A multi-fidelity approach to predict stress fields in elastic structures with random pores through Bayesian deep learning

Vasileios Krokos

Supervisors: Pierre Kerfriden
Viet Bui Xuan
Stéphane P. A. Bordas

A thesis submitted to the graduate school
in fulfilment of the requirements for the degree of
Doctor of Philosophy

Cardiff School of Engineering

June, 2023

Summary

Multiscale structures, although being prevalent in engineering applications, are challenging to analyse due to the high computational cost involved in simulating them using direct numerical simulations. This increased cost is the result of the very fine mesh that needs to be used to capture the effect of the smallest geometrical features. Common approaches to tackle multiscale problems fall either in the homogenisation or domain decomposition techniques. Homogenisation, that is computationally less expensive, assumes that the macroscale displacement gradients vary slowly over the structure. That assumption is commonly violated in real world applications, thus it is necessary to employ domain decomposition techniques, which are computationally more expensive and practically more intrusive.

The objective of this study is to develop a multiscale surrogate modelling technique that can be used to perform fast stress predictions in structures exhibiting spatially random microscopic features. The proposed approach does not assume separation of scales and it does not require prior geometrical parametrisation of the multiscale problem. The input to the developed framework is an inexpensively calculated macroscale solution obtained using a coarse finite element mesh that ignores the microscale features. The framework outputs corrections to the macroscale field that take into account the existence of the microscale features. A Neural Network (NN) is utilised to perform these fine scale corrections. As opposed to classical surrogate models like Gaussian Processes, NNs can efficiently learn and operate on image and graph data. In contrast to the relevant literature, in this thesis we are primarily interested in cases where multiple fine scale features are interacting with each other and/or the boundaries of the structure.

Another area of focus is the exploration of Bayesian NNs (BNNs). As opposed to deterministic NNs, BNNs can be used to provide uncertainty information for their prediction, which is crucial in engineering applications. Nonetheless, most of the works found in the relevant literature are deterministic. In this thesis we use the Bayes By Backprop method to convert the developed NNs to BNNs. Additionally, we show how this uncertainty can be utilised to solve problems reported by NN practitioners. Firstly, using efficient Bayesian conditioning algorithms we impose physics-based corrections to the output of the network, leading to improved performance in cases where training data are sparse. Additionally, we adopt selective learning which is commonly used to reduce the need for labelled data in segmentation tasks. In this work we show how it can be used in the context of regression tasks.

Lastly, we adopt two techniques from the computer vision domain to reduce the training data requirements. To this end, we use mechanically consistent rotations as data augmentation technique and transfer learning to efficiently train models with limited number of data.

The conducted experiments show that the proposed approach manages to accurately predict both the stress distribution and the maximum equivalent stress in the examined structures, as long as these lay inside the training data distribution. Additionally, the uncertainty of the prediction is quantified and shown to positively correlate with the prediction error. Finally, in terms of data requirements we show that with a dataset of reasonable size, below 500 FE simulations, the performance of the networks is satisfactory.

Acknowledgements

TL;DR: This journey with all its moments from the very bad to the very good ones changed me both as a researcher and as a person. I thank all those who were part of it and helped me in any way!

For the rest of you, here is the outline of the section:

```
\begin{enumerate}
    \item Supervisors
    \item RAINBOW
    \item Synopsys
    \item Family, Friends and More
\end{enumerate}


% Supervisors
```

I would like to acknowledge my supervisors Pierre, Viet, Stéphane and my former supervisor Philippe for trusting me and treating me like a researcher whose opinion was not less important than theirs. Pierre, your deep understanding of science in multiple levels, was a real inspiration to me. Your guidance was invaluable, suggesting meaningful directions to explore, helping me get out of the many pitfalls that, so annoyingly, appeared along the way but also making me understand the value and potential of our work. I appreciate your honesty and I believe our interaction really made me a better researcher, able to defend my work but also being more open to, constructive, criticism. Viet, first and foremost thank you for always being there and genuinely caring. You provided an objective view and helped me to keep an eye on the greater picture. Thank

you for all your guidance, advice, and help, that definitely did not stop where your "duties" as my PhD supervisor did. Our interaction helped me to grow as a person, become a useful member of a scientific/professional team that can efficiently communicate with the other members, ask for help, be open, and be able to share his "expertise" with both experts and non-experts in the field.

% RAINBOW

I had the privilege to be a part of the RAINBOW, Marie Sklodowska-Curie European Training Network, founded by European Union's Horizon 2020 research and innovation program (grant agreement No. 764644). I thank RAINBOW and all the people involved in it for giving me the opportunity to travel and live in different countries in Europe, meeting so many different people from around the world, and giving me a great insight in both academic and industrial environments. In this context I want to take the opportunity to thank Christine Sarah Andersen and Kenny Erleben for planning the RAINBOW events and coordinating the project.

% Synopsys

I also want to thank Synopsys-Simpleware for providing a friendly and social working environment. I specifically want to thank David Raymont for his help in my many technical questions and our lunch breaks :) , Mike Richardson for his advice on one of the most challenging parts of this thesis, and Rita for helping me with all the necessary paperwork for my PhD and most importantly for doing so while bringing her positive and joyful attitude with her.

% Family, Friends and More

Many other people contributed to different degrees to the completion of this chapter of my life. Antoine, I wasn't sure in which section to include you, but I decided that you belong in the friends and family section. Sharing our PhD journeys and supporting each other through all the ups and downs was something invaluable to me. I believe we had a few too many beers, a few too many times, but who is counting! My experience in Paris wouldn't be nearly as good if it weren't for you. See you in Greece! PS: Thank you for letting me know that I apparently like cycling now :P. Elena, thank

you for your constant support throughout this journey. Thanks for putting up with all my spreadsheets and my extremely complicated and yet surprisingly(?) inefficient decision-making process. Sorry that my PhD is so closely related to Neural Networks, I vow to commit myself to raise awareness of the dangers linked to them, including but not limited to: world domination, the Matrix and Skynet. Moreover, I would like to thank all of my friends who made this journey so enjoyable! Last but not least, I'd like to wholeheartedly thank my family, for their continuous emotional support and for believing in me and supporting my decisions even when it wasn't easy for them.

Cheers,
Vasilis, Vasileios, Vasilios

Contents

Chapter 1

Introduction

1.1 Motivation and strategies

Multiscale structural analyses are prominent in mechanical and bio-mechanical engineering (e.g. composite materials such as carbon-reinforced polymers or concrete, porous materials such as bones). Typically, there is a large ratio between the scale of the structure and the scale of the features that create excessive stress concentrations (microscale features). Consequently, analysing these structures using full Finite Element Analyses (FEA) is highly inefficient since the mesh needs to locally be very dense to capture the effect of the microscale features.

Common approaches to tackle multiscale problems usually fall either in the homogenisation or domain decomposition techniques. In both cases the multiscale problem is split into a macroscale and a microscale problem. In homogenisation [Évariste Sanchez-Palencia, 1987; Zohdi and Wriggers, 2005] the aim is to find a homogeneous material that has an equivalent behaviour to the original heterogeneous material. In homogenisation the microscale computations are performed over a representative volume element (RVE), assuming that the macroscale displacement gradients vary slowly over the structure. If that is not the case, for instance due to sharp macroscale geometrical features, domain decomposition methods are employed where the results of homogenisation are applied to the boundary of regions of interest for concurrent microscale corrections to be performed, which are typically computationally more expensive and practically more intrusive than methods based on RVEs [Raghavan and Ghosh, 2004; Oden et al., 2006; Kerfriden et al., 2009; Hesthaven et al., 2015; Paladim et al., 2016].

In this work we propose a multiscale surrogate modelling technique to tackle multi-

scale problems and specifically we focus on porous media with no separation of scales. The proposed technique is a Neural Network (NN) based concurrent scale coupling method. NNs possess a number of characteristics that make them ideal candidates for FE surrogate models while having clear advantages over traditional surrogate models like Gaussian Processes surrogates (GPs) or polynomial chaos surrogates. Classical surrogate models are usually limited to small parameter dimensions, typically 1 to 5, so in practice they cannot accept images and graphs as inputs, which are typical ways to represent the geometry and the FE solution. Two common approaches are used to reduce the dimensionality of the input. First approach is to use handcrafted features [Al-Dirini et al., 2020]. This approach is problem specific, requires expert knowledge and does not yield optimum results. Another approach is to use a linear dimensionality reduction technique, such as Principal Component Analysis (PCA) [Liang et al., 2018; Ziaeipoor et al., 2020]. For the types of random geometries studied in this thesis, linear dimensionality reduction techniques fail to capture attractive spaces of small dimension. Consequently, both these approaches introduce an error in the geometry encoding stage, and thus to the input of the classical surrogate models. A deep NN could be used to perform the geometry encoding prior to using a classical surrogate modelling technique but in this work we choose to explore end-to-end deep NN based surrogate models. We will explore NNs designed to specifically process image and graph data, namely Convolutional (CNN) and Graph (GNN) NNs respectively.

Multiple works that use NNs as FE surrogates can be found in the literature over the last couple of years. A three-step procedure is typically followed. The first step is the training dataset generation step, where many expensive FE simulations are performed in advance, subject to parameter variations (such as geometry, boundary conditions, etc.). The second step is the training step, where a NN is trained to approximate the family of generated solutions. The final step is the inference step, where the trained NN is used online to almost instantly make predictions on unseen data.

We make a distinction between the image-based and graph-based works found in literature. That distinction reflects the structure of this thesis but also the chronological trend in the literature. Firstly, we refer to the image-based works [Nie et al., 2019; Mendizabal et al., 2020a; Sun et al., 2020; Jiang et al., 2021; Wang et al., 2021; Deshpande et al., 2022b]. We have a dedicated section where we discuss them in-depth, [section 3.3.2], but we want to highlight here the work from [Nie et al., 2019]. The authors developed a CNN model, StressNet, for stress field prediction in 2D linear elastic cantilevered structures subjected to external static loads. The 2D CNN based

framework we develop in [section 5] is an adaption of StressNet to multiscale problems.

The CNN based works are typically limited to 2D problems due to the increased computational cost of 3D CNNs. Consequently, the literature dedicated to physics-based simulations using 3D CNNs is rather limited [Mendizabal et al., 2020b; Rao and Liu, 2020]. In this thesis, we propose to follow a different approach, that of geometric learning. Geometric learning can be seen as an extension of previously established NN techniques from Euclidean, grid structures to non-Euclidean, graph and manifold structures. Specifically, in order to circumvent the increased computational cost associate with 3D voxel-based CNNs, we employ GNNs [Sanchez-Gonzalez et al., 2020; Pfaff et al., 2021; Deshpande et al., 2022a] to perform geometric learning on a graph representing the manifold of interest. The porous materials examined in this thesis can be fully described by the surface mesh of the structure without using any bulk information. Thus, the manifold of interest is the surface mesh of the material. In cases where volume information is needed, for instance structures with heterogeneous material properties, GNNs can still be used by operating on the volume mesh, but this might not have the same gain compared to standard voxel-based CNNs. In this case, the computational gain is attributed to the fact that GNNs are more versatile and can operate on variable resolution unstructured meshes in contrast to CNNs that require a constant resolution structured mesh. We discuss in-depth the GNN literature in [section 3.3.3]. The 3D GNN based framework we develop in [section 6] is using the GN Block from [Battaglia et al., 2018] as a convolution block.

There are multiple challenges involved in using NNs to solve engineering problems. The two main ones are described here. First of all, NNs typically require a large number of training data, which in practice is not available in realistic engineering problems since one simulation can take multiple hours or even days. Moreover, in engineering applications the confidence in the prediction is key. The performance of NNs deteriorates quickly when data outside the training data distribution is provided (as we show in [section 2.6]), and classical NN methods do not inform the user of the uncertainty of the prediction. These two problems add additional obstacles to the already challenging task of incorporating NN techniques in real engineering workflows.

1.2 Aims and contributions of this thesis

The aim of this thesis is to create NN based FE surrogate models to make fast and reliable predictions on multiscale problems, by using an as limited as possible training dataset. The contributions of the thesis along with comparison with the state-of-the-art works can be found in detail below.

1.2.1 Multiscale Neural Networks

The first contribution of this thesis is the development of a NN based domain decomposition surrogate model to circumvent the need for direct numerical analysis of stress simulations in porous materials and structures. This NN will take as input an inexpensively computed macroscopic approximation of the stress field, where the pore network is ignored, and a fine scale representation of the geometry of the structure and of the network of microscale pores. The NN will output a corrected stress field that emulates the output of the direct numerical simulation. We produce three such networks, one that operates on pixel based 2D problems, one that operates directly on the 3D mesh used to perform FE simulations, and finally a lightweight network that operates on the centroids of the microscale features.

Most of the CNN and GNN based FE surrogate models in literature do not deal with multiscale problems, like in the works of [Mendizabal et al., 2020a; Pfaff et al., 2021; Deshpande et al., 2022b,a]. Some works do exist where geometrical features are taken into account but the examined cases are of low complexity [Nie et al., 2019; Jiang et al., 2021], where the size of the geometrical features is comparable with the size of the structure and only a few geometrical features are interacting with the boundaries of the structures. In other works, as for instance in [Sun et al., 2020], a complex microstructure exists but the training dataset is generated from a single specimen and with a single FE simulation implying low generalisation ability both in terms of different structures and boundary conditions. Most of the GNN works that take into account geometrical features are focused on cracks [Mylonas et al., 2022; Perera et al., 2022] while in this thesis we focus on spherical inclusions. In [Perera et al., 2022], the examined cases include up to 19 cracks, of the same size, and with only 3 different orientations. In this thesis, we examine cases where multiple fine scale features are interacting with each other, with the boundaries of the structure or both while both the structure and the boundary conditions change from one FE simulation to another, leading to improved

generalisation.

1.2.2 Bayesian Neural Networks

The second contribution of this thesis is the conversion of the developed NNs to Bayesian NNs. To this end, we use the Bayes by Backprop method [Blundell et al., 2015] to convert the deterministic NNs to probabilistic NNs, that can output Credible Intervals (CIs) along with a mean prediction. The CIs are broad for cases where the network is uncertain of the prediction and thus the user is aware that the prediction is less reliable.

Existing works on FE surrogate models are focused on deterministic networks that do not provide any uncertainty information for their prediction [Nie et al., 2019; Mendizabal et al., 2020a; Sun et al., 2020; Guo and Buehler, 2020; Jiang et al., 2021; Wang et al., 2021; Pfaff et al., 2021; Perera et al., 2022; Deshpande et al., 2022a, 2023]. Nonetheless, NNs are notorious for their lack of sensible extrapolation ability when faced with data outside the training data distribution. This might lead to dire consequences in engineering applications. Limited works can be found in the literature where Bayesian NNs are being used as FE surrogates. A good example is the work of [Deshpande et al., 2022b], where the authors employ a similar strategy as the one followed in this thesis. Nonetheless, we highlight that our work in probabilistic CNNs [Krokos et al., 2021] precedes this work. An example from the GNN literature is the work of [Mylonas et al., 2022] where they use the local reparameterisation trick [Kingma et al., 2015] to create a Bayesian GNN. In this thesis we use a generalisation of the Gaussian reparameterisation trick [Opper and Archambeau, 2009; Kingma and Welling, 2014; Rezende et al., 2014] as described by [Blundell et al., 2015] which we find to yield better results.

1.2.3 Data requirements and generalisation ability

Limited data availability is one of the main reasons NN techniques are not being used in real engineering applications today. This is being reported by many Machine Learning (ML) practitioners like for instance in [Rajender and Samanta, 2023], where the authors state that researchers have been trying to use ML techniques in diverse fields like structural performance, structural health monitoring, and concrete mechanics but a major drawback is the lack of training datasets. Another example is the work of [Niu et al., 2023] where the authors mention that an optimally trained ML model could be

used to decipher crack characteristics, but lack of labelled datasets is a major limitation. This commonly leads to one of the two following problems. If synthetically generated data can be used, then researchers have to create very large datasets like for instance in [Croom et al., 2022] where a dataset consisting of 100,000 random microstructure images was generated. In cases where this is not possible, researchers train their ML models using the available data, but the resulting model suffers from low generalisation capacity [Penido et al., 2022].

In the computer vision field, data augmentation techniques are standard practice to increase the performance of CNNs by artificially increasing the size of the dataset. The most common data augmentation techniques are shifting, flipping, rotating and zooming. Although some of these techniques have successfully been applied to engineering problems like microstructure segmentation [Goetz et al., 2022], they have not yet been adopted for regression tasks. This can be attributed to the fact that their extension to regression NNs is not straightforward. In this thesis we attempt to bridge this gap. To this end, we demonstrate the use of mechanically consistent rotation as a data augmentation step, for the developed regression CNN. We show how introducing this step helps the network to increase its generalisation ability in cases where the training dataset is not sufficiently large. Another common technique from the computer vision field that could be used to reduce the training data requirements is transfer learning. Examples can be found in the literature where transfer learning has successfully been employed to train CNNs for material microstructure segmentation [Goetz et al., 2022; Stuckner et al., 2022]. In this thesis, we extend the use of transfer learning from CNN segmentation tasks to GNN regression tasks.

Two more techniques are being studied in this thesis to reduce the size of the training dataset. Both of these techniques are based on the uncertainty extracted from the Bayesian NNs. The first technique is selective learning, which is extensively studied in [Islam, 2016; Gal et al., 2017]. This technique allows the ML researcher to train an initial NN with a small dataset and then based on the uncertainty of the network to select a small subset of the unlabelled data to label. We highlight here, that labelling data can be very challenging either computationally or in terms of time or financial resources. This technique is commonly used in medical segmentation networks [Ozdemir et al., 2021], and it is implemented in popular open-source ML platforms [Cardoso et al., 2022]. Also in fields like material science, researchers have used selective learning to accelerate the search and discovery of new materials [Lookman et al., 2019]. The literature in regression tasks is very scarce and not focused on engineering applications [Tsymbalov

et al., 2018]. Consequently, in this thesis we apply selective learning in a regression CNN with the aim to reduce the labelled data requirements and thus the computational cost of creating the training dataset. Lastly, we propose to use the Ensemble Kalman method to apply online stress corrections in cases where the accuracy of the NN is low because of the lack of appropriate training data. We stress that this is different to Physics Informed Neural Networks (PINNs) where physical constraints are enforced during training [Raissi et al., 2019, 2020; Hennigh et al., 2021]. In this work we do not explore PINNs, although it is an interesting future direction.

1.3 Thesis structure

- Chapter [2], is dedicated to Machine Learning and specifically Neural Networks. We explain the basic principles behind different types of NNs starting from the most basic ones and we put emphasis on the ones we use in this thesis.

- In Chapter [3], we provide a literature review of different NN based techniques used to tackle engineering problems. We also review two common approaches to solve multiscale problems, namely homogenisation and concurrent scale modelling. Lastly, we discuss works from the literature that utilise NN assisted methods for the solution of multiscale problems.

- In Chapter [4], we introduce the proposed multiscale approach along with typical structures of interest. We also refer to key concepts like the equations whose solution we try to reproduce and the Quantity of Interest used in this thesis.

- In Chapter [5], we apply the developed multiscale framework to a 2D image-based problem, using a CNN. We study the network's performance and compare it to homogenisation. We also evaluate the quality of the uncertainty extracted from the network and we demonstrate how to use it in a selective learning framework to reduce the number of fine scale simulations needed to train the model. Lastly, we demonstrate how to use mechanically consistent rotations as a data augmentation technique to reduce the training data requirements.

- In Chapter [6], we extend the 2D CNN to a 3D GNN, and we apply it to 3D mesh-based multiscale problems. We study the network's performance using problems of different complexity. We also evaluate the extracted uncertainty and we

demonstrate how to use it in an online stress correction technique to improve the network's prediction.

- In Chapter [7], we modify the 3D GNN to operate on the centroids of the microscale features, creating a lightweight model that we call centroid GNN. We study the network's performance using both synthetic and real data. Additionally, we employ a transfer learning technique to take advantage of the knowledge gained from one dataset, to reduce the training data needed to make predictions on a different dataset. Lastly, we evaluate the quality of the uncertainty extracted from the network and we demonstrate how to use this uncertainty in a Bayesian optimisation framework to identify microscale features associated with the maximum stress in a dataset.

1.4 Publications

Part of the work contained in this thesis has also appeared in the following papers, presentations, and workshops:

1.4.1 International journals

- Vasilis Krokos, Stéphane P.A. Bordas, Pierre Kerfriden: A graph-based probabilistic geometric deep learning framework with online enforcement of physical constraints to predict the criticality of defects in porous materials, *International Journal of Solids and Structures*, 2023

- Vasilis Krokos, Viet Bui Xuan, Stéphane P.A. Bordas, Philippe Young, Pierre Kerfriden: A Bayesian multiscale CNN framework to predict local stress fields in structures with microscale features, *Computational Mechanics*, 2022

1.4.2 Conference papers, presentations and workshops

- Vasilis Krokos, Viet Bui Xuan, Pierre Kerfriden: A 3D Bayesian Multiscale Graph Neural Network Framework To Predict Local Stress Fields In Structures With Microscale Features. Presentation in the 18^{th} European Mechanics of Materials Conference, 4-6 April 2022 (ORCA entry)

- Vasilis Krokos, Viet Bui Xuan, Pierre Kerfriden: Probabilistic predictions of equivalent stress fields in heterogeneous materials using GraphCNNs and variational dropout. Workshop attendance and presentation in the Physics and Image workshop at Mines ParisTech, 15^{th} of June 2021

- Vasilis Krokos, Pierre Kerfriden, Philippe Young, Viet Bui Xuan, Stéphane Bordas: Data-Driven Multiscale Computational Modelling in Biomechanics using Convolutional Neural Networks. Paper submitted at WCCM 2020 planned for Paris, 19-24 July 2020 (rescheduled to virtual format 11-15 January 2021) (ORCA entry)

- Vasilis Krokos, Viet Bui Xuan, Stéphane Bordas, Philippe Young, Pierre Kerfriden: Bayesian convolutional neural network as probabilistic surrogates for the fast prediction of excess stress in structures with microscale patterns, local features or defects. Workshop attendance and presentation on Model Order Reduction and Probabilistic Model Learning Workshop @ Centre des Matériaux, 17^{th} of September 2020

Chapter 2

Machine Learning

In this chapter we introduce Neural Networks (NNs) using linear regression. By highlighting the fundamental components of linear regression, we get the opportunity to see similarities with NNs. Specifically, we refer to key concepts like the type of NN (fully connected, convolutional, etc.), the loss function, the activation function and the optimiser.

