13	1.91	1.65	2.22	1.34	8E-04	6E-05	4E-05	2E-05	1E-05	4.E+03	10191.3
Mild	31.44	13.65	23.4	32.68	-0.22	20.63	1.51	-8.79	-8.77	1.71	-0.49	1.5	0.98	5E-04	1E-04	7E-05	2E-05		4.E+03	10957.99
Mild	25.2	11.11	18.94	28.42	-0.22	16.2	1.46	-9.22	-9.08	1.7	0.69	1.44	0.66	2E-04	8E-05	8E-05	8E-06	6E-06	4.E+03	14376.71
Mild	32.34	12.05	23.29	41.86	-0.21	22.76	1.89	-9.21	-9.16	1.93	2.13	1.88	1.02	6E-04	3E-05	3E-05	2E-05	1E-05	4.E+03	7679.91
Mild	22.02	8.73	15.84	21.59	-0.2	16.87	1.93	-8.95	-8.85	1.81	0.87	1.92	1.26	8E-04	7E-05	5E-05	2E-05		4.E+03	10507.6
Mild	23.74	10.46	18.01	29.89	-0.23	14.85	1.42	-13.33	-13.33	1.72	0.26	1.4	0.73	2E-04	9E-05	8E-05	8E-06	7E-06	4.E+03	14488.55
Mild	32.92	11.99	23.68	45.95	-0.21	20.48	1.71	-8.34	-8.31	1.97	2.38	1.7	0.77	6E-04	3E-05	2E-05	8E-06	5E-06	4.E+03	14060.55
Mild	29.91	11.44	21.59	33.99	-0.21	21.57	1.88	-8.92	-8.91	1.89	1.57	1.87	1.08	7E-04	4E-05	4E-05	3E-05	2E-05	4.E+03	10578.78
Mild	21.27	9.17	15.72	19.45	-0.21	15.62	1.7	-7.72	-7.67	1.71	0.32	1.69	1.1	4E-04	6E-05	4E-05	2E-05		4.E+03	14030.92
Mild	32.56	12.59	23.77	42.25	-0.21	21.03	1.67	-7.01	-6.72	1.89	1.95	1.66	0.83	5E-04	7E-05	3E-05	3E-05	1E-05	4.E+03	14152.76
Mild	32.94	11.89	23.44	39.66	-0.22	23.06	1.94	-8.77	-8.66	1.97	2.15	1.93	1.06	7E-04	5E-05	3E-05	1E-05	7E-06	4.E+03	13854.27
Mild	23.33	9.66	16.95	21.12	-0.21	17.59	1.82	-14.93	-14.89	1.75	0.2	1.81	1.26	3E-04	1E-04	1E-04	1E-05		4.E+03	10191.87
Mild	27.4	11.54	20.22	26.92	-0.22	19.18	1.66	-11.63	-11.63	1.75	0.85	1.65	0.85	4E-04	7E-05	5E-05	4E-05	1E-05	4.E+03	14482.37

Severity	CF	Crest Factor	Impulse Factor	Kurtosis	Mean	Peak Value	RMS	SINAD	S/N	Shape Factor	Skewness	Std	Band Power	Peak Amp1	Peak Amp2	Peak Amp3	Peak Amp4	Peak Amp5	Peak Freq1	Peak Freq3
Mild	34.13	12	24.33	44.91	-0.21	23	1.92	-2.19	-2.01	2.03	2.33	1.9	0.99	8E-04	4E-05	3E-05	1E-05	7E-06	4.E+03	9875.45
Moderate	26.27	10.89	19.72	34.67	-0.2	16.86	1.55	-11.84	-11.82	1.81	1.85	1.54	0.73	4E-04	3E-05	3E-05	2E-05		4.E+03	13961.3
Moderate	26.6	11.9	20.11	26.48	-0.22	16.48	1.39	-13.63	-13.57	1.69	-0.24	1.37	0.77	1E-04	9E-05	4E-05	1E-05		4.E+03	14581.48
Moderate	26.57	8.08	17.28	24.42	-0.21	22.71	2.81	-7.2	-7.18	2.14	0.81	2.8	2.14	5E-04	4E-04	2E-04	8E-05	4E-05	4.E+03	6955.8
Moderate	30.3	12.14	22.2	32.4	-0.23	22.88	1.88	-10.78	-10.72	1.83	1.35	1.87	1.06	4E-04	5E-05	3E-05	2E-05		4.E+03	13565.91
Moderate	18.85	8.49	14.21	21.15	-0.22	11.94	1.41	-11.65	-11.64	1.67	0.22	1.39	0.76	3E-04	1E-04	5E-05	3E-05	2E-05	4.E+03	11019.3
Moderate	29.29	11.84	21.52	32.54	-0.22	19.84	1.68	-9.79	-9.62	1.82	1	1.66	0.98	2E-04	1E-04	1E-04	6E-05	2E-05	3.E+03	10552.02
Moderate	25.93	8.15	17.05	24.64	-0.22	23.38	2.87	-10.9	-10.86	2.09	0.7	2.86	2.1	6E-04	2E-04	2E-04	1E-04	4E-05	4.E+03	10383.41
Moderate	22.21	10.3	16.93	23.92	-0.2	14.75	1.43	-10.78	-10.77	1.64	1.38	1.42	0.71	3E-04	4E-05	2E-05	2E-05		4.E+03	10208.15
Moderate	29.86	12.35	21.96	30.63	-0.2	19.64	1.59	-13.44	-13.3	1.78	1.09	1.58	1	2E-04	1E-04	7E-05	4E-05	2E-05	4.E+03	6999.45
Moderate	33.37	10.53	22.79	40.93	-0.23	23.73	2.25	-8.42	-8.4	2.16	0.49	2.24	1.34	2E-04	2E-04	2E-04	6E-05	4E-05	4.E+03	10457.26
Moderate	26.01	8.84	17.7	26.48	-0.21	22	2.49	-13.79	-13.7	2	1.38	2.48	1.67	7E-04	9E-05	9E-05	5E-05	2E-05	4.E+03	6883.98
Moderate	22.65	10.14	17.13	26.29	-0.2	14.78	1.46	-11.15	-11.15	1.69	0.99	1.44	0.85	3E-04	8E-05	4E-05	2E-05		4.E+03	14410.52
Moderate	30.47	10.49	21.62	44.47	-0.22	20.94	2	-2.61	-2.43	2.06	0.87	1.98	1.14	2E-04	2E-04	1E-04	4E-05	2E-05	3.E+03	6812.82
Moderate	29.47	9.69	20.13	33.59	-0.21	21.87	2.26	-12.99	-12.89	2.08	1.17	2.25	1.3	4E-04	9E-05	3E-05	2E-05		4.E+03	13857.25
Moderate	24.87	9.81	17.94	24.34	-0.19	18.51	1.89	-13.24	-13.2	1.83	1.08	1.88	1.18	4E-04	1E-04	1E-04	4E-05	3E-05	4.E+03	11281.35
Moderate	31.75	11.81	23	39.2	-0.19	20.99	1.78	-6.49	-6.43	1.95	0.97	1.77	1	2E-04	1E-04	1E-04	5E-05	2E-05	4.E+03	6856.87
Moderate	39.27	11.96	26.52	43.05	-0.2	27.38	2.29	-2.92	-2.76	2.22	1.16	2.28	1.44	3E-04	2E-04	9E-05	6E-05	5E-05	4.E+03	6584.92

Severity	CF	Crest Factor	Impulse Factor	Kurtosis	Mean	Peak Value	RMS	SINAD	S/N	Shape Factor	Skewness	Std	Band Power	Peak Amp1	Peak Amp2	Peak Amp3	Peak Amp4	Peak Amp5	Peak Freq1	Peak Freq3
Moderate	30.38	11.12	21.48	33.47	-0.19	22.15	1.99	-10.83	-10.79	1.93	1.37	1.98	1.26	5E-04	9E-05	8E-05	6E-05	2E-05	4.E+03	14235.88
Moderate	27.85	11.24	20.16	25.96	-0.22	20.51	1.83	-11.87	-11.85	1.79	1	1.81	1.14	2E-04	1E-04	1E-04	4E-05	3E-05	4.E+03	10851.47
Moderate	32.7	10.04	22.08	40.97	-0.21	23	2.29	-1.08	-0.85	2.2	0.94	2.28	1.35	3E-04	2E-04	1E-04	8E-05	4E-05	4.E+03	6632.07