2.1 Introduction

In the last decade, Machine Learning (ML) and specifically Deep Learning has attracted tremendous attention and an increasing number of researchers are working actively in applying and improving the state-of-the-art techniques. Nevertheless, the theory behind NNs exists from the early 1950s, and it was mostly the lack of appropriate computational resources and realistic expectations that hindered the exponential growth of the field that we see nowadays. Specifically, in terms of Artificial Intelligence (AI), the 1970s is known as "AI winter" (an analogy to nuclear winter), because of the reduced funding and interest in the field caused by pessimism about ML effectiveness. In contrast to that, we are now in the era of "AI spring" attributed to a number of successful AI projects. Some examples include, language translation (Google Translation), image recognition (The ImageNet competition [Russakovsky et al., 2015]), various game-playing ML models such as AlphaZero [Silver et al., 2017] (chess) and AlphaGO (Go) [Silver et al., 2016] both developed by DeepMind, and most recently tackling the Protein Structure Prediction problem [Tunyasuvunakool et al., 2021].

In the next sections we will explore the basics of NNs by reviewing the basic types

of NNs with emphasis on the 2 types we use in this PhD work.

2.2 Linear regression

A common way to introduce NNs is through linear regression, which can be seen as a NN with fully connected layers under specific conditions. Given a set of N 1-dimensional pairs of input/output data points $[(x_1, y_1), ..., (x_N, y_N)]$ we try to "train" a linear model, $f(x)$, to map the inputs, x, to the outputs y. This model has parameters W and b and is of the form

$$f(x) = xW + b \tag{2.1}$$

Training the model means to find the parameters $\theta = (W, b)$ so that the model "best" fits the data. How well the model fits the data is determined through an error function that needs to be minimised. The error function used in the case of linear regression is the sum of the squared errors between the data and the model prediction divided by the number of data points

$$\mathbf{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \|y_i - f(x_i)\|^2 \tag{2.2}$$

In the context of NNs the error function is called the "loss function", and the specific loss function introduced in equation [2.2] is called the Mean Squared Error (MSE).
For the linear model introduced above, the minimisation problem of equation [2.2] has an analytical solution

$$W^{opt} = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{2.3a}$$

$$b^{opt} = \bar{y} - W^{opt}\bar{x} \tag{2.3b}$$

where $\bar{x}$ and $\bar{y}$ are the mean values for x and y respectively.

As can be seen in the code snippet below in a few lines of code we can create a simple dataset and fit a linear function to it. Visualisation of the results can be found

in [Fig 2.1]

```
import numpy as np
from sklearn import linear_model

def create_dataset(N, e=1):

    x = np.linspace(-1, 1, num=N)
    y = 10*x + 3

    # add random noise to the data
    y += np.random.normal(0, e, x.shape[0])

    return x.reshape(-1,1), y.reshape(-1,1)

# create dataset
x, y = create_dataset(N)

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model
regr.fit(x, y)

# Make predictions
y_pred = regr.predict(x)
```

After introducing this simple linear regression problem we will now move to regression problems in terms of neural networks.

2.3 Fully connected Neural Networks

NNs with fully connected layers is the most basic and easy to explain type of NNs. We choose to start with them since we can easily compare them with the linear regression model we saw in the previous section but also because other types of NNs, as for instance Convolutional and Graph NNs, tend to use fully connected layers as part of their architecture.

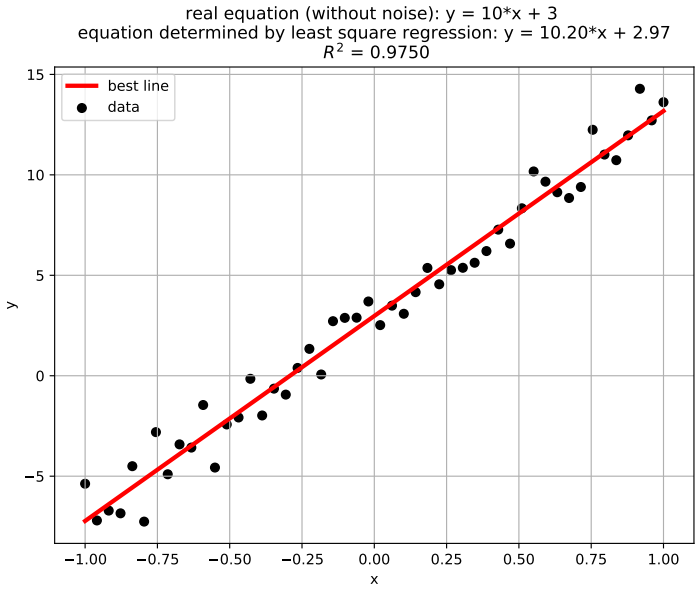


Figure 2.1: In this figure we can observe the line that best fits a set of data points.

2.3.1 Single hidden layer NNs

Given N pairs of input/output data points $[(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_N, \mathbf{y}_N)]$ where $\mathbf{x}_i \in \mathbb{R}^Q$ and $\mathbf{y}_i \in \mathbb{R}^D$ we try to find a function, $\mathbf{f}(\mathbf{x})$, to map the inputs $\mathbf{x}$, commonly referred to as input layer in the NN terminology, to the outputs $\mathbf{y}$. This function will be a NN with a single hidden layer, a sketch of which can be seen in [Fig 2.2] [1], where $Q = 3$, $D = 2$ and the hidden layer has $K = 6$ neurons. The single hidden layer NN is composed of the input layer, a hidden layer and finally the output layer.

The output of the hidden layer is called activation of the layer and can be calculated as follows

$$\mathbf{z} = \sigma(\mathbf{x}\mathbf{W_1} + \mathbf{b_1}) \tag{2.4}$$

where $\mathbf{W_1}$ is the weight matrix of size $Q \times K$, $\mathbf{b_1}$ is the bias vector that holds K elements and finally $\sigma(\cdot)$ is an element-wise non-linearity that is called activation function in the context of neural networks. Here we use the Rectified Linear (ReLU) activation function, ReLU(x) = $\mathbf{max}(0,x)$, but other options exist such as the hyperbolic tangent function ($tanh(x)$) and the sigmoid ($\frac{1}{1+e^{-x}}$).

Currently the activation of the hidden layer is a matrix of size $N \times K$, which is

[1] The NN is drawn using the free online tool "NN-SVG" [LeNail, 2019]

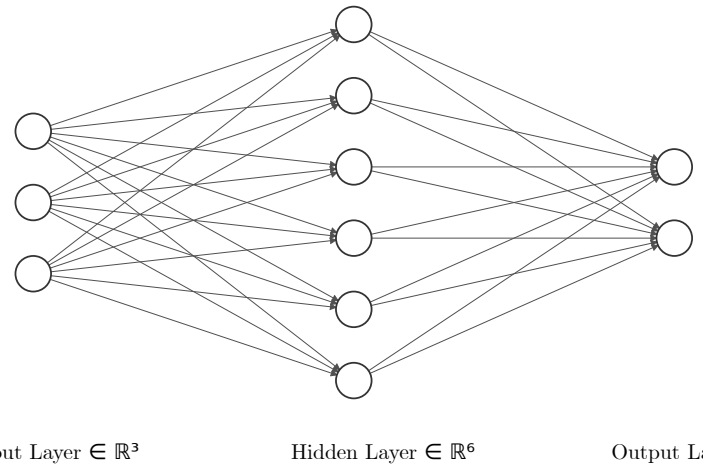


Figure 2.2: We can see a sketch of a single hidden layer fully connected NN. The input is of dimension 3, the hidden layer has 6 dimensions and finally the output layer has 2 dimensions

inconsistent with the output matrix that needs to be of size $N \times D$. The output layer transforms the activation in the following way

$$\hat{\mathbf{y}} = \mathbf{z}\mathbf{W_2} + \mathbf{b_2} \tag{2.5}$$

where $\hat{\mathbf{y}} = \mathbf{f}(\mathbf{x})$ is the output of the network, $\mathbf{W_2}$ is the weight matrix of the output layer of size $K \times D$ and $\mathbf{b_2}$ is the bias vector holding D elements.

We ultimately want to find the parameters, $\theta = (\mathbf{W_1}, \mathbf{b_1}, \mathbf{W_2}, \mathbf{b_2})$, so that the NN fits the data. To this end we need to formulate a loss function that depends on these parameters and minimise it. The MSE is usually a sensible choice of a loss function in regression tasks. The loss function is minimised using an iterative method. In the NN terminology this iterative method is called optimiser. For the optimiser let us assume that we use standard Stochastic Gradient Descend (SGD), which is a well-known algorithm in the field of computational sciences. We further discuss optimisers in section [2.7].

This single hidden layer NN is equivalent to the linear regression model we analysed in section [2.2], under one condition. The activation function has to be the linear activation function, which is basically the identity function. In fact, a NN with any number of fully connected layers, is exactly equivalent to the linear regression model as long as it only uses linear activation functions. The equation of the single hidden layer NN becomes

$$\hat{\mathbf{y}} = \mathbf{x}\mathbf{W}^* + \mathbf{b}^* \tag{2.6a}$$

$$\mathbf{W}^* = \mathbf{W_1}\mathbf{W_2} \tag{2.6b}$$

$$\mathbf{b}^* = \mathbf{b_1}\mathbf{W_2} + \mathbf{b_2} \tag{2.6c}$$

Below we show the few lines of code required to train a NN with a single hidden layer to solve the same problem posed for the linear model in section [2.2] and compare the 2 solutions. From [Fig 2.3] we can see that the 2 solutions are indeed the same. If we repeat the same procedure with any number of layers we will get the same results because without non-linearities (introduced by the activation functions) the fully connected NN is just a linear model.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# get shape of input output and define neurons in hiddel layer
input_shape  = x.shape[-1]
output_shape = y.shape[-1]
hidden_dim   = 3

# build model
model = Sequential()
model.add(Dense(hidden_dim, input_dim=input_shape))
model.add(Dense(output_shape))

# compile model; define loss function and optimiser
model.compile(loss='MSE', optimizer='SGD')

# fit the keras model on the dataset
model.fit(x, y, epochs=100, batch_size=10)

# make predictions with the model
y_NN = model.predict(x)
```

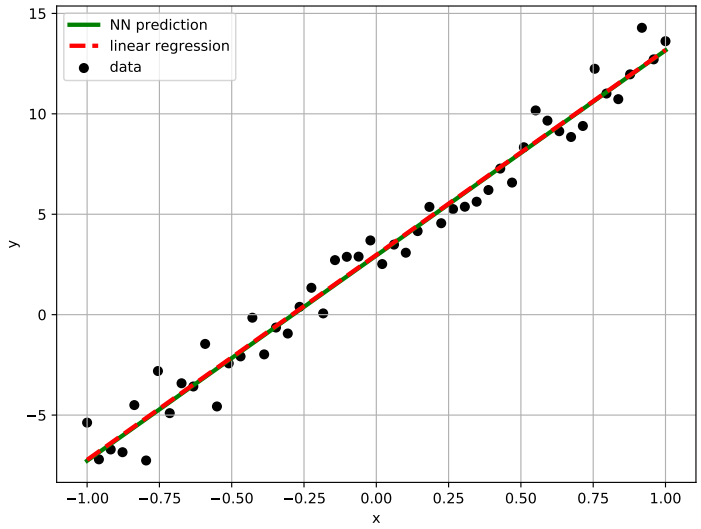


Figure 2.3: In this figure we compare the fitting of a linear model and a NN with the linear activation functions on the dataset introduced in [2.2].

2.3.2 Multi-layer NNs

It is very straight forward to extend the single layer architecture to a neural network of any number of hidden layers - although training a very deep model is a challenging task for numerical reasons. For an input $\mathbf{x}$, the output, $\mathbf{f}(\mathbf{x})$, of a multi-layer NN with P hidden layers is given by the following equations

$$\mathbf{f}(\mathbf{x}) = \sigma_{P+1}(\mathbf{z}_P \mathbf{W}_{P+1} + \mathbf{b}_{P+1}) \tag{2.7a}$$

$$\mathbf{z}_p = \sigma_p(\mathbf{z}_{p-1} \mathbf{W}_p + \mathbf{b}_p) \tag{2.7b}$$

$$\mathbf{z}_1 = \sigma_1(\mathbf{x} \mathbf{W}_1 + \mathbf{b}_1) \tag{2.7c}$$

where $\mathbf{z}_p$ is the activation of a hidden layer p, σ_p is the activation function of the hidden layer p and finally $\mathbf{W}_p$ and $\mathbf{b}_p$ are the weights and biases of the hidden layer p respectively, the subscript $P+1$ refers to the output layer.

2.4 Convolutional Neural Networks

In this section we will explain the basics of Convolutional Neural Networks (CNNs), NNs specifically designed to process image data. We will start from a brief historical review demonstrating how CNNs helped advance the field of deep learning in general and we mention the main contributions that are still considered standard practice in deep learning today.

2.4.1 Introduction

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) played a key role in advancing the area of computer vision and deep learning. In 2012, AlextNet [Krizhevsky et al., 2012] won the ILSVRC competition and became one of the most influential works in the field of CNNs. The most important innovations were the use of ReLU activation function, in contrast to more traditional tanh and hyperbolic tangent, and the use of GPUs for training. Two years later, Google's GoogLeNet [Szegedy et al., 2015], introduced the concept of Inception blocks in an attempt to train deeper NNs. Inception blocks leverage feature detection at different scales to approximate an optimal local sparse structure in a CNN. In 2015, Microsoft's ResNet [He et al., 2016] was the next breakthrough introducing the concept of residual connections that allows the training of very deep neural networks of hundreds of layers. Residual connections are necessary because even though in theory the interpolation ability of the network should increase with the number of layers, in practice training deep NNs is very challenging due to problems like vanishing or exploding gradients causing the NN to not be able to learn simple functions like the identity function between input and output [Hochreiter et al., 2001; Sussillo and Abbott, 2015]. Residual connections allow the NN itself to choose its depth by skipping the training of a few layers [Kim et al., 2016; Zagoruyko and Komodakis, 2016; Lim et al., 2017]. Lastly, in 2017 the SENet network [Hu et al., 2018] introduced the Squeeze-and-Excitation (SE) block that can adaptively recalibrate channel-wise feature responses by explicitly modelling interdependencies between channels resulting in improved generalization across datasets and improvement in the performance [Cheng et al., 2018; Li et al., 2018].

2.4.2 Convolution operation

One of the main characteristics of CNNs is the convolution operation. We can see an example of a convolution between an image and a filter in [Fig 2.4]. In the figure, the filter is applied on the top left corner of the image. In reality we can imagine the filter sliding along both the horizontal and vertical direction to iteratively cover the entire original image, although this is omitted for simplicity. Filters can be used to detect features in images. For example in [Fig 2.5] we show the effect of applying 2 filters in the original image. These 2 filters are known as Sobel filters and can be used to detect horizontal and vertical edges. Other filters, or more commonly a lot of filters applied one after the other, can be used to detect more complex features as for instance faces. Filters of different size detect different sized features. A common choice for the size of the filters is 3×3 but also 5×5 and 7×7 are used for larger images.

The main idea behind CNNs is that the parameters of the filters do not have to be defined by hand but they should be considered as parameters that the network will optimise in order to minimise the loss function. Training the network ultimately results in defining these parameters. Typically, in CNNs with multiple convolutional layers, the first layers of the CNN detect low level features, like horizontal and vertical lines, and as we go further in the network the filters detect higher level features, as for example faces.

As we show in the previous paragraph, the image size before and after the convolution with a filter of size f [3] is not the same. In the simplest case, for an image with input size a_{in} and output size a_{out} we have $a_{\text{out}} = a_{\text{in}} - f + 1$. In more realistic cases the relationship between input and output size is governed by two hyperparameters, namely padding p, and stride s

$$a_{\text{out}} = \lfloor \frac{a_{\text{in}} + 2p - f}{s} + 1 \rfloor \tag{2.8}$$

- **padding**(p): Due to the nature of the convolution operation the output image reduces in size. This leads to problems with very deep neural networks because after applying a lot of convolutional layers the image will get too small. Additionally, the pixels on the boundary of the image and especially the corner pixels are visited less times by the filters compared to the rest of the pixels. To tackle both

[2]Photo of capybara provided by Karen Lau on Unsplash.

[3]From now on we will assume that all images and filters have the same pixel number along both directions for simplicity, nevertheless all the equations directly extend to non-square images and filters.

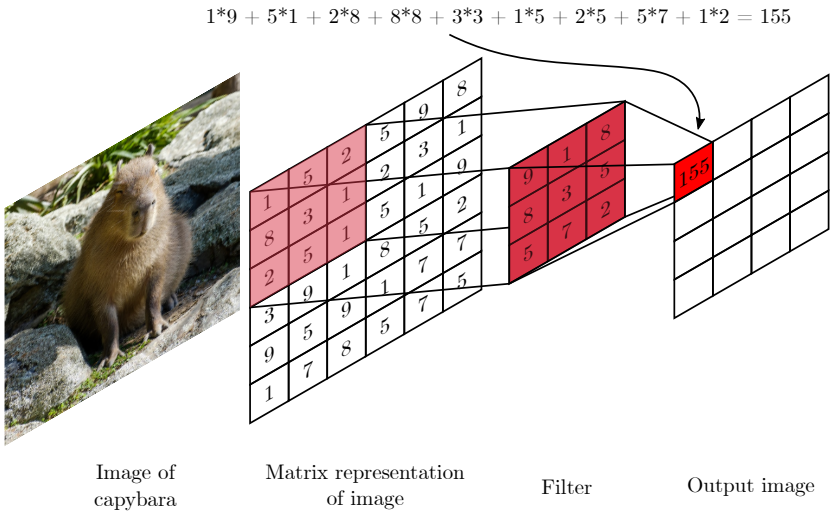


Figure 2.4: In this figure we can see a schematic representation of a convolution operation. The input image is of size 6×6, the filter is of size 3×3 and the output image is of size 4×4. We note that the RGB images have 3 channels with values from 0 to 255 but in this figure, we consider a 1 channel representation of the image (as for instance greyscale) with values from 1 to 9, for simplicity reasons. [2]

these problems padding was introduced. Padding adds a layer of pixels around the image, usually with value zero. The "thickness" of this layer is determined by the padding size. A common padding strategy is called "same padding" in the NN literature. This strategy chooses the padding size so that the image before and after the convolution has the same size. Using [Eq 2.8] and by setting $s = 1$ [4] and $a_{\text{in}} = a_{\text{out}}$ we find that $p = \frac{f-1}{2}$ in the case of "same padding".

- **stride**(s): Another hyperparameter that affects the convolution is the size of the step taken when sliding the filter across the image. Until now it was implied, although never explicitly stated, that the filters move one pixel along the vertical and horizontal direction. This is not necessarily true and the step size is determined by the stride parameter. Most commonly the stride is set to 1, although it can be set to higher numbers which results in downsampling the image, and focusing on the higher level features, which will lead to a computationally cheaper problem but might affect performance.

[4] Stride is set to one since it is only used when we explicitly want to reduce the size of the image so it makes no sense to use $s > 1$ with "same padding".

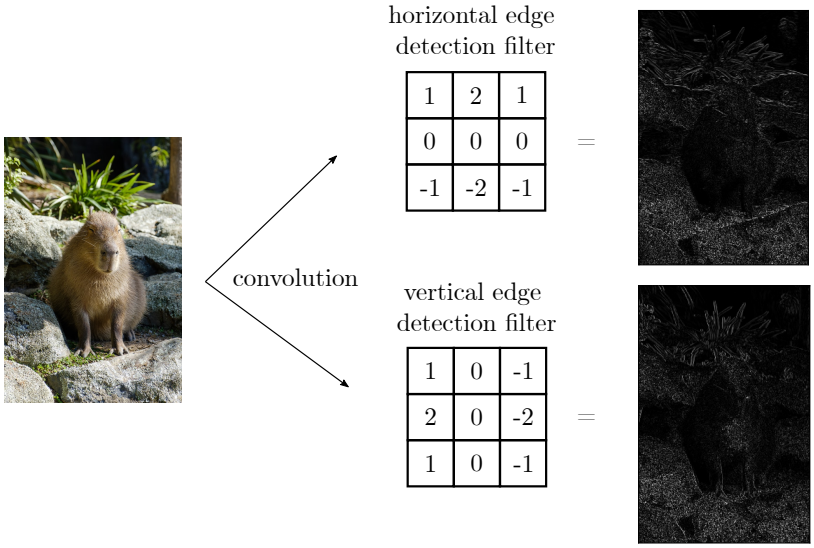


Figure 2.5: In this figure we can observe the effect of applying 2 filters, one for vertical and one for horizontal edge detection, on an image

2.4.3 Pooling

Another type of layer that is used in CNNs is the pooling layer. This layer has no parameters to be learned. It is used to downsample the image to reduce the computational cost and it also makes the network more robust to changes in the position of the features in the image. This is a step towards translation invariance which can also be achieved through data pre-processing by randomly translating the original images. In tasks where the exact position of the objects is not important, like in object recognition tasks, translation invariance is highly desirable. On the other hand, in tasks where the exact position of the pixels is important, like in segmentation tasks, translation invariance is not desirable. The most common type of pooling used in practice is max pooling, an example of which can be found in [Fig 2.6], and less frequently average pooling may be used by ML practitioners. We note that pooling can take place in a sliding window fashion, same as for the convolution operation, by defining a filter size, a stride and padding.

2.4.4 Upsampling

So far, we have introduced 2 operations, both of which can be used to downsample an image. As we will see in the next chapters of this thesis, some architectures require image upsampling. There are a lot of different ways to upsample an image, some

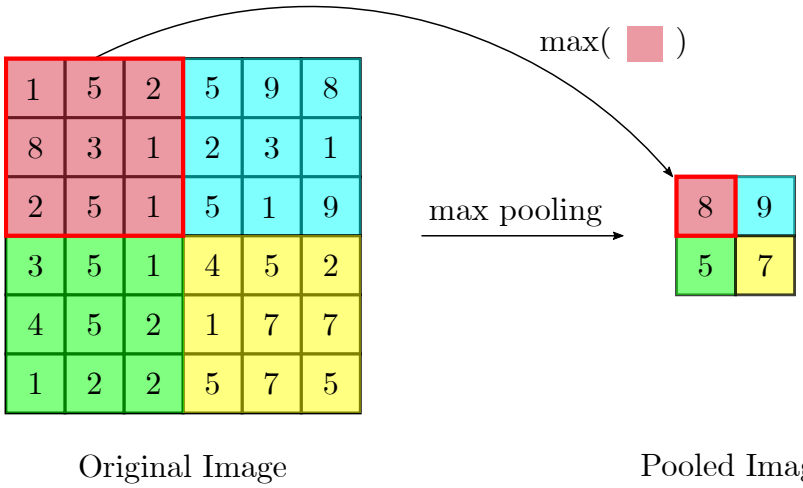


Figure 2.6: Max pooling on a 6×6 image, with $f = 3$, $s = 3$ and $p = 0$

of which have no learnable parameters and thus are data agnostic while others use learnable filters whose values can be determined during network training. Two examples of data agnostic operations are "Nearest Neighbour" [Fig 2.7] and "Bed of Nails" [2.8]. On the other hand, we can combine a convolution and upsampling layer to create a transposed convolution layer that has learnable parameters [Fig 2.9]. This operation can be viewed as a convolution with a fractional stride.

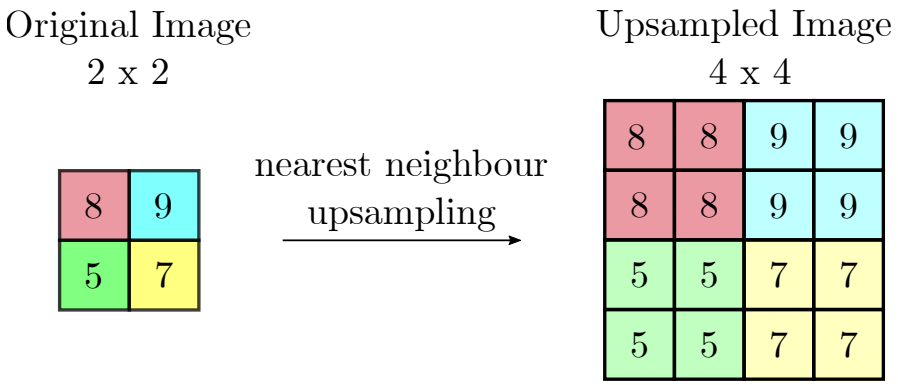


Figure 2.7: Upsampling a 2x2 image to a 4x4 image using nearest neighbour upsampling

2.4.5 Simple CNN

Below we provide the equations for a CNN with P convolutional layers. Assuming an image A and filters f_p we have

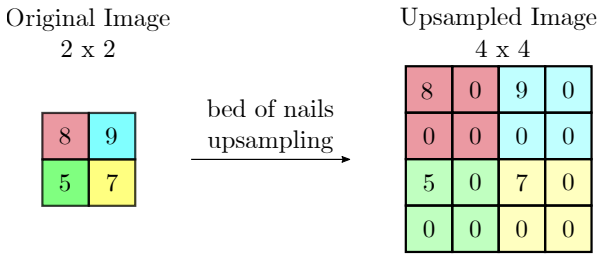


Figure 2.8: Upsampling a 2x2 image to a 4x4 image using bed of nails upsampling

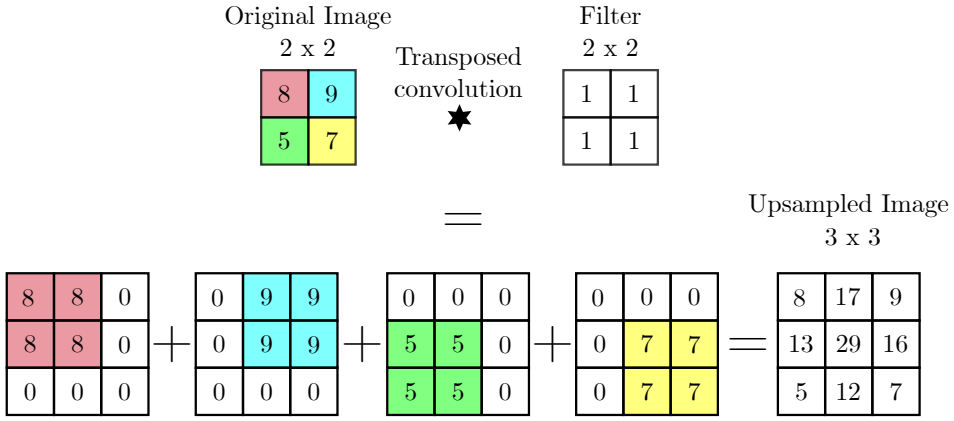


Figure 2.9: Upsampling a 2x2 image to a 3x3 using transposed convolution

$$\mathbf{f}(\mathbf{x}) = \sigma_{P+1}(z_P * f_{P+1} + b_{P+1}) \tag{2.9a}$$

$$z_p = \sigma_p(z_{p-1} * f_p + b_p) \tag{2.9b}$$

$$z_1 = \sigma_1(A * f_1 + b_1) \tag{2.9c}$$

where $*$ denotes the convolution operation.

2.4.6 Example of a CNN

Below we see a CNN that can be used for a 10-class classification problem [Fig 2.10]. The CNN uses convolutional layers for feature extraction and in the end some fully connected layers for the classification. The activation of the last layer is called "softmax" and it will output 10 numbers summing to 1. Each number corresponds to the probability that the input image belongs to one of the classes. In table [2.1] we show how the shape of the image changes inside the network and also the number of trainable parameters.

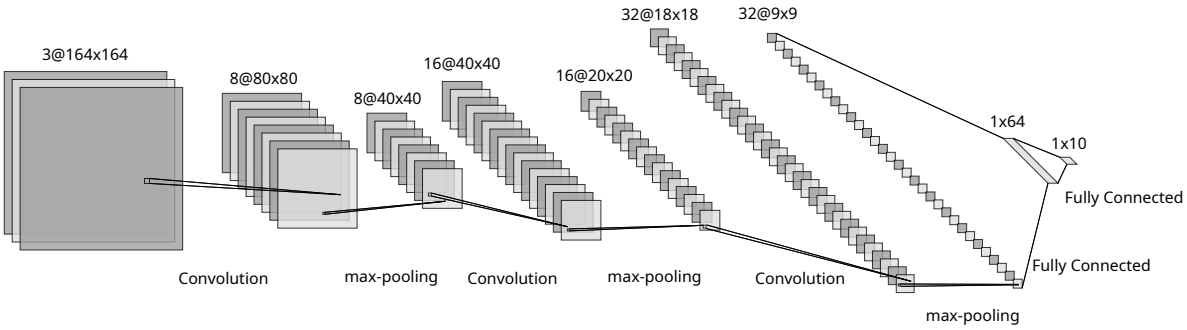


Figure 2.10: Sketch of a multilayer CNN with Convolutional, Pooling and Fully Connected layers

	output shape	number of parameters
Input	$164 \times 164 \times 3$	0
Conv(f=5, padding=0, filters = 8, s=2)	$80 \times 80 \times 8$	608
Max Pooling(f=2, s=2)	$40 \times 40 \times 8$	0
Conv(f=3, padding=same, filters = 16, s=1)	$40 \times 40 \times 16$	1,168
Max Pooling(f=2, s=2)	$20 \times 20 \times 16$	0
Conv(f=3, padding=0, filters = 32, s=1)	$18 \times 18 \times 32$	4,640
Max Pooling(f=2, s=2)	$9 \times 9 \times 32$	0
Flatten	2,592	0
FC(neurons = 64)	64	165,952
FC(neurons = 10)	10	650

Table 2.1: Image shape and number of parameters in the layers of a CNN.

2.4.7 Comparison with fully connected layers

As already mentioned, CNNs were developed to work with images, but one may wonder why not use the standard fully connected layers to do that. Indeed an image of any size can be flattened and each pixel can be considered as an input feature. Let us discuss why this seemingly natural approach leads to both theoretical and practical problems.

Let us consider a single channel 128×128 image, which for today's standards is a very low-resolution image. [5] If we apply 100 3×3 filters on this image, because the weights are shared, we end up with $100 \times 3 \times 3 = 900$ weights. On the other hand, if we connect

[5]For comparison I will mention that the original capybara image used in section [2.4.3] was of resolution 3066×4599 and a mid-range smartphone can take 4032×3024 photos.

this input layer with a single hidden layer of 100 neurons we end up with $128 \times 128 \times 100 = 1,638,400$ weights! The difference is striking. We note that training networks with large number of parameters is associated with increased memory requirements, slower training times and difficulties for the optimiser to converge. Additionally, in order to avoid overfitting, training requires a very large training dataset.

The advantage of convolutional layers over fully connected ones is amplified if we consider that the number of weights in the CNN remains the same regardless of the size of the image, while in the fully connected layers it scales linearly.

Even in the case where computational resources and data availability is not a problem, we could argue that convolutional layers are more suited for dealing with images. In contrast to fully connected layers, convolutional layers take advantage of locality and translation equivariance. Locality means that pixels which are closer to one another more strongly affect each other compared to pixels that are far away. In the context of CNNs, translation equivariance means that an object can be detected regardless of its position in the image. This is achieved through weight sharing in convolutional layers.

2.5 Graph Neural Networks

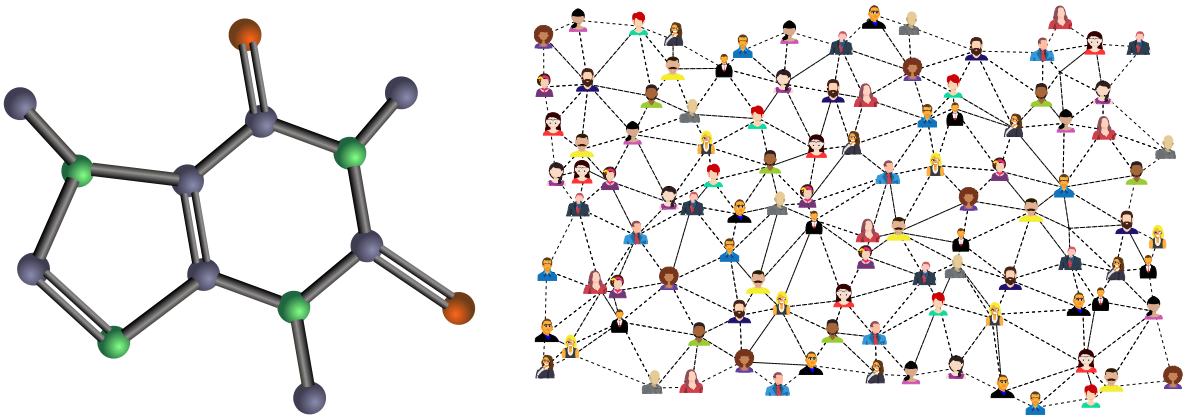
In this section we will introduce Graph Neural Networks (GNNs), NNs that are designed to operate on graph structures in contrast to images. We start by giving a few examples of problems that can be tackled with GNNs, continue with a literature review of GNNs and lastly we give examples of graph convolutions.

2.5.1 Introduction

As demonstrated in the previous section CNNs have advantages over NNs with fully connected layers in cases where the input is an image. Nevertheless, there is a great variety of data that cannot be represented using images but instead have a graph structure. Examples include molecules [Fig 2.11a] and social networks [Fig 2.11b]. In order to analyse data with underlying graph structure researchers developed GNNs.

2.5.2 Literature review

GNNs have been massively developed over the last years with various works on segmentation [Hanocka et al., 2019; Schult et al., 2020; Lei et al., 2021] and shape corre-



(a) (b)

Figure 2.11: On the left (a), a sketch of a caffeine molecule and on the right (b) a sketch of a social network. Images by OpenClipart-Vectors (a) and Gordon Johnson (b) from Pixabay.

spondence or retrieval tasks [Masci et al., 2015; Gong et al., 2020].

One of the first works on deep learning applied to Non-Euclidean data is described in [Qi et al., 2017a]. The proposed network, PointNet, operates on point clouds. PointNet has a simple architecture where the input point cloud is first processed through 2 shared multi-layer perceptrons (MLPs) that map the three-dimensional input to a latent space of 1024 dimensions. After that, a max pooling layer is used to create global features and finally a NN with 3 fully connected layers is used to predict the classification scores. PointNet is applied to unordered point clouds and thus it is permutation and translation invariant. This is achieved by the use of a symmetric aggregation function, max pooling, and pose normalisation respectively. PointNet is followed by PointNet++ [Qi et al., 2017b] which is better in capturing local structures and thus in recognising fine-grained patterns resulting in increased generalisability in complex scenes. PointNet++ builds a hierarchical grouping of points that are then processed with PointNet. Another network inspired from PointNet is called EdgeConv [Wang et al., 2019]. EdgeConv operates on point clouds and it successfully captures local geometry information while at the same time achieves permutation invariance. In contrast to the previous approaches, EdgeConv creates edge features between points and their neighbours. The edge features are created through an edge function that takes as input the node features of the node and its neighbours. The edge function drives the behaviour of the network since it defines how information is diffused between points and their neighbours. The most common edge function, referred to as asymmetric edge function in the original work,