Moderate	31.91	10.85	22.43	41.08	-0.22	22.1	2.04	-9.3	-9.24	2.07	1.57	2.02	1.14	4E-04	7E-05	6E-05	3E-05	3E-05	4.E+03	6880.93
Moderate	29.7	11.93	21.44	24.9	-0.21	23.23	1.95	-10.24	-10.24	1.8	0.84	1.94	1.45	3E-04	1E-04	6E-05	4E-05		4.E+03	14356.77
Moderate	31.61	10.81	22.11	37.87	-0.22	23.77	2.2	-15.15	-15.15	2.05	0.85	2.19	1.28	3E-04	2E-04	1E-04	6E-05	4E-05	4.E+03	10525.23
Moderate	33.73	10.96	23.29	40.18	-0.22	22.96	2.1	-0.85	-0.59	2.13	1.37	2.08	1.11	4E-04	9E-05	5E-05	4E-05	2E-05	4.E+03	10479.41
Moderate	27.88	11.37	20.43	30.79	-0.19	18.61	1.64	-9.65	-9.52	1.8	1.48	1.63	0.91	3E-04	5E-05	4E-05	2E-05		4.E+03	8956.02
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Healthy	5.89	4.01	5.01	2.98	-0.14	3.54	0.88	-15.16	-15.16	1.25	-0.06	0.87	0.28	3E-05	2E-05				6.E+03	
Healthy	5.48	3.7	4.65	3.08	-0.13	3.36	0.91	-13.8	-13.75	1.26	0.08	0.9	0.28	3E-05	3E-05				1.E+04	
Healthy	5.5	3.76	4.69	2.94	-0.13	3.34	0.89	-14.75	-14.73	1.25	-0.02	0.88	0.27	3E-05	3E-05				5.E+03	
Healthy	5.81	3.91	4.91	3.05	-0.14	3.52	0.9	-13.96	-13.96	1.26	0.04	0.89	0.28	3E-05	3E-05	5E-06			6.E+03	22953
Healthy	5.46	3.69	4.62	2.92	-0.13	3.21	0.87	-14.5	-14.47	1.25	0.04	0.86	0.26	3E-05	2E-05				6.E+03	
Healthy	5.17	3.48	4.37	2.92	-0.15	3.15	0.91	-11.66	-11.65	1.26	-0.02	0.89	0.29	3E-05	3E-05				6.E+03	
Healthy	5.65	3.85	4.8	3	-0.13	3.41	0.89	-14.26	-14.26	1.25	0	0.88	0.28	4E-05	3E-05	5E-06			1.E+04	21997.51
Healthy	6.29	4.24	5.32	3	-0.13	3.86	0.91	-15.8	-15.8	1.26	-0.04	0.9	0.29	3E-05	3E-05				1.E+04	
Healthy	7.03	4.71	5.93	3.11	-0.12	4.19	0.89	-14.23	-14.18	1.26	-0.08	0.88	0.28	3E-05	3E-05				6.E+03	

Severity	CF	Crest Factor	Impulse Factor	Kurtosis	Mean	Peak Value	RMS	SINAD	S/N	Shape Factor	Skewness	Std	Band Power	Peak Amp1	Peak Amp2	Peak Amp3	Peak Amp4	Peak Amp5	Peak Freq1	Peak Freq3
Healthy	5.47	3.72	4.65	2.98	-0.13	3.44	0.93	-14.02	-13.99	1.25	0	0.92	0.3	3E-05	3E-05				5.E+03	
Healthy	5.94	4.02	5.03	2.99	-0.14	3.64	0.9	-15.24	-15.22	1.25	-0.02	0.89	0.29	3E-05	3E-05				5.E+03	
Healthy	5.84	3.96	4.95	2.95	-0.13	3.53	0.89	-10.86	-10.85	1.25	0.02	0.88	0.28	3E-05	2E-05				5.E+03	
Healthy	5.96	4.02	5.03	3	-0.15	3.63	0.9	-13.75	-13.75	1.25	0.02	0.89	0.28	3E-05	3E-05				1.E+04	