explicitly combines the global shape structure with local neighbourhood information. The edge features are then passed through an MLP to update their values. Lastly, a channel wise symmetric aggregation function is applied to the updated features resulting in a permutation invariant network.

Apart from point clouds, a more informative way to represent 3D objects is through meshes. An early attempt for geometric learning on meshes is introduced by [Hanocka et al., 2019]. The proposed network is called MeshCNN and can be used for classification and segmentation tasks. Because MeshCNN operates directly on the mesh it can take advantage of the connectivity of the mesh which offers increased topological information compared to operating on a point cloud. MeshCNN defines convolution and pooling operations directly on the edges of mesh. The most interesting feature of MeshCNN is the goal oriented pooling operation that it introduces, that gradually drops edges of the initial mesh and results in a new mesh with only the most informative edges for the given task. For the mesh convolution the authors define 5 features on the edges of the mesh (the dihedral angle, two inner angles and two edge-length ratios for each face) and then use symmetric functions to process pairs of opposing edges and thus the convolution is invariant to edge order. MeshCNN result in better accuracy compared to PointNet++.

2.5.3 Graph convolutions

As seen in the previous sections, an image convolution is defined as a mathematical operation that involves sliding a small matrix, filter, over each local region of the image, and computing a dot product between the values of the filter and the pixels in the local region. On the other hand, in graph data the neighbourhood is not uniform and varies based on the underlying graph structure. Researchers use different mathematical operations to define convolution on graphs. In general, these operations can be divided in spectral and spatial methods. Spectral methods make use of the graph Laplacian matrix and its eigendecomposition to define convolution operations, while spatial methods are based on directly applying convolution on the nodes, edges or the neighbourhood of the graph.

- **Spectral methods**: These methods operate on the frequency instead of the spatial domain. According to the convolution theorem, image convolution on the spatial domain is equivalent to matrix multiplication in the frequency domain.

This is achieved using Discrete Fourier Transformations, which unfortunately requires a regular grid (which is compatible to image but not graph structures). In order to extend this idea from images to graphs we need to define a more generic basis, namely the eigenvectors of the graph Laplacian.

The Laplacian, L, is a matrix representation of the graph. It can be calculated using the Adjacency matrix, A, and the Degree matrix, Deg, of the graph as $L = Deg - A$. An example can be found in [Fig 2.12].

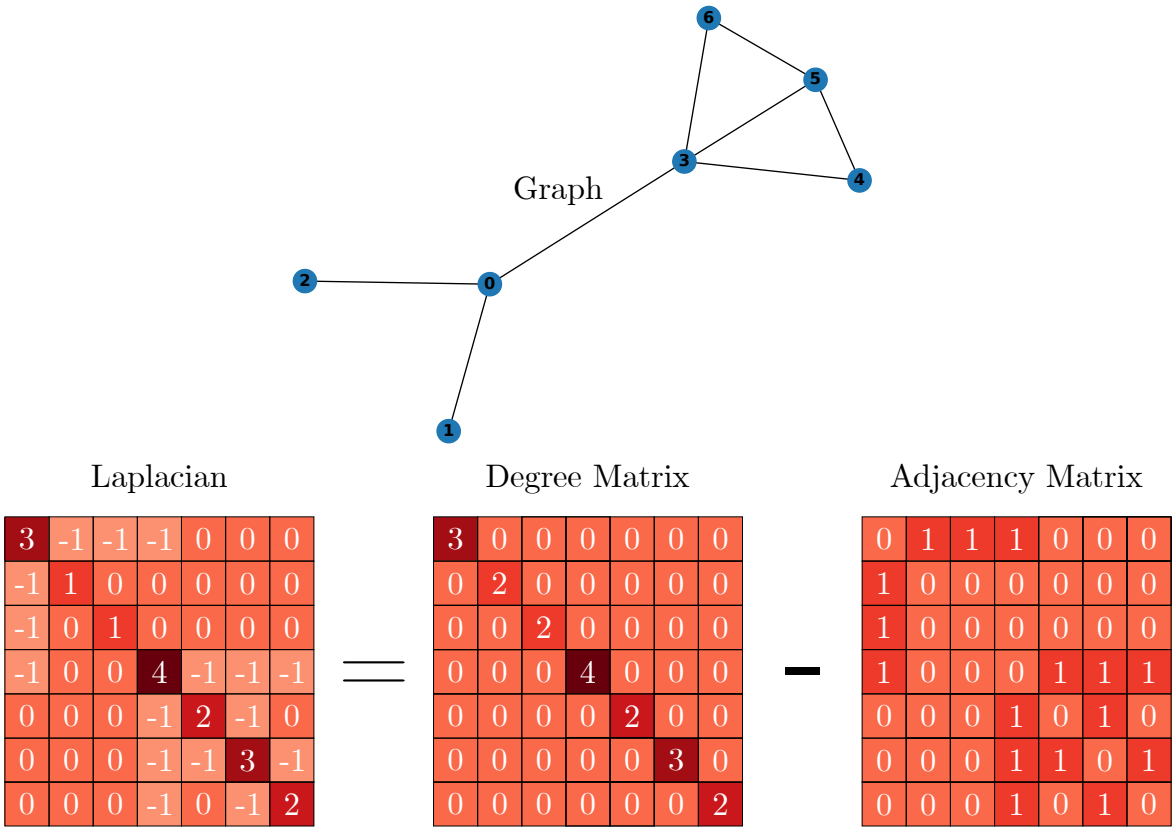


Figure 2.12: Calculation of the Laplacian matrix. The diagonal elements of the Degree matrix can be calculated by per row summation of the Adjacency matrix.

To perform convolution on the spectral domain we need to calculate the eigenvectors of the Laplacian and in practice it is enough to calculate only a few of them, specifically the ones that correspond to the smallest eigenvalues.

Now we can define mathematically the convolution in the spectral domain

$$X_{p+1} = V(V^T X_p \odot V^T W_p^{spectral}) \tag{2.10}$$

Where X_p is the node features before the convolution (assumed to be one dimensional), for the graph of [Fig 2.12] it would be of size $[7 \times 1]$. V is the eigenvector matrix of size $[7 \times 5]$ (we choose to use 5 eigenvectors), $W_p^{spectral}$ is the filter that we try to learn which is of size $[7 \times 5]$ (5 filters), and X_{p+1} is the updated node features of size $[7 \times 5]$. Finally, $\odot$ denotes element-wise multiplication. To extend the convolution to a case where the input feature has multiple dimensions, we repeat the same procedure for each dimension, and then sum the results over all dimensions, similar to how convolution is performed on multi-channel images.

Spectral methods have the disadvantage of having to decompose the Laplacian matrix which has a computational complexity of $O(n^3)$. Additionally, as we showed the size of the filters depends on the structure of the graph. These 2 facts result in increased computational difficulty of applying this method to large graphs. Several methods have been developed to overcome these problems like [Bruna et al., 2014; Defferrard et al., 2016] but this is outside of the scope of this work.

- **Spatial methods**: These methods do not require storing or decomposing a Laplacian matrix. In the context of this thesis we primarily focus on these methods. After seeing an extensive literature review about spatial methods (section [2.5.2]) we explain in more detail the "EdgeConv" method [Wang et al., 2019]. Formally the "EdgeConv" can be described as

$$\mathbf{x}_i^* = \underset{k=1:K}{\text{AGG}}[h_\Theta(\mathbf{x}_i, \mathbf{x}_j^k)] \tag{2.11}$$

where $\mathbf{x}_i$ refers to the node features of the central node in the input of the convolution and $\mathbf{x}_i^*$ refers to the node features of the central node in the output of the convolution. The central node has K neighbours with node features $\mathbf{x}_j^k$. AGG refers to a symmetric aggregation function as for instance **max** or **mean**, making this convolution invariant to permutations of the input. Finally h_Θ refers to a nonlinear function with parameters Θ.

We see an example of an "EdgeConv" in [Fig 2.13]. In (A) we have extracted a subgraph from a larger graph. The subgraph is composed of the central node, whose value we want to update, and its neighbours. The central node has node features $\mathbf{x}_i$ and its 5 neighbours have node features $[\mathbf{x}_j^1, \mathbf{x}_j^2, \mathbf{x}_j^3, \mathbf{x}_j^4, \mathbf{x}_j^5]$. In (B) we encode features from the nodes of the graph to the edges of the graph. The edge

features are defined as $\mathbf{e}_{i,j} = [\mathbf{x}_i, \mathbf{x}_i - \mathbf{x}_j]$. This choice leads to a convolution that is not translation invariant. If translation invariance is of interest, different edge features can be used such as $\mathbf{e}_{i,j} = [\mathbf{x}_i - \mathbf{x}_j]$. In (C) we pass the edge features through an MLP, that is a NN with 2 fully connected layers and a ReLU activation function. This implies that in this case we choose $h_\Theta = \text{MLP}([\mathbf{x}_i, \mathbf{x}_i - \mathbf{x}_j])$, where Θ are the weights and biases of the MLP that will be optimised. In (D) we see the updated edge features on the graph. Finally, in (E) we have used a **max** aggregation function to aggregate the updated edge features and update the node features of the central node.

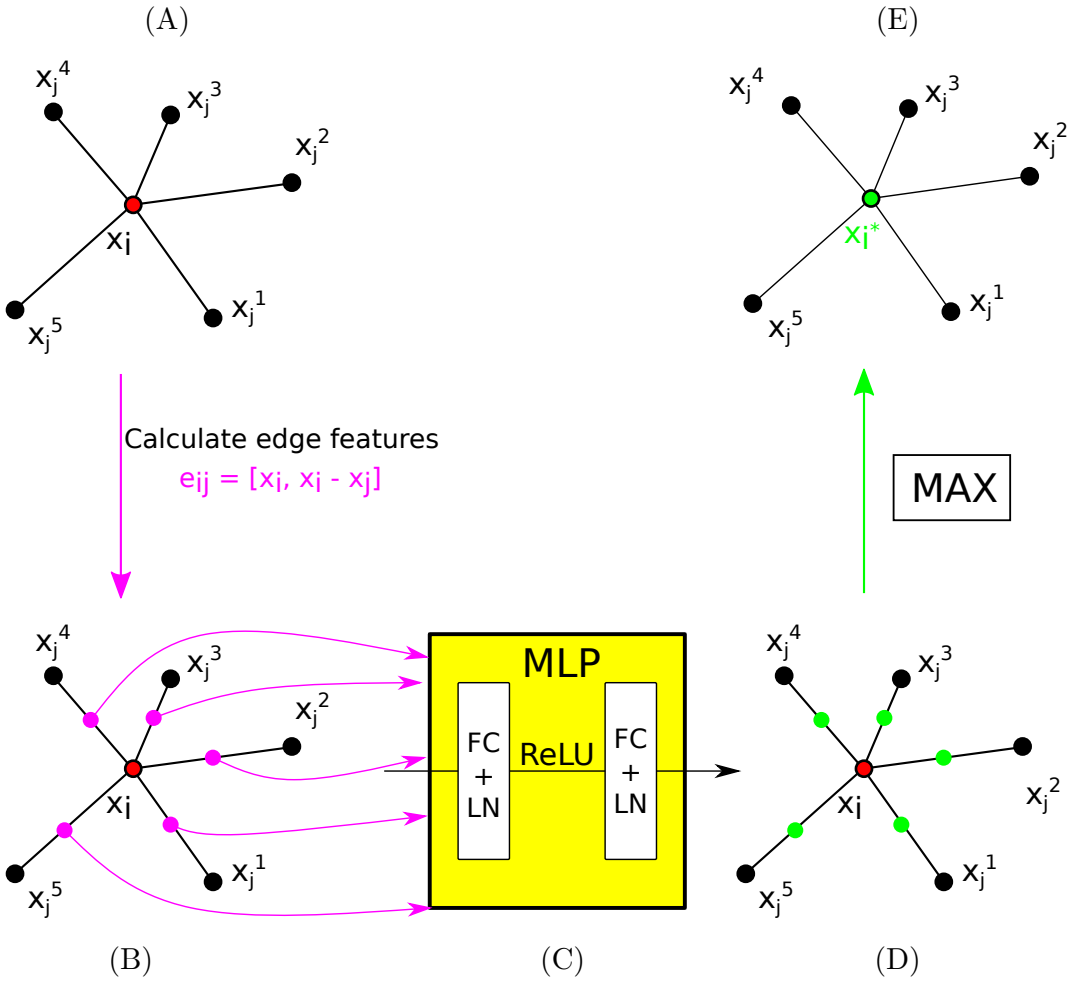


Figure 2.13: Example of an EdgeConv with AGG = **max**, and $h_\Theta = \text{MLP}([\mathbf{x}_i, \mathbf{x}_i - \mathbf{x}_j])$. The node features of the central node are denoted with $\mathbf{x}_i$ before the convolution and the updated ones with $\mathbf{x}_i^*$.

To sum up, we have seen two examples of graph convolutions. In the context of this

PhD thesis we focus only on the spatial graph convolutions. We explained in detail how to perform an "EdgeConv", which we believe is a simple introduction to the convolution we use, namely "GN Block" [Battaglia et al., 2018], for which we will give more details in [6.2.5].

2.6 Bayesian Neural Networks

In this section we demonstrate the importance of getting uncertainty estimation from the NN. We give a brief literature review of techniques commonly used to convert a deterministic NN to a Bayesian NN, that can quantify the uncertainty of its own prediction, and we discuss their advantages and disadvantages. Lastly, we give more details on the method we use in this thesis to extract uncertainty from the NN.

2.6.1 Introduction

Even though NNs are being used today in a variety of tasks, they usually lack a key property, namely quantifying the uncertainty of their predictions. The extrapolation ability of NNs is rather poor, thus making predictions on data outside the training domain can lead to dire consequences. For example in [Fig 2.14] we see a simple 2 layer fully connected NN that is forced to make a prediction outside of the training data range. We notice that it manages to fit the data very well but outside the training range the behaviour is very unpredictable. On the left, it predicts 0 for all the values, while on the right the prediction keeps decreasing almost linearly with a very steep slope. Consequently, we see that the NNs are rather unreliable in applications that they might face input data very different from the training data.

A well understood and widely used method that can be used to solve this problem is the Gaussian Process (GP). A GP is a stochastic regression model that outputs a mean prediction along with credible intervals. It requires a covariance (or kernel) function, which is a measure of similarity between 2 data points. A common choice is the Squared Exponential kernel

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2l^2}) \tag{2.12}$$

where $\mathbf{x}_i, \mathbf{x}_j$ are two data points and l is called length-scale which is a hyperparameter that defines how close 2 points have to be to interact with each other. Commonly 2

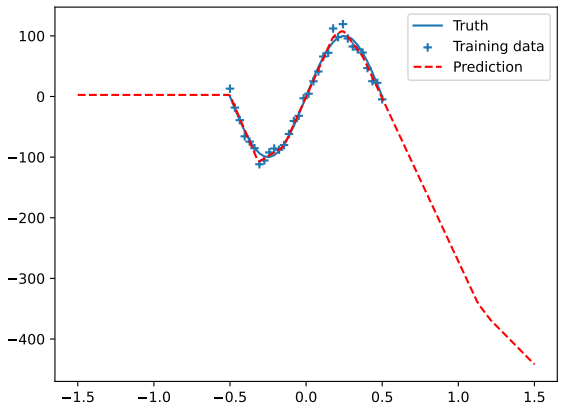


Figure 2.14: Prediction of a NN for values inside and outside the training range. The NN has 2 hidden fully connected layers, a ReLU activation function for all the layers apart from the last one where a linear activation function is applied and it is trained with the Adam optimiser.

more hyperparameters are added to the model, namely the variance, σ, and the noise ϵ. The variance controls how much the prediction is allowed to deviate from the mean and the noise represents the noise in the data. These 3 hyperparameters can be optimised by minimising the negative marginal log likelihood. An example of a GP can be seen in [Fig 2.15] for the same dataset as in [Fig 2.14]. We can observe that the GP not only fits the data, but it is also able to output very broad Credible Intervals (CIs) outside the training data range expressing the uncertainty of its prediction.

Unfortunately, the computational complexity of GPs is cubic with the number of data points which makes its application very expensive on large datasets. Additionally, GPs lose efficiency in high dimensional spaces and lastly, they are not designed to operate on image and graph data representations like CNNs and GNNs do. For these reasons, in this thesis we try to combine the advantages of NNs with the probabilistic nature of GPs and thus we use Bayesian NNs (BNNs) that are able to output a mean prediction and CIs.

2.6.2 Bayesian modelling

In order to get a probabilistic output from the NN we have to replace the constant parameters of the NN with distributions over the parameters [Fig 2.16]. Following a standard Bayesian logic we define a prior distribution of parameters, $p(\boldsymbol{\omega})$, that

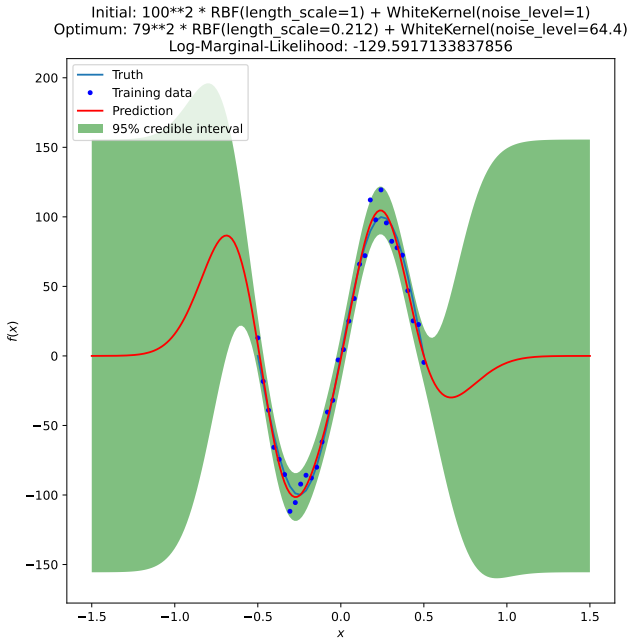


Figure 2.15: Prediction of a GP for values inside and outside the training range. On the title of the figure we can observe the initial and optimised values for the hyper parameters. Specifically the original values are $[100, 1, 1]$ and the optimised $[79, 0.212, 64.4]$ for the variance, length scale and noise level respectively.

reflects our prior beliefs for the parameters of the network. Given some data $\mathbf{D} = [(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_N, \mathbf{y}_N)]$ where we denote the input of the network with $\mathbf{X} = [\mathbf{x}_1, .., \mathbf{x}_N]$ and the output with $\mathbf{Y} = [\mathbf{y}_1, .., \mathbf{y}_N]$, we can update the prior and get the posterior distribution, that better fits the data. Based on the Bayes rule the posterior can be rewritten as

$$p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega})p(\boldsymbol{\omega})}{p(\mathbf{Y}|\mathbf{X})} \tag{2.13}$$

where the denominator, called model evidence, can be calculated by marginalising over $\boldsymbol{\omega}$

$$p(\mathbf{Y}|\mathbf{X}) = \int p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega})p(\boldsymbol{\omega})d\boldsymbol{\omega} \tag{2.14}$$

for linear models this can be calculated analytically but is intractable for NNs. Consequently, we cannot calculate the posterior this way but as we show in the next section we can approximate it by a variational distribution.

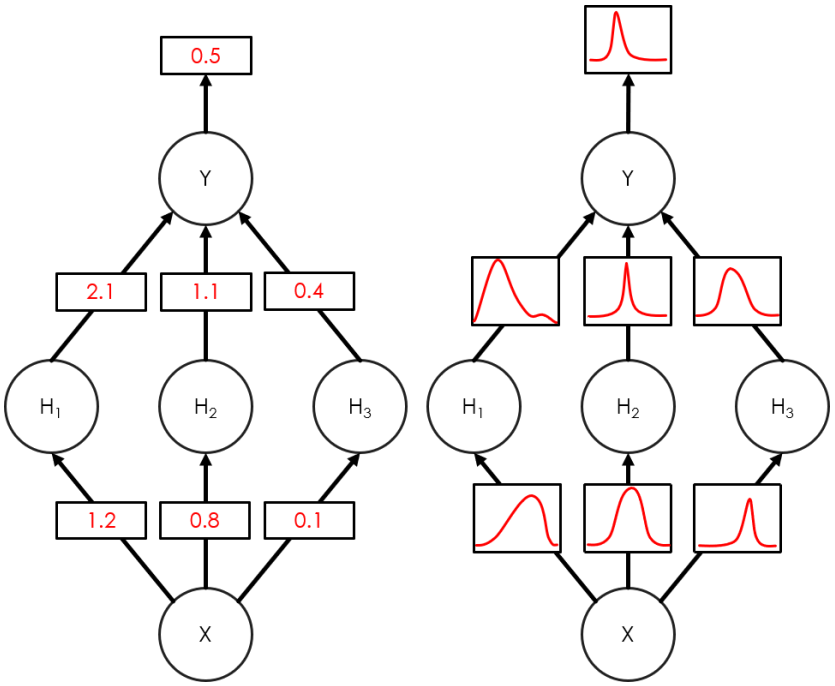


Figure 2.16: On the left we have a sketch of a plain Neural Network with constant weights and on the right a Bayesian Neural Network where the weights are replaced by distributions. In both sketches the biases have been omitted for simplicity.

2.6.3 Variational Inference

As seen in the previous section the posterior distribution of the parameters of the BNN cannot be calculated analytically. A common approach to circumvent this problem is to perform Variational Inference (VI), which consists in approximating the posterior density by a variational distribution $q(\boldsymbol{\omega}|\theta)$ [Hinton and van Camp, 1993; Graves, 2011] parameterised by a set of learnable parameters θ. The approximate distribution needs to be as close as possible to the real one. To this end, the loss function that needs to be minimised is the Kullback-Leibler (KL) divergence between the 2 that can be more practically written as

$$\begin{aligned}
\theta^{opt} &= \arg\min_{\theta} \mathrm{KL}[q(\boldsymbol{\omega}|\theta)||p(\boldsymbol{\omega}|\mathbf{D})] \\
&= \arg\min_{\theta} \int q(\boldsymbol{\omega}|\theta) \log \frac{q(\boldsymbol{\omega}|\theta)}{p(\boldsymbol{\omega})p(\mathbf{D}|\boldsymbol{\omega})}\, d\boldsymbol{\omega} \\
&= \arg\min_{\theta} [\mathrm{KL}[q(\boldsymbol{\omega}|\theta)||p(\boldsymbol{\omega})] - \mathbb{E}_{q(\boldsymbol{\omega}|\theta)}[\log p(\mathbf{D}|\boldsymbol{\omega})]]
\end{aligned} \tag{2.15}$$

This loss is known as the evidence lower bound (ELBO) and it consists of a data free and a data dependent term. The data free term is the KL divergence between the approximate posterior and the prior that acts as a regularisation term. The KL term may or may not be able to be calculated analytically depending on the choice of prior and posterior. The data dependent term is the negative log likelihood that encourages the BNN to fit the data.

Several techniques have been proposed to perform variational inference and quantify the uncertainty in BNNs [Hinton and van Camp, 1993; Graves, 2011; Kingma and Welling, 2014; Blundell et al., 2015]. These techniques are computationally expensive, for instance in [Blundell et al., 2015] choosing Gaussian distributions as approximate posterior distributions for the weights results in doubling the NN parameters without substantially increasing NN's capacity.

To tackle this problem Stochastic Regularisation Techniques (SRTs) are employed to approximate variational inference. Examples of SRTs include dropout [Hinton et al., 2012], dropConnect [Wan et al., 2013] and multiplicative Gaussian noise [Srivastava et al., 2014]. These techniques regularise NNs by injecting random noise. By far the most common SRT used for extracting uncertainty from a NN is Bernoulli dropout [Gal and Ghahramani, 2016], where the output of each layer is multiplied with a random variable ϵ sampled from a Bernoulli distribution $\epsilon \sim Bernoulli(1-p)$, where p is the zero-out probability.

Nevertheless, we opt for better uncertainty quality over reduced computational cost and thus we use the Bayes by Backprop method as described by [Blundell et al., 2015], instead of an SRT. In the next section we provide more details about the method.

2.6.4 Bayesian linear layer

In this section we explain how a Bayesian linear layer works. In reality the procedure is very similar to a simple linear layer but the weights and biases instead of being constant are being sampled from the (approximate) posterior distribution. We consider the approximate posterior to be a fully factorised Gaussian [Graves, 2011]. The prior is also a Gaussian distribution corresponding to L2 regularization [Blundell et al., 2015]. All weights within one layer share the same prior mean, μ, and the same prior standard deviation, σ.

In a forward pass of the BNN the weights are sampled from the posterior distribution, and gradients are evaluated using a generalisation of the Gaussian reparametri-

sation trick [Kingma and Welling, 2014], as described in [Blundell et al., 2015]. The sampling from the variational posterior distribution is performed by sampling from a unit Gaussian scaled by the posterior standard deviation, σ^*, and shifted by the posterior mean, μ^*. To ensure its positivity, the standard deviation is parameterised as $\sigma^* = \log(1 + \exp(\rho^*))$. Consequently, the variational parameters to be optimised are $\theta = (\hat{\mu}, \hat{\rho})$, where $\hat{\mu} = (\mu, \mu^*)$ and $\hat{\rho} = (\rho, \rho^*)$.

2.6.5 Example of a Bayesian NN

In this section we will demonstrate the use of a BNN in the dataset introduced in [Fig 2.14]. Specifically, we convert the deterministic NN used to obtain the aforementioned figure to a BNN. To this end we replace the standard linear layers of the NN with Bayesian ones (as explained in section [2.6.4]). Lastly, we need to replace the "MSE" loss function with the "ELBO" loss function, introduced in [Eq. 2.15]. No change needs to be made to the optimiser compared to the deterministic case. The results can be found in [Fig 2.17] where we can observe that the BNN was able both to fit the data and also to provide broad CIs far from the training dataset.

2.6.6 Extension to CNNs and GNNs

So far we have only referred to Bayesian *linear* layers. Nevertheless, the same logic can be directly applied to convolutional layers where instead of weights we have filters. As for the spatial GNNs, we have already seen in [2.5] that they are basically composed of MLPs. Replacing the linear layers in the MLPs with Bayesian linear layers results in a Bayesian GNN.

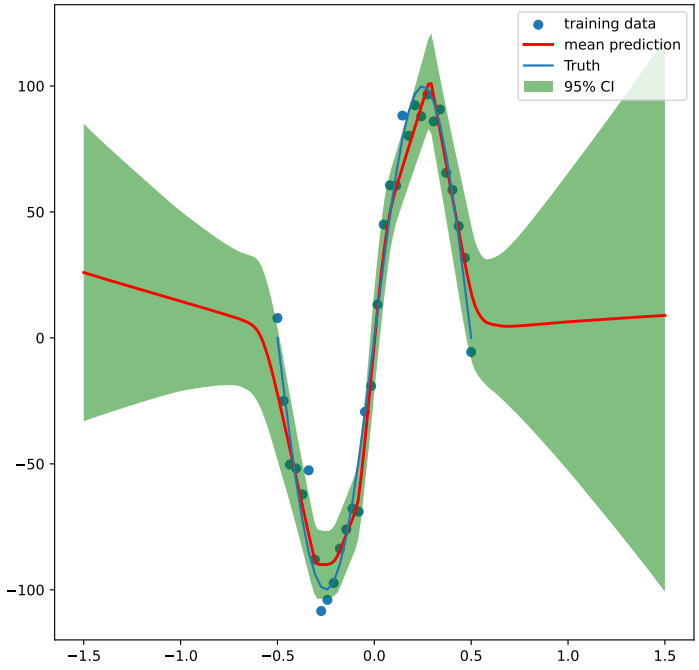


Figure 2.17: Prediction of a BNN for values inside and outside the training range. The mean prediction fits the data while the CIs are broad outside the training data range.

2.7 Optimiser

A common point in all the sections above, regardless of the type of the NN or if it is Bayesian or deterministic, is the optimiser. With the term optimiser we refer to the algorithm that we use to solve the minimisation problem that we formulate so that the output of the network matches the available data. The optimiser iteratively changes the value of the network parameters, θ, in order to minimise the value of the loss function. Every iteration of the optimiser is called "epoch" in the NN terminology.

All the popular frameworks for ML offer efficient and robust implementations of all the major optimisers. Consequently, in practice although ML practitioners spend a lot of time deciding on the structure of the network (type of architecture, number of layers, number of neurons, etc.) they do not focus their energy on the optimiser where typically a few experiments are enough to choose the algorithm and the value of the hyperparameters. Nevertheless, we believe that it is important in the context of this thesis to explain the basic ideas behind the most common optimisers starting from the most basic optimisers such as Gradient Descent to more sophisticated examples such as the Adam optimiser. A non-exhaustive list can be found below.

- **Gradient Descent**: This is the most basic first-order optimization algorithm. It is easy to implement but it may get trapped to local minima, it requires a lot of memory and it updates the parameters only after calculating the gradient on the whole dataset, which is inefficient for large datasets.

$$\theta_{t+1} = \theta_t - a \cdot \nabla L(\theta_t) \tag{2.16}$$

  where θ_{t+1} is the parameters for epoch $t+1$, θ^t is the parameters for epoch t and a is called learning rate. The learning rate is a hyperparameter that determines the step size at each epoch. At every epoch, the parameters are updated towards the direction dictated by the gradient and with a step size given by the learning rate.

- **Stochastic Gradient Descent (SGD)**: This algorithm was developed to tackle the infrequent parameter updates of the classical gradient descend algorithm. Specifically it updates the model parameters after calculating the gradients for each training example, $(\mathbf{x}^i, \mathbf{y}^i)$. This results in lower memory requirements, faster convergence and improved ability to find the global minimum. Nevertheless, the

frequent parameter update results in increased variance in the model parameters and requires reduction of the learning rate with the epochs otherwise it results in overshooting even if it has located the global minimum.

$$\theta_{t+1} = \theta_t - a \cdot \nabla L(\theta_t; \mathbf{x}^i; \mathbf{y}^j) \tag{2.17}$$

In practice SGD is employed in a mini-batch optimisation procedure. This implies that the dataset is divided in batches of size n, $(\mathbf{x}^{i:i+n}, \mathbf{y}^{i:i+n})$. The parameter update in this case happens after calculating the gradients for each batch. This results in higher memory requirements but reduced parameter variance.

$$\theta_{t+1} = \theta_t - a \cdot \nabla L(\theta_t; \mathbf{x}^{i:i+n}; \mathbf{y}^{i:i+n}) \tag{2.18}$$

From now on we will omit the parameters $\mathbf{x}^{i:i+n}$ and $\mathbf{y}^{i:i+n}$ in the calculation of gradients for simplicity.

Another trick that can be used to help the SGD to converge faster is momentum. With the cost of having one more hyperparameter, γ, momentum reduces the fluctuation to irrelevant directions. This is achieved by adding a fraction of the previous update vector to the current update vector. This favours paths whose gradients point to the same direction.

$$\theta_{t+1} = \theta_t - m_t \tag{2.19a}$$

$$m_t = \gamma m_{t-1} + a \cdot \nabla L(\theta_{t-1}) \tag{2.19b}$$

- **Adaptive Gradient Algorithm (AdaGrad)**: This optimiser aims at removing a key drawback of the aforementioned methods, namely the need to manually choose the learning rate. AdaGrad not only automatically determines the learning rate but it does so for each parameter and each epoch separately. Apart from being easier to use, AdaGrad is also more suited for dealing with sparse data since it allows parameters associated with infrequently occurring features to have larger learning rates compared to the ones associated with more frequently occurring features.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{a_i}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i} \tag{2.20a}$$

$$G_{t,i} = \sum_{\tau=1}^{t} g_{\tau,i} g_{\tau,i}^{\mathrm{T}} \tag{2.20b}$$

$$g_{t,i} = \nabla L(\theta_{t,i}) \tag{2.20c}$$

where G_t is a diagonal matrix where every diagonal element, $G_{t,ii}$, is equal to the sum of the squares of the gradients with respect to θ_i up to the given time step t. $g_{t,i}$ is the derivative of the loss function for the parameter θ_i, at a given time t. ϵ is a small term added for numerical stability, usually close to 1e-8. For simplicity, and since the update of all the parameters can be vectorised, we will drop the second subscript and refer to all the parameters together as θ_t, although each parameter can be updated separately with its own learning rate.

- **Root Mean Squared Propagation (RMSProp)**: RMSProp is one of the most commonly used optimisers and it can be seen as an extension of AdaGrad. AdaGrad has the disadvantage of excessively reducing the learning rate resulting in slower training times. This stems from the fact that AdaGrad scales the learning rate using the entire history of gradients. RMSProp, introduces a more efficient strategy where instead of the entire history an exponentially decaying average of the partial derivatives for the current epoch is used, u_t. This allows the optimiser to take decisions based on the most current shape of the search space. This comes with the cost of introducing a new hyperparameter, λ, usually set to 0.9.

$$\theta_{t+1} = \theta_t - \frac{a}{\sqrt{u_t + \epsilon}} \cdot g_t \tag{2.21a}$$

$$u_t = \lambda u_{t-1} + (1 - \lambda) g_t^2 \tag{2.21b}$$

- **Adaptive Moment Estimation (Adam)**: Adam, which is the most popular optimiser, is a combination of RMSProp and AdaGrad. It works by storing an exponentially decaying average of both past gradients, $\tilde{m}$, and squared gradients, $\tilde{u}$.

$$\theta_{t+1} = \theta_t - \frac{a}{\sqrt{\tilde{u}_t + \epsilon}} \cdot \tilde{m}_t \tag{2.22a}$$

$$\tilde{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{2.22b}$$

$$\tilde{u}_t = \frac{u_t}{1 - \beta_2^t} \tag{2.22c}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.22d}$$

$$u_t = \beta_2 u_{t-1} + (1 - \beta_2) g_t^2 \tag{2.22e}$$

All the optimisers have a common characteristic, which is the calculation of partial derivatives of the loss with respect to the model parameters. This in practice gives us the sensitivity of the loss function with respect to the network parameters. The partial derivatives are calculated using the chain rule. This calculation is called backpropagation in the NN terminology.

2.8 Conclusions of the chapter

In summary, this chapter covered the fundamentals of the various types of Neural Networks (NNs) utilised in this thesis, namely fully connected, convolutional, graph and Bayesian NNs along with main concepts common to all of them such as the activation function, loss function and optimiser.

Chapter 3

Finite Element surrogate models

3.1 Introduction

In the first two sections of this chapter we discuss about different surrogate modelling techniques to reproduce the solution of Finite Element solvers. We refer to both classical surrogate models and state-of-the-art Neural Network (NN) based models. We focus on the latter and we discuss different types of NNs, their architectures, building blocks and application domains. In the last two sections we review two common strategies to tackle multiscale problems, namely homogenisation and concurrent multiscale modelling, and we discuss works from the literature that utilise NN assisted methods for the solution of multiscale problems.

3.2 Classical surrogate models

Two common classical surrogate modelling techniques are polynomial chaos expansion (PCE) and Gaussian Processes (GPs) [Ghanem and Spanos, 1991; Xiao et al., 2009]. We have already discussed GPs in detail in section [2.6.1]. PCE is a method used to approximate the output, $\mathbf{Y}$, of a function, which is usually expensive to evaluate, using a polynomial function of random input variables, $\mathbf{X}$. These polynomials are constructed to be orthogonal to the distribution of the input random variables. Mathematically, PCE can be expressed as

$$\mathbf{Y} = \sum_{i \in \mathbb{N}} c_i \Psi_i(\mathbf{X}) \tag{3.1}$$

where c_i is a coefficient and Ψ_i a polynomial basis function.

Both these techniques are limited to small parameter dimensions, typically 1 to 5, and thus are not compatible with input images or graphs representing structures with random distributions of pores. Consequently, in this thesis we investigate NN based surrogate models that can efficiently work with arbitrarily images and graphs.

3.3 PDEs and Machine Learning

Machine Learning techniques have been investigated widely in the last years as FE surrogates that can reproduce the solution of PDEs. A variety of different NNs can be found in literature for that purpose. Below we give a literature review and discuss the state-of-the-art in 4 types of NNs, namely Fully Connected Networks, CNNs, GNNs, and Physics Informed Neural Networks (PINNs).

3.3.1 NNs with fully connected layers

In the context of this PhD thesis we do not focus on NNs with fully connected layers. Nevertheless, we give two examples of works using such networks in engineering applications. An early work that uses simple fully connected layers to replace expensive patient-specific FEA models is described in [Liang et al., 2018]. The methodology can be dissected in three distinct parts, namely shape encoding, nonlinear mapping and stress decoding. In the first step, shape encoding, Principal Component Analysis (PCA) is used to encode the geometry into a latent dimension, where it can be represented by a small number of parameters. We note that in the cases we examine in this thesis PCA typically does not yield in a latent space of small dimensions, due to the existence of random distributions of microscale geometrical features. In the second step, nonlinear mapping, a NN is used to map the encoded geometry to the target stress distribution encoded in a low dimensional space. Lastly, in the stress decoding stage, the stress is mapped from the low dimensional space to the output space. Other NNs with fully connected layers have then been used for deformation predictions in beam structures for non-linear, time dependent problems as shown by [Meister et al., 2018]. In this work a NN was used to predict vertex-wise accelerations for a large time step based on the current state of the system.

3.3.2 CNNs

Various CNN based surrogate models can be found in the literature, which are better suited to deal with image data. Firstly, [Nie et al., 2019] deployed a CNN model for stress prediction on cantilevered structures. They used an Encoder Decoder architecture. The encoder gradually transforms the input space into a low-dimensional representation, then a number of residual blocks are applied and lastly the decoder gradually upscales the data to the original resolution. An example of an Encoder Decoder CNN can be found in [Fig 3.1]. Encoder Decoder networks are very popular, their success is in a great degree attributed to the so called "bottleneck" layer where the network is forced to summarise all the information in a small number of variables that should be informative enough for the network to be able to reconstruct the output from. The authors use residual blocks in the bottleneck layer. A general structure for a Residual Block can be found in [Fig 3.2]. As we can see, even if the NN chooses to ignore some layers ($F(X) = 0$) it will learn to map the input of the block to the output of the block. In this case the expression of the output would be simplified to: $F(X) + X = 0 + X = X$. This way we can use a large number of residual blocks and the network will simply ignore the ones it does not need. The name residual comes from the fact that the network tries to learn the residual, $F(X)$, or in other words the difference between the true output, $F(X) + X$, and the input, X. Lastly, the authors make use of another novel layer, namely the Batch Normalisation (BN) layer. In deep neural networks the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This phenomenon is known as internal covariate shift. This slows down the training by requiring lower learning rates and careful parameter initialisation [Ioffe and Szegedy, 2015]. BN aims at reaching a stable distribution of activation values throughout training. To achieve that, BN normalises the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. After the normalisation, BN tries to scale and shift the normalised output by adding two trainable parameters to each layer. [Ioffe and Szegedy, 2015; Santurkar et al., 2019]. Additionally, BN makes the optimization landscape significantly smoother. This smoothness induces a more predictive and stable behaviour of the gradients, allowing for faster training [Santurkar et al., 2019].

Additionally, [Sun et al., 2020] extended the work of [Nie et al., 2019] to non-linear elastic FE problems. They created an Encoder-Decoder CNN for the prediction of stress fields on Fibre-reinforced Polymers. They predict the z component of the stress

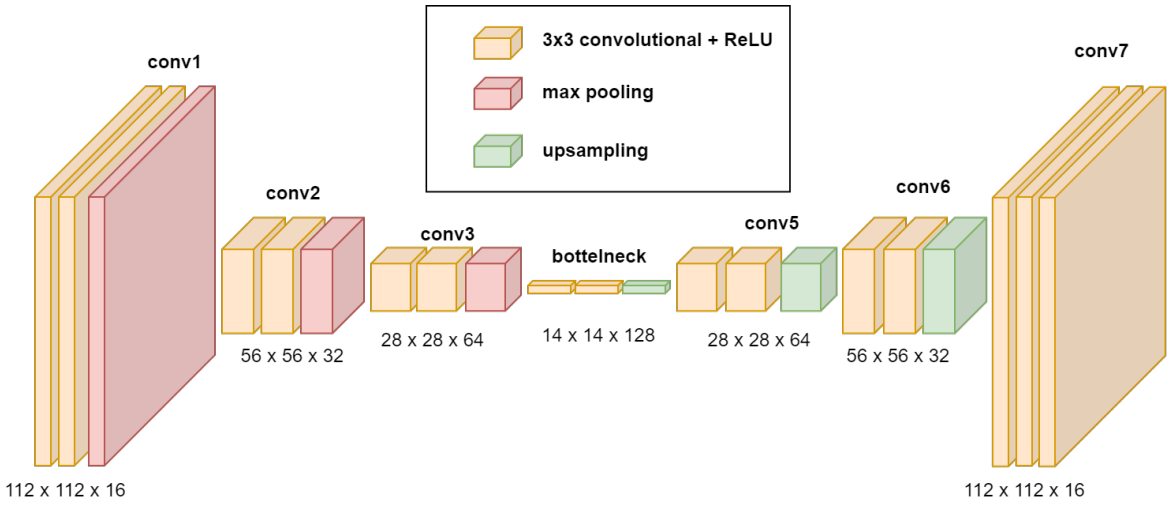


Figure 3.1: Sketch of an Encoder-Decoder CNN. Diagram created using "draw.io".

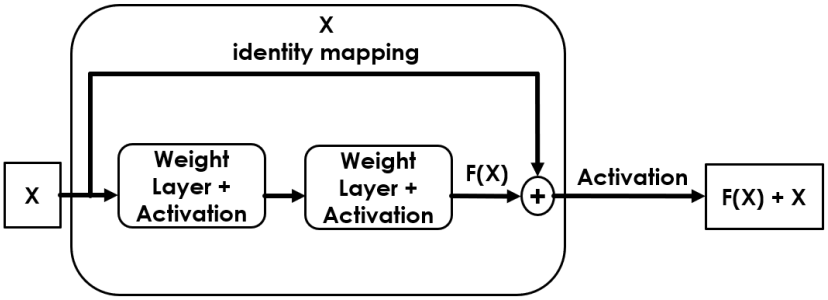


Figure 3.2: Structure of a generic residual block with input X and output $F(X) + X$.

tensor and they report a value of about 70% for the coefficient of determination in the test set.

Later, [Mendizabal et al., 2020a] used a U-Net for the prediction of soft tissue deformations. The U-Net is a network initially built for precise medical image segmentation [Ronneberger et al., 2015]. U-Net has an encoder decoder architecture but the notable advantage is the existence of skip connections that directly transfer information from the encoder to the decoder. An example of a U-Net can be found in [Fig 3.3]. The authors demonstrated their method on models of livers where the input of the network was the forces on the livers and the output was the nonlinear soft tissue deformation. They underlined the great potential of similar methods for real time biomechanics simulations.

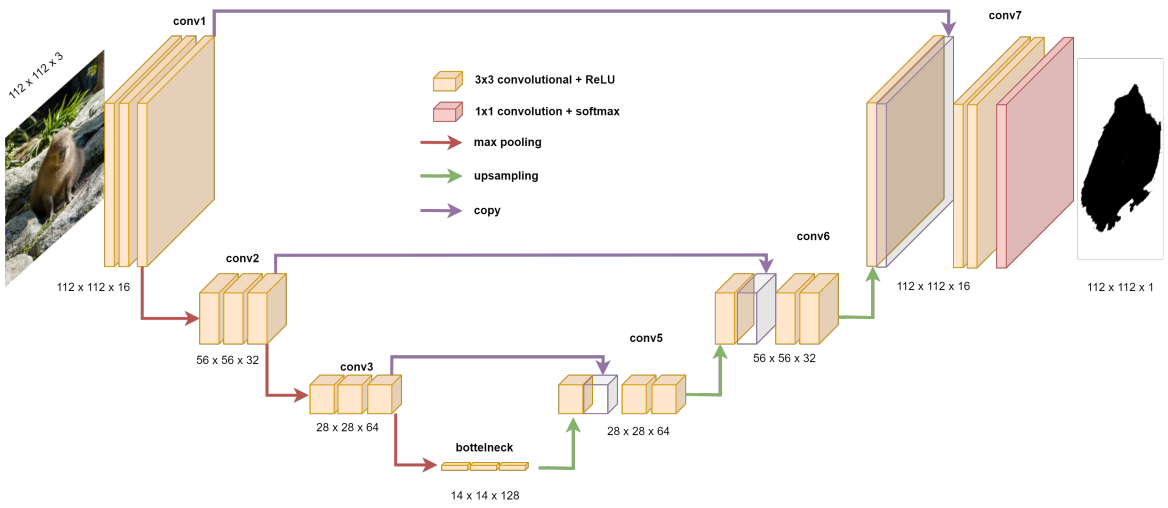


Figure 3.3: Example of a U-Net used for image segmentation

Moreover, [Jiang et al., 2021] used a conditional Generative Adversarial Network (GAN) for predicting the Von Mises stress on 2D structures. GANs, are generative models composed of a generator and a discriminator that are trained together. The generator generates samples that are then mixed with real examples from the training dataset and the discriminator tries to classify them as real or fake. During training, the two parts are competing against each other, which results in a generator that creates increasingly realistic samples, to the point that they are indistinguishable from the samples found in the training set. An example of a GAN can be found in [Fig 3.4]. The authors, compare this approach with their previous work, [Nie et al., 2019], and find that the GAN model outperforms the simple CNN model.

Lastly, [Wang et al., 2021] used a Convolutional Aided Bidirectional Long Short-term Memory Network to predict the sequence of maximum internal stress until material failure. Long Short-term Memory Networks (LSTMs) are a type of recurrent NN suitable for processing instead of single data entire sequences of data. Common applications involve handwriting recognition and speech recognition.

3.3.3 GNNs

In this section we briefly review geometric learning methods reported in literature including the one we use in this work.

GNNs have shown very promising results in solving computational engineering problems. An early example can be found in the work of [Guo and Buehler, 2020], where

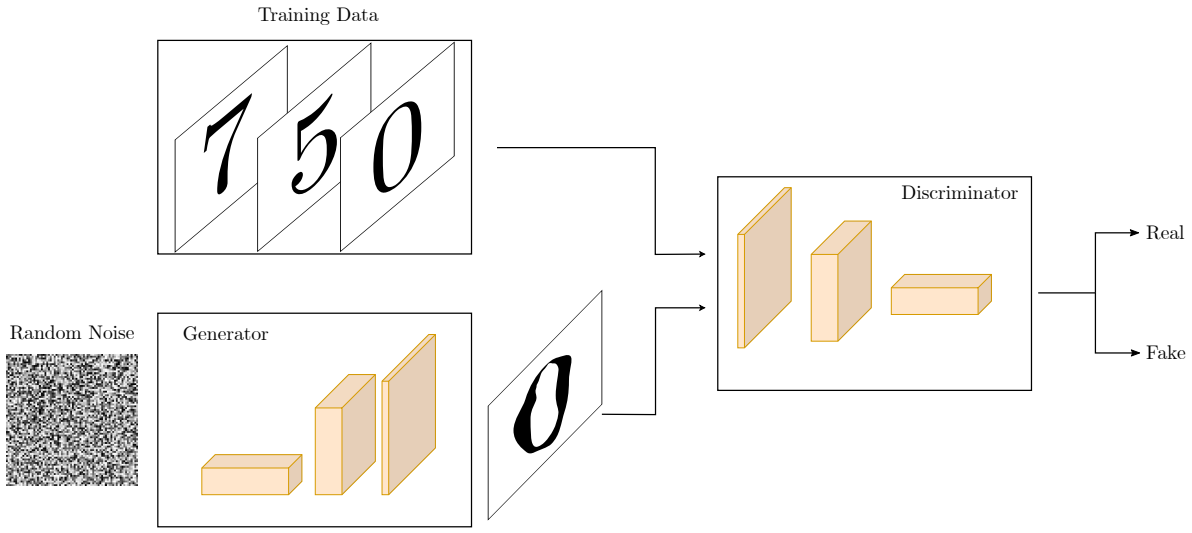


Figure 3.4: Example of a GAN for digit generation

the authors developed a semi-supervised approach to design architected materials using a binary classification GNN. The input to the GNN is the load levels for 1% of the nodes and the output is the load levels for the rest of the nodes. The load level is a binary label, where the two possible values are the low stress and high stress area. The framework requires experimental data at test time (the load levels for 1% of the nodes). The authors mention that such data can be obtained from embedded sparse tensors.

In the same year, [Vlassis et al., 2020] used a GNN for modelling anisotropic hyperelastic materials, where polycrystals are modelled using a graph of monocrystals. The nodes represent the monocrystals and the edges their connectivity. This allowed the authors to fully take advantage of all the microstructure information instead of being dependent on classical hand-crafted descriptors like density and porosity. The GNN uses the graph convolution layers introduced in [Kipf and Welling, 2017], that take as input a symmetric normalised graph Laplacian matrix, computed through the graph connectivity, and the node features.

$$\mathbf{z}_{p+1} = \sigma_{p+1}(\mathbf{L}^{\mathbf{sym}}\mathbf{z}_p\mathbf{W}_{p+1} + \mathbf{b}_{p+1}) \tag{3.2a}$$

$$\mathbf{z}_1 = \sigma_1(\mathbf{L}^{\mathbf{sym}}\mathbf{X}\mathbf{W}_1 + \mathbf{b}_1) \tag{3.2b}$$

where $\mathbf{z}_p$ is the activation of the layer p, $\mathbf{L}^{\mathbf{sym}}$ is the symmetric normalised graph Laplacian, $\mathbf{X}$ is a matrix created by stacking together all the node features, σ is an

element-wise non-linearity, and finally $\mathbf{W}_p$ and $\mathbf{b}_p$ are the weights and biases of the layer p.

Another type of engineering application where GNNs can naturally be applied to, is particle-based dynamic systems. A great example is the work by [Sanchez-Gonzalez et al., 2020], where GNNs are used to simulate systems of interacting particles, with one or more types of particles as for instance water and sand. The graph of the system is constructed by considering each particle as a node and by building connections (edges) with their closest neighbours. The physics is predicted by message passing through the nodes (particles) of this graph. The GNN has an encoder - processor - decoder structure. The encoder embeds the physical particle-based state into a latent graph, the processor is used to pass messages between the nodes of the graph and finally the decoder extracts dynamics information from the nodes of the latent graph.

Later, [Pfaff et al., 2021] used GNNs for mesh-based dynamic simulations. In contrast to particle-based problems, mesh-based problems allowed the authors to use the connectivity dictated by the mesh to define the edges of the graph. They used a similar architecture as in [Sanchez-Gonzalez et al., 2020]. They provided a variety of examples ranging from deformable bodies to fluid and air flow. Most importantly they make a comparison between a GNN and a CNN architecture where they conclude that the GNN had superior performance since it was able to make good predictions both on the large and fine scale, in contrast to the CNN whose prediction was only good in the large scale.

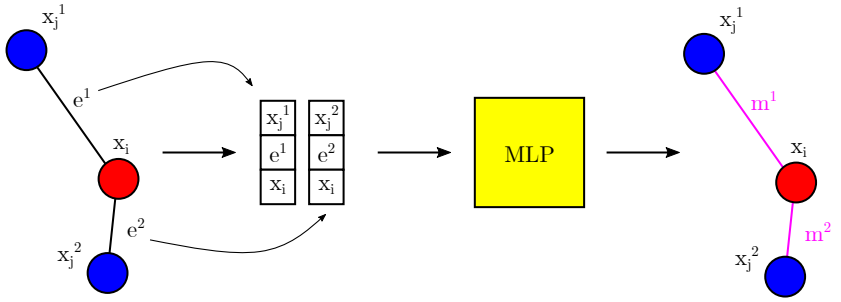
Additionally, [Mylonas et al., 2022] used a Bayesian GNN to infer the position and shape of an unknown crack via patterns of dynamic strain field measurements at discrete locations. A graph is constructed with nodes corresponding to the positions of sensors used to measure the strain tensors. The connectivity of the graph is constructed by creating edge connections between nodes that are closer to each other. Features are encoded both on the nodes and edges of the graph. Strain tensors are encoded on the nodes and relative positions on the edges. The node and edge features are updated through a number of GN blocks [Battaglia et al., 2018] and finally aggregated through a mean function to produce global features that correspond to the position and orientation of the crack. The uncertainty of the prediction is calculated through Variational Inference using the local reparametrisation trick [Kingma et al., 2015].

Moreover, [Lino et al., 2021] developed a multiscale GNN that efficiently diffuses information across different scales making it ideal for tackling strongly nonlocal problems such as advection and incompressible fluid dynamics.

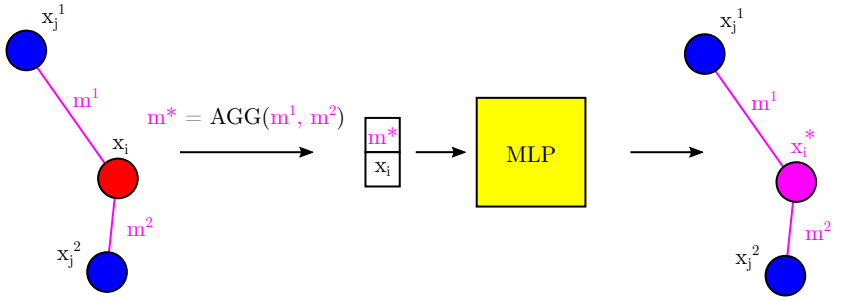
Furthermore, in [Perera et al., 2022] the authors develop a GNN based framework to simulate fracture and stress evolution in brittle materials due to multiple microcracks' interaction. The GNN predicts the future crack-tip positions and coalescence, crack-tip stress intensity factors, and the stress distribution throughout the domain at each future time-step.

Lastly, [Deshpande et al., 2022a] introduced MAgNET, a U-Net based GNN with mesh pooling and unpooling operations, in accordance with the U-Net architecture for CNNs, that efficiently scales with the size of the problem. MAgNET is used to predict nonlinear force-displacement mappings and the authors provided examples of applying MAgNET in real-world geometries such as those arising in biomechanics. In their latest work, [Deshpande et al., 2023], the authors compared MAgNET with a novel attention-based architecture called Perceiver IO, proposed by [Jaegle et al., 2022]. Perceiver IO was developed with the goal to easily integrate and transform arbitrary information for arbitrary tasks, so the authors use it without adding information about the underlying data structure (such as the mesh connectivity). In terms of training, Perceiver IO requires less parameters but much more training time. Nonetheless, it is much faster in the inference stage. In terms of results, for inputs of small size, the Perceiver IO slightly outperforms MAgNET, but as the size and complexity of the mesh increases it becomes less robust and fails to learn efficiently, which is attributed to the fact that the mesh connectivity is not provided so it has to implicitly learn the nodal data dependencies.

A lot of the aforementioned GNNs, and the ones we use in this PhD thesis, use one of the graph convolutions introduced by [Battaglia et al., 2018], that we will refer to from now on as "GN Block". This formulation allows encoding information both on the nodes and the edges of the mesh. Node information is treated as absolute information, for instance material properties, while edge information is treated as relative information, for instance relative position between the nodes. The edge and node features are updated using MLPs. To update the edge features of an edge connecting two nodes, both the edge and node information are taken into account. Specifically, the edge features of the edge and the node features of the two aforementioned nodes are concatenated and passed through an MLP to get the updated edge features. The updated edge features are referred to as "messages" in the GNN terminology. To update the node features, firstly, the messages (updated edge features) are aggregated using a symmetric aggregation function. Then, the aggregated messages are concatenated with the central node features. Finally, the updated node features are calculated by passing this concatenation through an MLP [Fig 3.5b].



(a) Edge update



(b) Node update

Figure 3.5: This figure summarises the procedure followed by a GN block to update the node and edge features of a graph. On the top (a), we see a sketch of the edge update step of the GN block. x_i are the node features of the central node, (x_j^1, x_j^2) are the node features of the neighbouring nodes, (e^1, e^2) are the edge features of the edges that are connected to the central node and $(\mathrm{m}^1, \mathrm{m}^2)$ are the calculated messages, that can also be interpreted as the updated edge features. On the bottom (b), we see a sketch of the node update step of the GN block. AGG denotes a symmetric aggregation function, m^* are the aggregated messages and finally x_i^* are the updated node features of the central node.

3.3.4 PINNs

Another branch of Machine Learning that is very promising for engineering applications is that of Physics Informed Neural Networks (PINNs) [Raissi et al., 2019]. PINNs are able to incorporate the physics laws in their training process, greatly reducing the amount of data needed and simultaneously increasing the generalisation of the network. We note two interesting applications of PINNs.

Firstly, PINNs are able to learn from incomplete data, for instance in the work by [Raissi et al., 2020], the authors manage to extract velocity and pressure fields by only using concentration as labelled data, for 3D physiologic blood flow in a patient-specific intracranial aneurysm (ICA). That is of great significance, for instance in biomechanics simulations, because by measuring the value of a passive scalar, for instance dye or

smoke, which is easy to measure, one can infer more interesting quantities that are much harder to measure. Additionally, performing standard FE simulations in this case is rather cumbersome since the geometry and the Boundary Conditions (BCs) are not easy to define. PINNs can overcome this problem since this information is indirectly encoded in the passive scalar data. The aforementioned procedure for a 2D case can be found in [Fig 3.6]. We can see that the loss function has two terms, one forces the predicted concentration to match the measured concentration and the second term forces the network to respect the constitutive law.

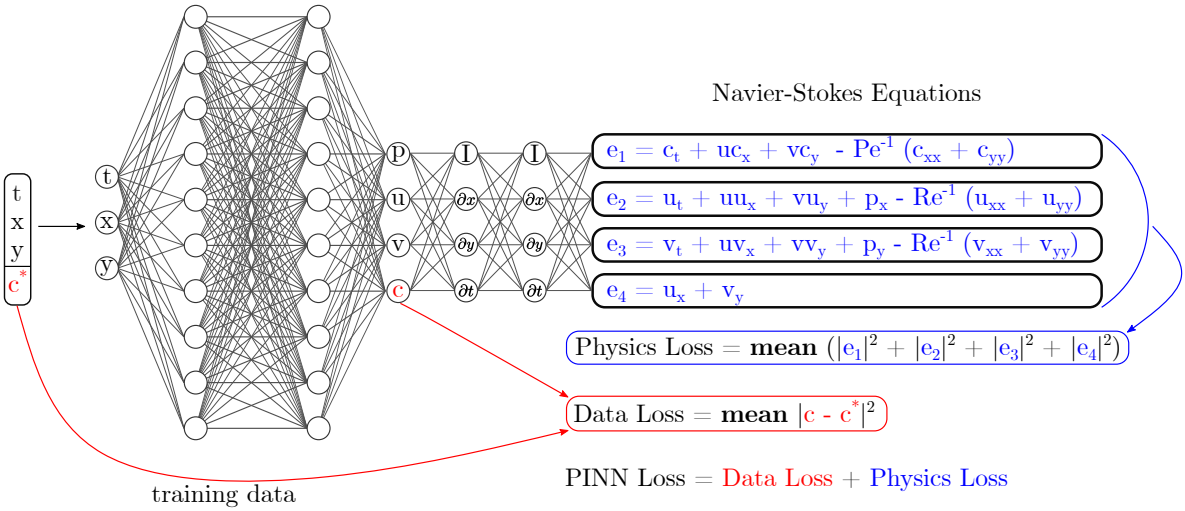


Figure 3.6: Sketch of a PINN. The input of the PINN is points in space-time where the validity of the constitutive law will be evaluated, (x, y, t), along with measured data of concentration, c^*. The constitutive law can be evaluated and thus enforced in any point in the computational domain.

Secondly, PINNs can be very efficient in design space exploration problems. That has captured the attention of the industry, for instance NVIDIA created SIMNET™[Hennigh et al., 2021] which uses a variant of PINNs that uses no data at all to perform multi-physics simulation. This is easily derived from the loss shown in [Fig 3.6] if we set the "Data Loss" term to zero. This approach can be seen as NN based solver, but in contrast to traditional solvers it can be trained with multiple design parameters in a single run. For the inverse problem of optimising the geometry of a heat sink parameterised by 12 parameters, SimNet accelerates the design by a factor of 45,000 compared to a commercial solver and by a factor of 135,000 compared to OpenFOAM.

3.4 Multiscale methods

Multiscale structural analyses are prominent in mechanical and bio-mechanical engineering (e.g., composite materials such as carbon-reinforced polymers or concrete, porous materials such as bones). Full Finite Element Analysis (FEA) for stress prediction is usually prohibitively expensive for those structures, as the finite element mesh needs to be very dense to capture the effect of the fine scale features. Therefore, a common approach is to split the problem into a macroscale mechanical problem, and local microscale computations. The macroscale problem diffuses the overall stress field in the entire structure without fully resolving the material, while the local microscale computations are needed to correct the macroscale fields and characterise the constitutive law to be used at the macroscale. Multiscale computational modelling can be approached in two ways, namely homogenisation and concurrent multiscale modelling. In this section we briefly discuss these 2 strategies.

3.4.1 Homogenisation

In homogenisation [Évariste Sanchez-Palencia, 1987; Zohdi and Wriggers, 2005], the aim is to find a homogeneous material that has an equivalent behaviour to the original heterogeneous material. To this end, the purpose is to find homogeneous governing equations in the macroscale by knowing the governing equations in the microscale. In homogenisation we perform all microscale computations over an RVE. The RVE needs to be large enough compared to the size of the heterogeneities so that different RVEs have very similar microscale statistics but at the same time it should be small enough so that macroscale displacement gradients do not vary over the material sample [Fig 3.7].

Let us see an example where we consider a linear elastic heterogeneous material [Fig 3.8]. We consider a body force $\boldsymbol{f}$ and a macroscopic displacement $\mathbf{u}^{\mathrm{M}}$, thus we have the following macroscopic equations.

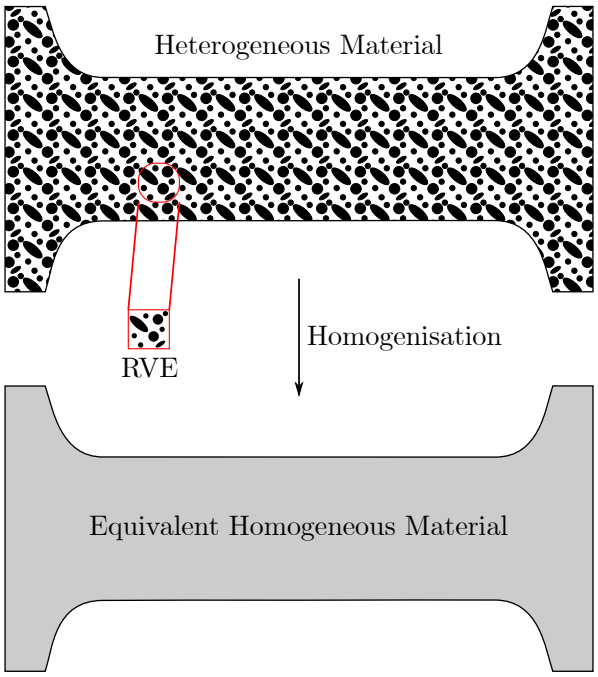


Figure 3.7: The top figure corresponds to a heterogeneous material where we can see an RVE. The bottom figure corresponds to the same structure but for a homogeneous material whose behaviour is equivalent to the heterogeneous material.

$$\nabla \boldsymbol{\sigma}^{\mathrm{M}} + \boldsymbol{f} = 0 \tag{3.3a}$$

$$\boldsymbol{\sigma}^{\mathrm{M}} \cdot \mathbf{n} = 0 \tag{3.3b}$$

$$\boldsymbol{\epsilon}^{\mathrm{M}} = \frac{1}{2}(\nabla \mathbf{u}^{\mathrm{M}} + (\nabla \mathbf{u}^{\mathrm{M}})^{\top}) \tag{3.3c}$$

$$\tag{3.3d}$$

where $\boldsymbol{\sigma}^{\mathrm{M}}$ is the macroscale stress, $\mathbf{n}$ the unit normal vector, and finally $\boldsymbol{\epsilon}^{\mathrm{M}}$ is the macroscale strain. The unknown constitutive relation can be expressed as

$$\boldsymbol{\sigma}^{\mathrm{M}} = \hat{\mathbf{D}} : \boldsymbol{\epsilon}^{\mathrm{M}} \tag{3.4}$$

and can be rewritten as

$$\begin{pmatrix} \boldsymbol{\sigma}^{\mathrm{M}}_{11} \\ \boldsymbol{\sigma}^{\mathrm{M}}_{22} \\ \boldsymbol{\sigma}^{\mathrm{M}}_{12} \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{D}}_{1111} & \hat{\mathbf{D}}_{1122} & \hat{\mathbf{D}}_{1112} \\ \hat{\mathbf{D}}_{2211} & \hat{\mathbf{D}}_{2222} & \hat{\mathbf{D}}_{2212} \\ \hat{\mathbf{D}}_{1211} & \hat{\mathbf{D}}_{1222} & \hat{\mathbf{D}}_{1212} \end{pmatrix} \begin{pmatrix} \boldsymbol{\epsilon}^{\mathrm{M}}_{11} \\ \boldsymbol{\epsilon}^{\mathrm{M}}_{22} \\ 2\boldsymbol{\epsilon}^{\mathrm{M}}_{12} \end{pmatrix} \tag{3.5}$$

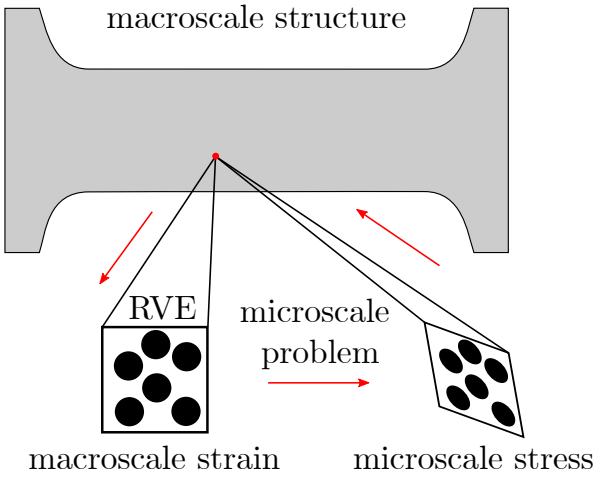


Figure 3.8: Sketch of the homogenisation procedure. A macroscopic strain is prescribed in the RVE. After solving the FE problem the microscopic stress is calculated in the RVE that is later averaged to get the macroscopic stress.

The purpose of homogenisation is to find the matrix $\hat{\mathbf{D}}$. To this end we perform calculations on the RVE, given some microscale strain $\boldsymbol{\epsilon}^{\mathrm{m}}$ we have

$$\nabla \boldsymbol{\sigma}^{\mathrm{m}} = 0 \tag{3.6a}$$

$$\boldsymbol{\sigma}^{\mathrm{m}} = \mathbf{D} : \boldsymbol{\epsilon}^{\mathrm{m}} \tag{3.6b}$$

where the constitutive relation is known and thus we can calculate the microscale stress, $\boldsymbol{\sigma}^{\mathrm{m}}$. To connect the microscale and macroscale we perform scale bridging by averaging

$$\boldsymbol{\sigma}^{\mathrm{M}} = \langle \boldsymbol{\sigma}^{\mathrm{m}} \rangle \tag{3.7a}$$

$$\boldsymbol{\epsilon}^{\mathrm{M}} = \langle \boldsymbol{\epsilon}^{\mathrm{m}} \rangle \tag{3.7b}$$

where $\langle \cdot \rangle$ denotes the mean operation.

Performing this procedure 3 times for 3 different micro strain tensors will fully define $\hat{\mathbf{D}}$. For instance, setting

$$\boldsymbol{\epsilon}^{\mathrm{M}} = \langle \boldsymbol{\epsilon}^{\mathrm{m}} \rangle = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \tag{3.8}$$

we get from [Eq 3.5]

$$\begin{pmatrix} \langle \boldsymbol{\sigma}^{\mathrm{m}}_{11} \rangle \\ \langle \boldsymbol{\sigma}^{\mathrm{m}}_{22} \rangle \\ \langle \boldsymbol{\sigma}^{\mathrm{m}}_{12} \rangle \end{pmatrix} = \begin{pmatrix} \hat{\mathbf{D}}_{1111} & \hat{\mathbf{D}}_{1122} & \hat{\mathbf{D}}_{1112} \\ \hat{\mathbf{D}}_{2211} & \hat{\mathbf{D}}_{2222} & \hat{\mathbf{D}}_{2212} \\ \hat{\mathbf{D}}_{1211} & \hat{\mathbf{D}}_{1222} & \hat{\mathbf{D}}_{1212} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \Rightarrow \begin{pmatrix} \hat{\mathbf{D}}_{1111} \\ \hat{\mathbf{D}}_{2211} \\ \hat{\mathbf{D}}_{1211} \end{pmatrix} = \begin{pmatrix} \langle \boldsymbol{\sigma}^{\mathrm{m}}_{11} \rangle \\ \langle \boldsymbol{\sigma}^{\mathrm{m}}_{22} \rangle \\ \langle \boldsymbol{\sigma}^{\mathrm{m}}_{12} \rangle \end{pmatrix} \tag{3.9}$$

and if we repeat the same procedure for

$$\boldsymbol{\epsilon}^{\mathrm{M}} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, \quad \boldsymbol{\epsilon}^{\mathrm{M}} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{3.10}$$

we can define the other 6 components of $\hat{\mathbf{D}}$

3.4.2 Concurrent multiscale modelling

When the scales cannot be separated, scientists resort to domain decomposition-based approaches. The results of homogenisation are applied to the boundary of regions of interest for concurrent microscale corrections to be performed [Raghavan and Ghosh, 2004; Oden et al., 2006; Kerfriden et al., 2009; Hesthaven et al., 2015; Paladim et al., 2016]. An example can be found in [Fig 3.9]. We can see that the concurrent model is composed of the coarse and fine scale models. In the coarse scale the heterogeneities are represented on average through homogenisation, while on the fine scale the heterogeneities are represented explicitly. In contrast to homogenisation the 2 models are coupled directly.

These approaches are computationally more expensive and practically more intrusive than methods based on RVEs. However, their deployment is necessary when predicting the microscale response to fast macroscale gradients, for instance due to sharp macroscale geometrical features (cracks, notches, sharp corners). The work done in this PhD falls in this category of multiscale methods and will be explained in detail in section [4.5].

3.5 ML assisted multiscale methods

Multiscale computational modelling may be coupled to offline/online acceleration methods such as model order reduction (MOR) techniques [Barrault et al., 2004; Ryckelynck, 2009; Goury et al., 2016] and surrogate models [Ghanem and Spanos, 1991; Xiao et al.,

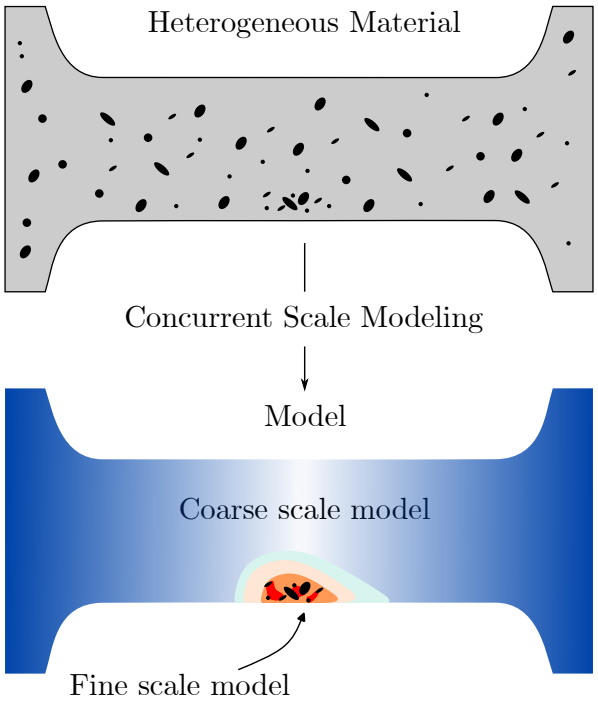


Figure 3.9: Sketch of the concurrent scale modelling procedure. The top image corresponds to the heterogeneous material. The bottom image corresponds to the model that is composed of the coarse and fine scale.

2009]. The idea is to realise many expensive computations in advance, subject to parameter variations, approximate the family of generated solutions using statistical regression, and use the statistical model online to produce solutions inexpensively. This is the approach followed in this PhD where a NN based surrogate modelling approach is developed to inexpensively generate microscale mechanical corrections given the result of coarse scale simulations.

An early work coupling machine learning and multiscale mechanics can be found in [Goury et al., 2016] where the authors want to perform computational homogenisation in multiscale elastic-damageable particulate composites. In order to reduce the computational cost they develop an online/offline reduced basis strategy. Later, [Bessa et al., 2017] created a dataset involving plasticity and damage calculations on RVEs, using the reduced order modelling technique described in [Liu et al., 2016] which uses a k-means clustering algorithm offline and introduces a method called self-consistent clustering analysis for the online stage. The dataset is created by sampling different microstructures, phase properties and external conditions. Two machine learning frameworks are being studied, depending on the size of the dataset, namely kriging and NNs. Moreover,

[Li et al., 2019] developed a topology optimisation framework for multiscale structures where a NN is used to compute the non-linear material responses, replacing the constitutive law used for the macroscale with a homogenised response of the microstructure for each point in the macroscale. Recently, [Saha et al., 2020] developed a multilevel NN approach to solve a variety of computational science and engineering problems, including homogenisation in multiscale problems. The offline stage deals with the microscale where a NN is trained to map the average micro stress in the RVE given the macro strain and material parameters, while the macroscale is solved online using a PINN.

3.6 Conclusions of the chapter

In this section we presented different surrogate models to reproduce the output of Finite Element (FE) solvers. We discussed why classical surrogate models like Gaussian Process and Polynomial Chaos Expansion cannot be used in the kind of problems examined in this thesis. Additionally, we gave examples from the literature of how Neural Networks can efficiently be used as FE surrogates in a variety of engineering fields. Furthermore, we presented two common approaches to solve multiscale problems, namely homogenisation and concurrent scale modelling. Lastly, works utilising Neural Network assisted methods to solve multiscale problems were presented.

Chapter 4

Multiscale problem formulation, structures of interest and accuracy metric

4.1 Introduction

In this chapter, we introduce the fundamental ideas that underlie the proposed approach. We present the structures of interest and describe the proposed multiscale framework. Moreover, we provide the mathematical equations that the NN is designed to approximate. Lastly, we explain the different quantities of interest being used and the accuracy metric employed to evaluate the performance of the NN.

4.2 Elasticity

We consider a 3D body occupying domain $\Omega_0 \in \mathbb{R}^3$ with a boundary $\partial\Omega_0 = S_0$. The body is subjected to prescribed displacements $\mathbf{U}_D$ on its boundary $\partial\Omega_{u,0}$ and prescribed tractions $\mathbf{T}_D$ on the complementary boundary $\partial\Omega_{T,0} = \partial\Omega \backslash \partial\Omega_{u,0}$. We consider a body force $\mathbf{f}$ and a displacement $\mathbf{u} : \Omega_0 \rightarrow \mathbb{R}^3$ and the associated deformed configuration $\Omega = \{X \in \Omega_0, \hat{X} = X + \mathbf{u}(X)\}$. The boundary value problem of hyperelasticity consists in finding $\mathbf{u}(X) = \arg\min_{\mathbf{u}^*} E_p(\mathbf{u}^*)$ where the potential energy is defined as

$$E_p(\mathbf{u}) = \int_{\Omega_0} W(\boldsymbol{E})\, d\Omega_0 - \int_{\Omega_0} \boldsymbol{f_0}\mathbf{u}\, d\Omega_0 + \int_{\partial\Omega_{T,0}} \boldsymbol{T_0}\mathbf{u}\, dS_0 \tag{4.1}$$

We consider a linear Saint-Venant–Kirchhoff material model defined by its strain energy density

$$W(\boldsymbol{E}) = \frac{1}{2}\lambda[\mathrm{tr}(\boldsymbol{E})]^2 + \mu \mathrm{tr}(\boldsymbol{E}^2) \tag{4.2}$$

where the Green-Lagrange strain tensor $\boldsymbol{E}$ is defined by

$$\boldsymbol{E} = \frac{1}{2}(\boldsymbol{F}^T\boldsymbol{F} - I) \tag{4.3}$$

with the definition of the deformation tensor

$$\boldsymbol{F} = \frac{\partial \hat{X}}{\partial X} \tag{4.4}$$

In the equations above, λ and μ are the Lamé elasticity parameters and I denotes the identity tensor.

The Cauchy stress tensor $\boldsymbol{\sigma}$ may be calculated as follows:

$$\boldsymbol{\sigma} = \frac{1}{J}\boldsymbol{F}\frac{\partial W}{\partial \boldsymbol{E}}\boldsymbol{F}^T \tag{4.5}$$

where the Jacobian of the deformation tensor reads as

$$J = \det(\boldsymbol{F}) \tag{4.6}$$

The prescribed volume force $\boldsymbol{f_0}$ and prescribed surface tractions $\boldsymbol{T_0}$ expressed in the reference configuration may be expressed as a function of their counterparts $\boldsymbol{f}$ and $\boldsymbol{T}$ in the deformed configuration, as

$$\boldsymbol{f_0} = J\boldsymbol{f} \tag{4.7}$$

$$\boldsymbol{T_0} = J\left\|\boldsymbol{F}^{-T}\boldsymbol{n_0}\right\|\boldsymbol{T} \tag{4.8}$$

where $\boldsymbol{n_0}$ is the unit normal vector in the reference configuration.

for the special case of linear elasticity, the Green-Lagrange strain tensor is replaced by the linearised strain tensor $\boldsymbol{\epsilon}$:

$$\boldsymbol{E} \approx \boldsymbol{\epsilon} = \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^\top) \tag{4.9}$$

$$\boldsymbol{\sigma} = \frac{\partial W(\boldsymbol{\epsilon})}{\partial \boldsymbol{\epsilon}} \tag{4.10}$$

4.3 Equivalent stress

We are interested in predicting mechanical quantities that indicate potential crack initiation sites. Two possible choices that we use in this thesis are the Tresca and Von Mises stress.

- Tresca : $\sigma_T = \frac{1}{2}(\sigma_{\max} - \sigma_{\min})$
- Von Mises: $\sigma_{\text{VM}} = \sqrt{\frac{1}{2}[(\sigma_{xx} - \sigma_{yy})^2 + (\sigma_{xx} - \sigma_{zz})^2 + (\sigma_{yy} - \sigma_{zz})^2] + 3(\sigma_{xy}^2 + \sigma_{xz}^2 + \sigma_{yz}^2)}$

where $\sigma_{\max}$ and $\sigma_{\min}$ are the maximum and minimum principal stress respectively.

4.4 Porous medium

Here we give an example of a typical multiscale structure that we examine in this thesis. An example of the structure, that we will refer to from now on as "dogbone", can be found in [Fig 4.1]. The dogbone specimen has a cylindrical hole as a macroscale feature. The material is porous, made of randomly distributed spherical pores as microscale features.

4.5 Multiscale problem

All the examples presented in this thesis correspond to porous media made of a homogeneous matrix with a random distribution of spherical voids. At the coarse scale level, the voids are ignored, and the fine scale constitutive law is used as a homogeneous material model for the entire structure, without further adjustment of the elasticity coefficients. We could have used various homogenisation schemes to obtain macroscopically accurate homogenised coefficients, but the approach followed in this thesis does not require the use of such a finely tuned homogenisation model.

We surmise that there exists a function $\mathcal{F}$ that takes as input the geometry of the microscale features and the local macroscale solution in a window $B \subset \Omega$, which we call patch, and outputs the microscale stress field in a sub-region of B namely $\hat{B} \subset B$ that we call Region of Interest (RoI) [Fig 4.2]. Essentially, the role of the NN developed in this thesis is to learn the function $\mathcal{F}$.

Finally, we note that $\mathcal{F}$ is a function to be learned by examples, which is why we do not expect the choice of the homogenised model to have a significant impact on the

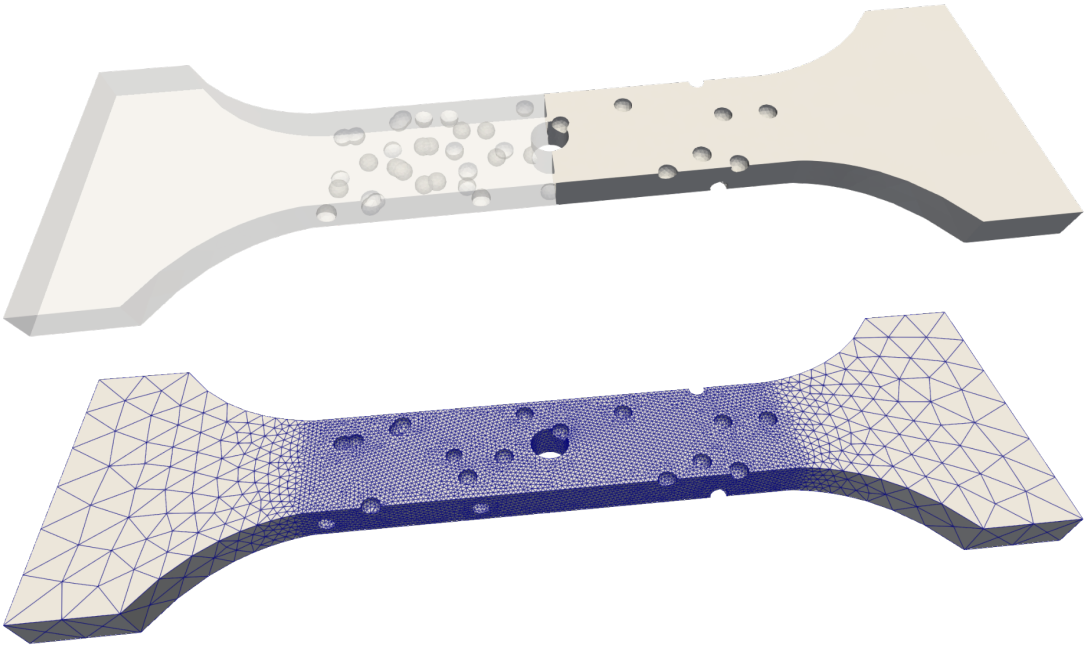


Figure 4.1: In the top subfigure, we can see a realisation of the dogbone structure. The dogbone is porous and it has a cylindrical hole in the middle. The porous phase is geometrically defined as the union of randomly distributed spheres. In the bottom subfigure, we see the mesh of the dogbone. We observe that the mesh is denser in the middle of the structure, where the porous phase is present, and coarser everywhere else.

quality of the result. The NN will be given sufficient amount of macro/micro stress pairs to compensate for systematic macroscale inaccuracies.

4.6 Accuracy

In this thesis we will define accuracy in a way that serves the purpose of accurately predicting the maximum equivalent stress in the RoI of the patches. We consider that the maximum equivalent stress is accurately calculated in the ROI of the patch if it is predicted with a relative error less than the threshold of 10%, unless stated otherwise. Given a set of patches, accuracy is defined as the percentage of patches for which the maximum equivalent stress is accurately predicted in the ROI of the patches. This procedure is summarised in [Algorithm 1] where **QoI** refers to either Von Mises or Tresca stress.

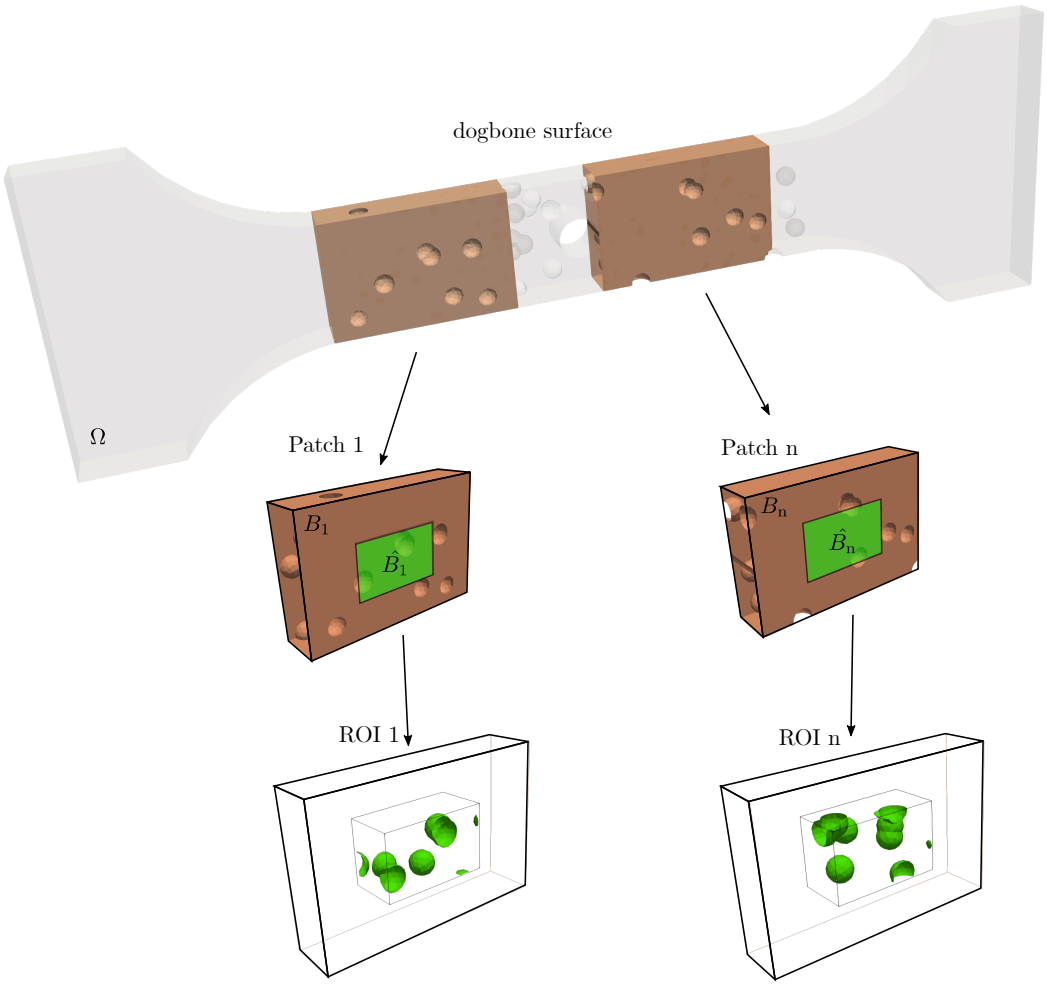


Figure 4.2: Porous material, Ω, with patches, B. We can observe a dogbone structure where we have extracted 2 patches. With green we can see the ROI of the patches, $\hat{B}$.

4.7 Conclusions of the chapter

In conclusion, this chapter acted as an introduction to the main concepts that the reader needs to be aware of to understand the topic of the thesis. Specifically, the types of examined structures were discussed along with the multiscale framework that will be used to analyse them. Additionally, we have provided an overview of the mathematical equations that the neural network is trying to approximate. Lastly, we explained the quantities of interest and the accuracy metric being used to evaluate the performance of the NN.

Algorithm 1 Compute accuracy

function ACC($datapoints$, $threshold = 0.1$)

    $N = \textbf{length}(datapoints)$
    $accepted = \textbf{zeros}(N)$

    for $patch$ **in** $datapoints$ **do**

        $S_{\text{NN}} = patch.\text{prediction}$ ▷ get predicted stress tensor for the current patch
        $S_{\text{FE}} = patch.\text{ground\_truth}$ ▷ get real stress tensor for the current patch

        $S_{\text{NN\_ROI}} = S_{\text{NN}}[patch.\text{ROI}]$ ▷ extract predicted stress values from the ROI
        $S_{\text{FE\_ROI}} = S_{\text{FE}}[patch.\text{ROI}]$ ▷ extract real stress values from the ROI

        $y_{\text{NN}} = \textbf{QoI}(S_{\text{NN\_ROI}})$ ▷ calculate the QoI for the predicted stress tensor
        $y_{\text{FE}} = \textbf{QoI}(S_{\text{FE\_ROI}})$ ▷ calculate the QoI for the real stress tensor

        $y_{\text{NN\_max}} = \textbf{max}(y_{\text{NN}})$ ▷ get the maximum QoI for the predicted stress tensor
        $y_{\text{FE\_max}} = \textbf{max}(y_{\text{FE}})$ ▷ get the maximum QoI for the real stress tensor

        $error = |y_{\text{NN\_max}} - y_{\text{FE\_max}}|/y_{\text{FE\_max}}$ ▷ calculate the relative error

        if $error \leq threshold$ **then** ▷ decide if the error is acceptable
            $accepted[patch.\text{ROI}] = 1$

    $accuracy = \textbf{sum}(accepted)/N$

    return $accuracy$

Chapter 5

Convolutional Neural Networks for the prediction of equivalent stress in 2D porous structures

5.1 Introduction

Multiscale computational modelling is challenging due to the high computational cost of direct numerical simulation by finite elements. In this chapter we propose a Convolutional Neural Network (CNN) based multiscale surrogate methodology, which can be used to perform fast stress predictions in 2D structures exhibiting spatially random microscopic features. In contrast to classical surrogate modelling techniques like Gaussian Processes or Polynomial Chaos Expansion, CNNs can efficiently operate on image data. The proposed method does not assume scale separability, and does not require prior parametrisation of the multiscale problem.

The idea of starting from a 2D case before applying the methodology to more realistic 3D cases emerged organically in the course of this PhD since the computational cost associated with 2D problems is much smaller compared to 3D cases, while all the concepts can be directly extended to 3D structures. The purpose of this chapter is to show that

- The coarse scale stress information in the patch along with the geometry information are sufficient for a NN model to make a fine scale prediction in the ROI.
- A CNN can be used as the aforementioned NN model, which operates on Finite

Element results converted to images, and in contrast to the relevant literature is able to make predictions in cases where multiple microscale features are interacting with each other and the macroscale features.

- Mechanically consistent rotations can be used as a data augmentation technique to reduce the training data requirements of the model, which is one of the main problems reported by researchers in their attempts to apply NNs to realistic engineering applications.

- The Bayes by Backprop method can be used to extract meaningful uncertainty information from the network, thus converting a deterministic CNN to a probabilistic one. In contrast to the NNs found in the relevant literature, this probabilistic CNN is able to provide reliable predictions, which is of crucial importance in engineering applications.

- The uncertainty information extracted from the CNN can be used in a Selective Learning framework to reduce the number of labelled data required for training, which tends to be the most expensive part of the training data generation.

5.2 Convolutional Neural Network

In this section we analyse the CNN we use to reproduce the FE result of the 2D problem. We explain the input and output of the network as well as its architecture.

5.2.1 Input-Output

As already discussed, the input of the network is patches extracted from the structure and not the entire structure. An example of a 2D structure used in this chapter along with patches extracted from it can be found in [Fig 5.1]. The input of the CNN are the 3 independent components of the macro stress tensor along with the geometry of the patch and the output is the micro Tresca stress, as can be seen in [Fig 5.2]. Specifically, the input of the CNN is a 3D array of size $[N_x \times N_y \times N_C]$ where: N_x and N_y are the size of the input image along the x and y direction respectively and N_C is the number of channels of every data point. Each data point has 4 channels namely σ_{xx}, σ_{yy}, τ_{xy} and *Geometry* corresponding to the xx, yy, xy component of the macro stress tensor and a binary image of the geometry respectively. The output of the model is an $[N_x \times N_y]$

image corresponding to the micro Tresca stress. Note that we are only interested in the ROI of the patch so all the statistics during training and inference are calculated there. Because we want to identify the effect of microscale features on the macroscale stress, we will scale the output with a number that reflects the intensity of the macro stress field. This number is the sum of the absolute principal stresses of the macro stress tensor $|\sigma_{\max}| + |\sigma_{\min}|$. The micro stress in areas away from microscale features should be the same as the macroscale stress because these features only have a local effect. This suggests that the output should be constant away from the microscale features and change rapidly very close to them. That is clearly visible in [Fig 5.2].

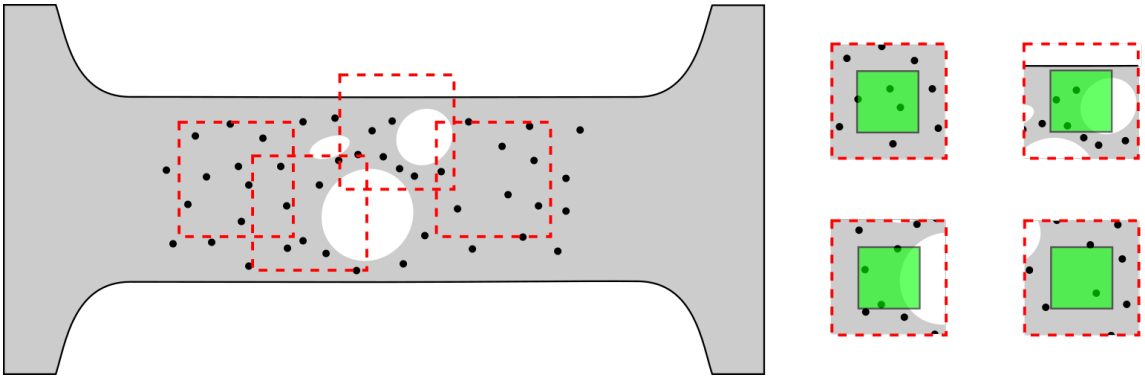


Figure 5.1: On the left the original structure and 4 patches that correspond to the red squares. On the right the extracted patches that will be fed to the Neural Network.

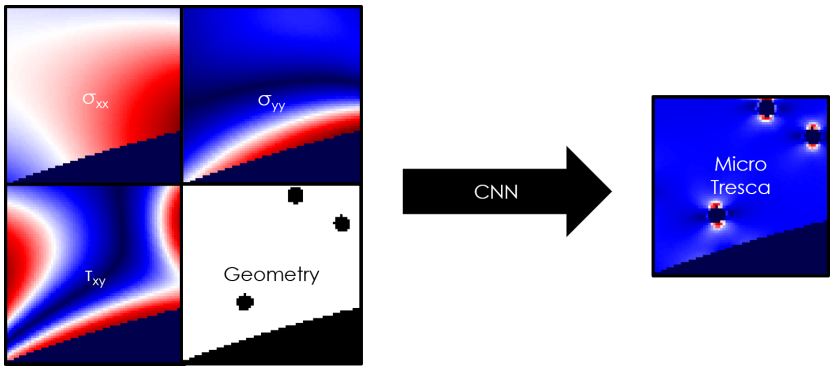


Figure 5.2: On the left, the input of the CNN (the three components of the macroscopic stress fields, converted into images, plus the binary image corresponding to the indicator function of the microstructure) and on the right the output, which is the microscale Tresca stress.

5.2.2 Loss function

The loss function used for the training of the deterministic networks in this chapter is the Mean Squared Error (MSE) between the scaled micro Tresca stress predicted by the CNN and the one calculated using direct microscale FEA. For the Bayesian networks the ELBO loss is used, as described in equation [2.15].

5.2.3 Architecture

The architecture of the network is inspired by the "StressNet", proposed by [Nie et al., 2019]. The network is an encoder decoder network. Three convolution blocks with increasing number of filters will downsample the input, after that five residual blocks are applied to the resulting array before using 3 deconvolution layers with a decreasing number of filters to upsample to the original dimension but with 1 channel instead of 4 [Fig 5.3].

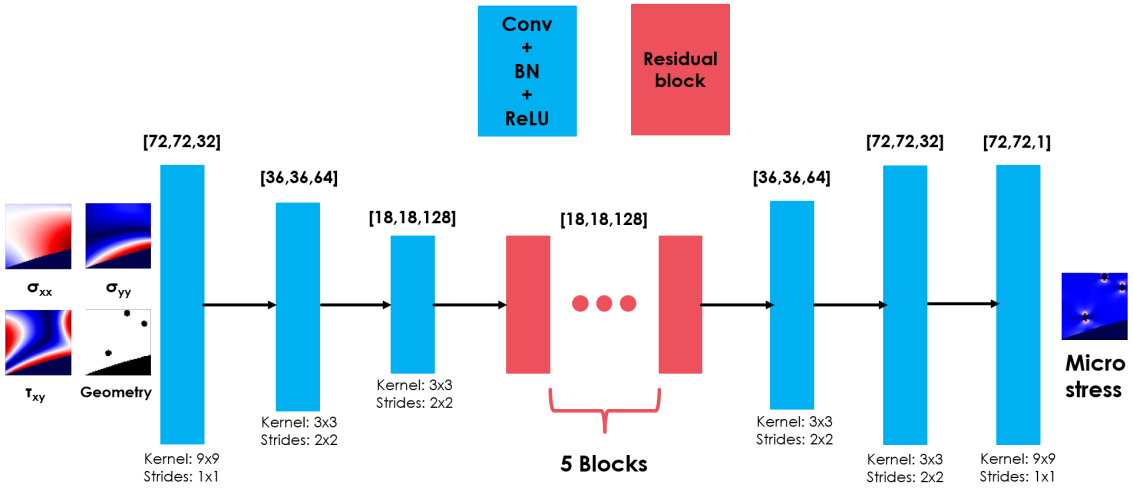


Figure 5.3: Structure of the CNN. σ_{xx}, σ_{yy} and τ_{xy} are the stress components on the x, y and xy direction respectively.

The residual blocks we will use in this work consist of two convolution layers, followed by a BN layer and a ReLU activation function each, and a Squeeze and Excitation block (SE) in the end [Fig 5.4]. The input and output of this block has exactly the same size as we choose the number of filters for the convolution layers to be the same as the number of filters at the input of the residual block.

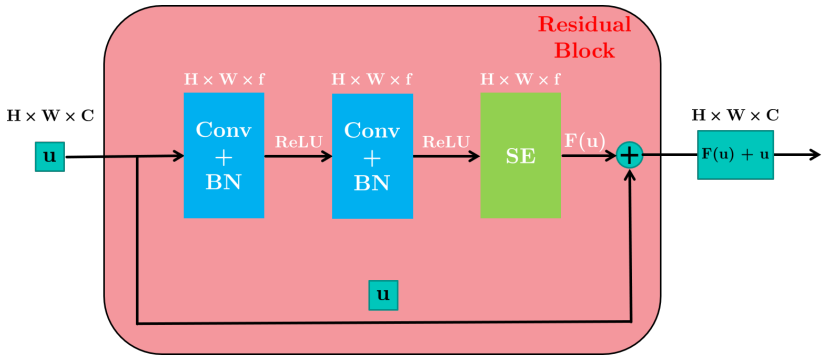


Figure 5.4: Structure of the residual block we are using with input u and output $F(u) + u$.

The SE block can adaptively recalibrate channel-wise feature responses by explicitly modelling interdependencies between channels resulting in improved generalization across datasets and improvement in the performance [Cheng et al., 2018; Li et al., 2018; Hu et al., 2018]. The input of the SE block has C channels, height H and width W, $[H \times W \times C]$. The input decreases in size using a global-averaging pooling layer resulting in a linear array of size $[1 \times C]$. After that, two fully connected layers downsample and then upsample the linear array. Firstly the linear array is downsampled by a factor of 16, $[1 \times C/16]$, as this is indicated to result in optimum performance [Hu et al., 2018], then a ReLU activation function is applied before upsampling again using a factor of 16 $[C/16 \cdot 16 = 1 \times C]$ and in the end a Sigmoid activation function is applied. Lastly, the linear array is reshaped to size $[1 \times 1 \times C]$ and multiplied with the input of the SE block [Fig 5.5].

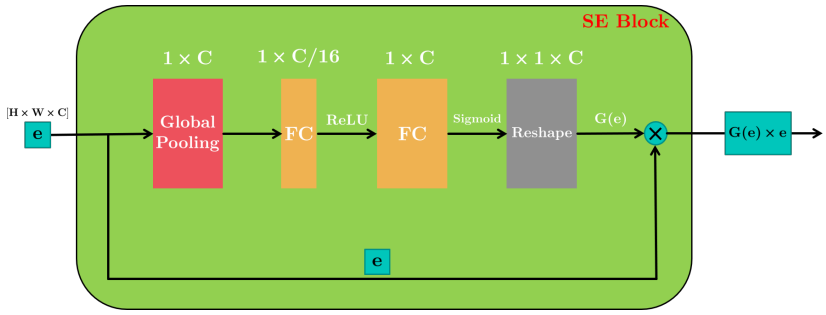


Figure 5.5: Structure of a generic SE block with input e and output $G(e) \times e$. FC stands for Fully Connected layer

5.3 Numerical examples

In this section we will present results for linear and nonlinear models both for deterministic and probabilistic CNNs. We will compare the CNN prediction with the FE prediction in the ROI level where we will be able to compare the 2 stress distributions and we will also compare the maximum values in all the ROIs in the test set. Additionally, we will introduce a framework that can be used to reduce the amount of labelled data, namely selective learning. We demonstrate that selective learning can lead to a 50% reduction in the labelled data requirement. Lastly, we demonstrate how mechanically consistent rotations can be used to reduce the training data requirements of the model.

5.3.1 Linear elasticity

Training dataset

For the purpose of training our model we have assumed a distribution of elliptical pores as macroscale features. We consider all the microscale features as disks with the same radius, R. The Young's modulus and the Poisson ratio of the structure are 1 and 0.3 respectively. We consider the linear elastic case and we will discuss the non-linear elasticity in [section 5.3.2]. We assume that for a distance larger than 4 radii from the centre of the microscale features the micro effect on the global stress field is negligible, for instance in the case of an infinite plate under uniaxial loading the maximum stress at $r = 4R$ is 1.04 times the macro stress [Pilkey and Pilkey, 2008]. It is assumed that the micro feature length is $2R$, and the interaction length is equal to $3R$. Given those 2 parameters we conclude that the patch length should be $18R$ and the ROI should be a $[8R \times 8R]$ window in the middle of the patch as shown in [Fig 5.6]. We chose $R = 4$ so the input is of size $[72 \times 72 \times 4]$, the output is of size $[72 \times 72]$ and the ROI is of size $[32 \times 32]$.

In this example we want to study the interaction between elliptical macroscale features and spherical microscale pores in an infinite domain. In order to achieve this, the boundary conditions are applied to a buffer area where the mesh is much coarser, as can be seen in [Fig 5.7]. The buffer area allows us to apply boundary conditions without introducing boundary effects on the fine mesh area. Additionally, because the mesh in the buffer area is very coarse the computational cost remains practically the same. We apply displacement as boundary conditions [Eq. 5.1].

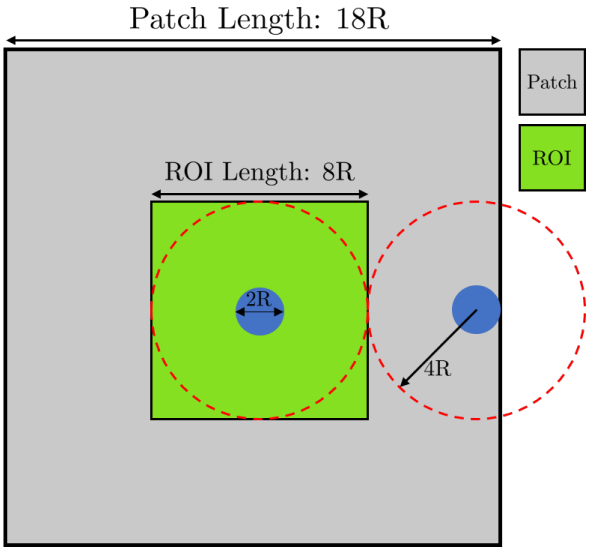


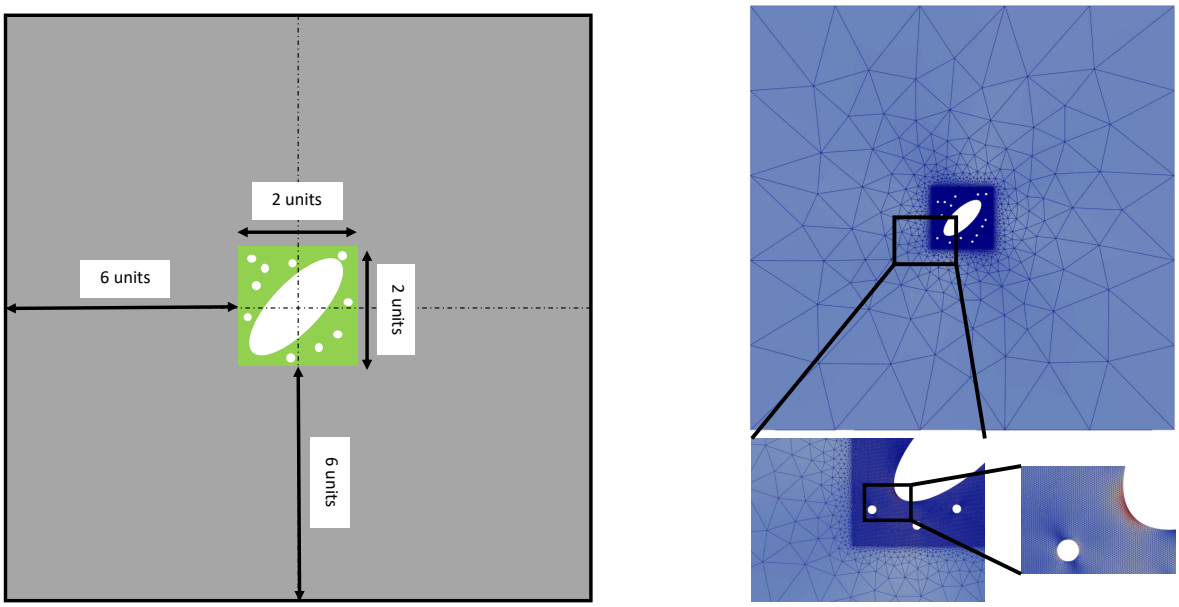
Figure 5.6: A sketch of the patch. In grey we see the patch, in green the region of interest and in blue we can see disks of radius equal to that of the microscopic pores.

$$u = \begin{bmatrix} E_{xx} & E_{xy} \\ E_{xy} & E_{yy} \end{bmatrix} (X - X_0)^\top \tag{5.1}$$

where E_{xx} is the far field displacement along the x direction, E_{xy} is the far field displacement along the xy direction, E_{yy} is the far field displacement along the y direction, X is the position of a point in $\mathbb{R}^2$ and X_0 is the initial position of the centre of the body in $\mathbb{R}^2$.

Scaling

Differences in the scales across input variables may increase the difficulty to model the problem, for example increased difficulty for the optimizer to converge to a local minimum or unstable behaviour of the network, thus a standard practice is to pre-process the input data usually with a simple linear rescaling [Bishop, 1995]. In our case we will scale the data not only to improve the model but also to restrict the space we have to explore. The space that we have to cover is infinite because the input can take any real value. Fortunately, Tresca stress scales linearly with the components of the stress tensor. This becomes obvious if we consider how the Tresca stress is calculated:



(a) (b)

Figure 5.7: On the left (a), a sketch of the buffer zone in grey and on the right (b) an example where the mesh and buffer area are visible.

The stress tensor can be rotated

$$\sigma' = Q \cdot \sigma \cdot Q^{\top} \tag{5.2}$$

using rotation matrix

$$Q = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{5.3}$$

The components of the stress tensor obtained after rotation are as follows

$$\sigma'_{xx} = \sigma_{xx}\cos^2\theta + \sigma_{yy}\sin^2\theta + 2\,\tau_{xy}\sin\theta\cos\theta \tag{5.4a}$$

$$\sigma'_{yy} = \sigma_{xx}\sin^2\theta + \sigma_{yy}\cos^2\theta - 2\,\tau_{xy}\sin\theta\cos\theta \tag{5.4b}$$

$$\tau'_{xy} = (\sigma_{yy} - \sigma_{xx})\sin\theta\cos\theta + \tau_{xy}(\cos^2\theta - \sin^2\theta) \tag{5.4c}$$

From [Eq. 5.4c] we can find that there must be an angle θ_p such that the shear stress after rotation is zero:

$$\tan(2\theta_P) = \frac{2\tau_{xy}}{\sigma_{xx} - \sigma_{yy}} \tag{5.5}$$

After inserting θ_p into [Eq. 5.4a, 5.4b], the 2 principal stress components can be obtained:

$$\sigma_{\max}, \sigma_{\min} = \frac{\sigma_{xx} + \sigma_{yy}}{2} \pm \sqrt{\left(\frac{\sigma_{xx} - \sigma_{yy}}{2}\right)^2 + \tau_{xy}^2} \tag{5.6}$$

The Tresca stress σ_T is equal to $\sigma_T = \frac{1}{2}$ ($\sigma_{\max}$-$\sigma_{\min}$). From [Eq. 5.6] it is trivial to show that if we replace $\sigma_{xx},\ \tau_{xy},\ \sigma_{yy}$ to $k \cdot \sigma_{xx},\ k \cdot \tau_{xy},\ k \cdot \sigma_{yy}$ where k is a scaling factor, then $\sigma'_{\max} = k \cdot \sigma_{\max}$ and $\sigma'_{\min} = k \cdot \sigma_{\min}$ where $\sigma'_{\max}$ and $\sigma'_{\min}$ are the new principal stresses after scaling the input and thus $\sigma'_T = \frac{1}{2}(\sigma'_{\max} - \sigma'_{\min}) = \frac{1}{2}(k\sigma_{\max} - k\sigma_{\min}) = k \cdot \sigma_T$ where σ'_T is the Tresca stress after scaling. Additionally, because we model linear elastic problems scaling the load terms with a scaling factor k will result in a local and global stress field multiplied by k as well. Here k^{-1} is the maximum stress value present in all the 3 stress components over the patch. This scaling of the input values allows us to make predictions on input data of any possible scale. We just have to calculate k, multiply the input by k to transfer it to the desired scale and then multiply the output by k^{-1} to get the true output.

Numerical example with 1 ellipse

Firstly, we created an initial dataset with simple examples [Fig 5.8]. A single ellipse in the middle playing the role of the macroscale feature, creating a diverse macroscopic stress field. Also, a few micro features are randomly positioned around the ellipse, accounting for the microscale features that will affect the macro stress field. All the micro features have a circular shape and the same radius, $R = 4$ units. Additionally, regarding the boundary conditions, the far field displacements are sampled independently from a unit Gaussian. This strategy contributes to improving the generalisation ability of the network since it further diversifies the macroscopic stress field in the training dataset. From 500 examples, generated in 43 hours on an Intel® Core™ i7-6820HQ CPU, we extracted 33,000 patches, 5,000 of which were used as a validation set. The patches were extracted such that the union of all the ROIs is equal to the entire domain Ω. Specifically, the ROI of the first patch is aligned with the top right corner of the domain and the rest of the patches are created using a sliding window and a stride equal to half the length of the ROI in each dimension. The patches that do not contain any microscale or macroscale features are discarded.

Experiments on this dataset showed very positive results. Training with 28,000 patches and validating on 5,000 unseen patches resulted in a validation accuracy of

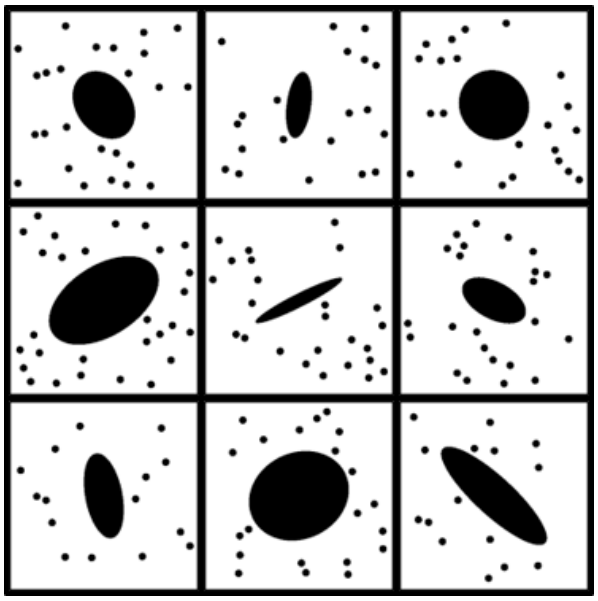


Figure 5.8: Nine examples from the "1 ellipse" dataset

96% when training with the Adam optimizer for 600 epochs, which required about 6 hours on a NVIDIA T4 GPU. Below, we can observe a diagram that shows the accuracy as a function of the relative error threshold [Fig 5.9].

We present results from 2 random patches [Fig 5.10] and then a result on the whole structure [Fig 5.11]. For the whole image the true Tresca stress is displayed and not the scaled version of it. The prediction is made again on the patch level and then the original image is reconstructed by stitching together the ROIs of the patches. This is possible if we align one corner of the ROI with a corner of the image and use a sliding window equal to the size of the ROI as can be seen in [Fig 5.12]. We can see that in all cases the CNN was able to accurately reconstruct the full micro stress field and it was also able to predict the maximum values with a very small error. Specifically, it is clear that away from the microscale features the microscale field is constant. We can also see that very close to the microscale features we have a very steep rise of the micro Tresca stress. The micro stress field is accurately predicted even in complicated cases where more than one micro features are interacting or micro features and macroscale features are interacting.

Lastly, we will investigate the effect of training with less data on the CNN accuracy. We randomly chose 10,000 patches, almost 30% of the available data, and train the NN with exactly the same settings. We make a prediction with both CNNs on the

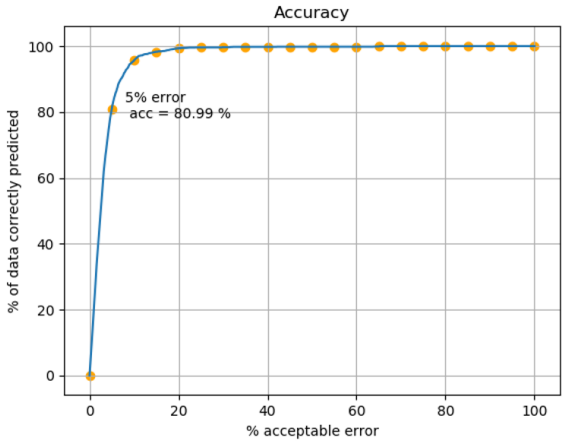


Figure 5.9: Accuracy as function of the threshold. Here, accuracy is defined as the percentage of patches in the test dataset for which the relative error between the maximum NN prediction and the maximum FE result in the ROI is less than a predefined threshold. This threshold corresponds to the x-axis of the diagram.

same patch. The results can be found in [Fig 5.13], where we can see that the CNN trained with more data better approximates the maximum stress value calculated by FEA compared to the CNN trained with less data. Additional experiments show that for the 10% threshold the accuracy is 6% higher for the large dataset even though we used almost 3 times as much data. This may look like a small increase, but it means that the mispredicted cases climbed from 4% to 10%. In reality most of the data that we rejected when we created the smaller training dataset (10,000 patches) were very similar to the ones accepted in the smaller dataset. A small portion of them contained new interactions that our network would have learned from. These cases are exactly the cases we are interested in because they contain complex examples that create strong interactions and sharp increase in the micro stress field. In section [5.3.3] we will demonstrate a Selective Learning framework that will allow us to identify these cases and train our network only on them. This way we keep the computational cost to the minimum while preserving the same level of accuracy.

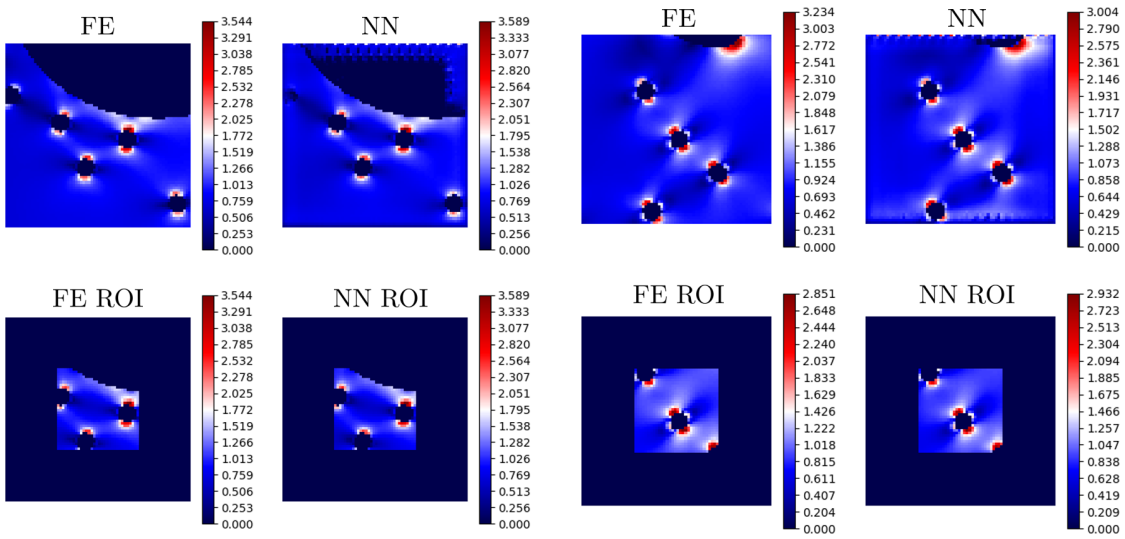


Figure 5.10: Evaluation of the Neural Network performance on 2 patches. On the top left of each example we see the scaled Tresca stress field computed by FEA and converted into an image for the whole patch and on the top right the NN prediction for the whole patch. On the second row we see the same but for the ROI.

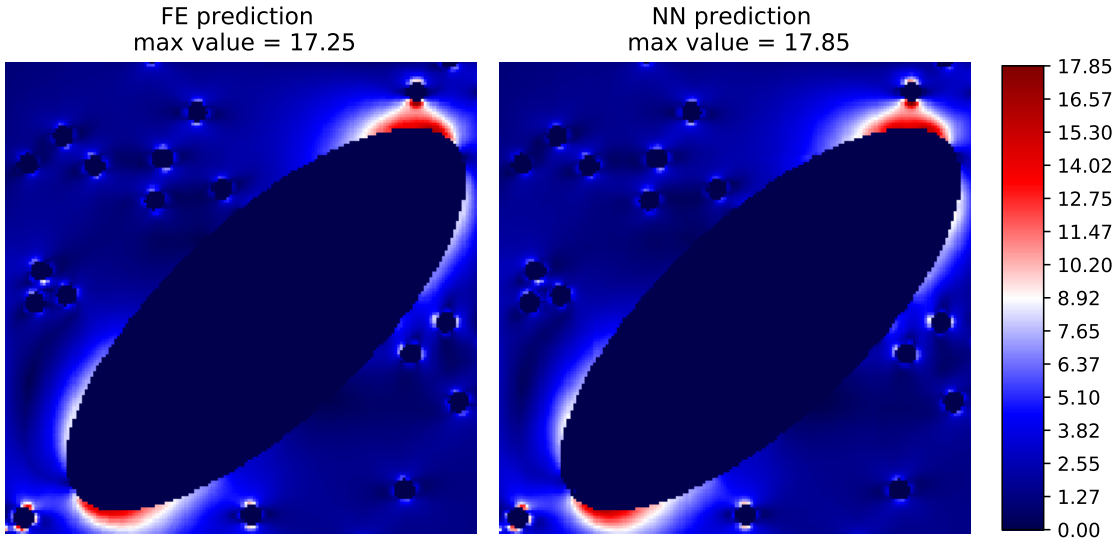


Figure 5.11: Comparison between the Tresca stress field computed by FEA and converted into an image, on the left, and an image reconstructed using the NN predictions on the patch level, on the right.

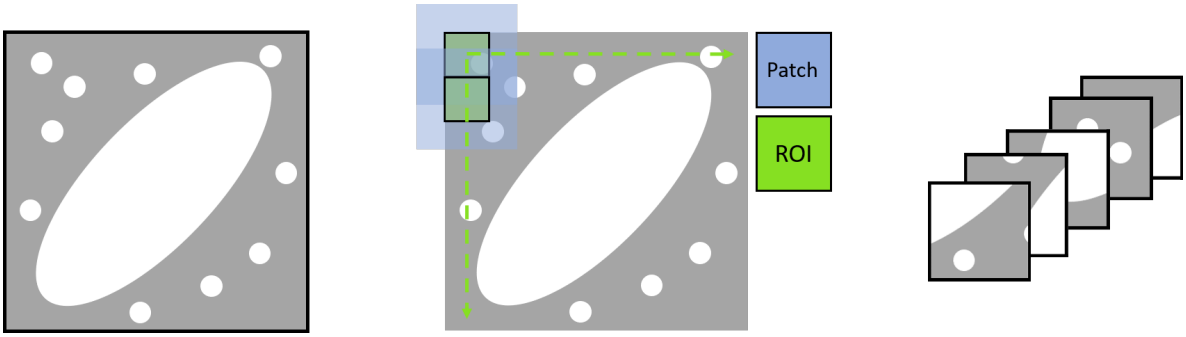
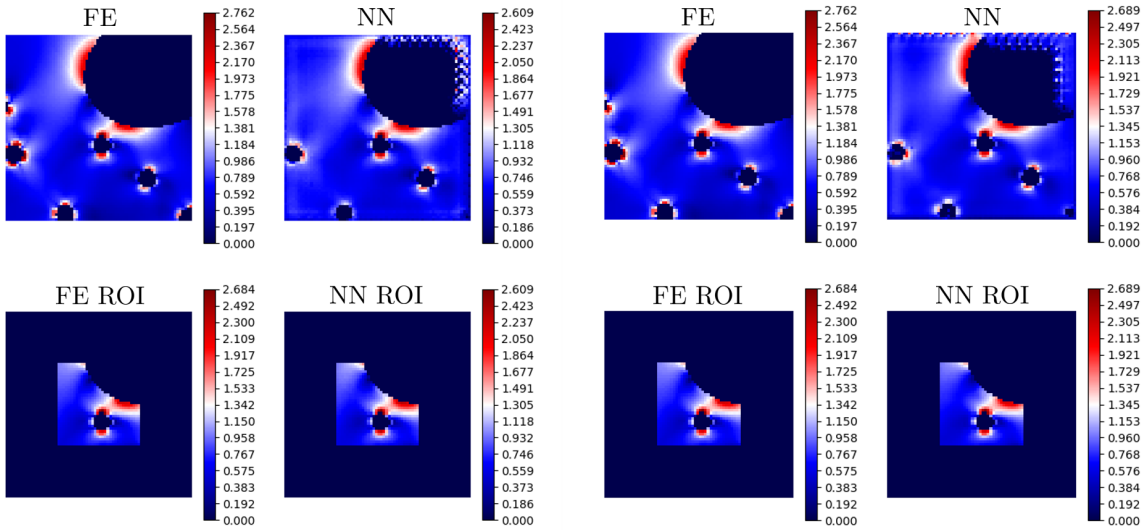


Figure 5.12: Patch generation for full image prediction. The corner of the ROI is aligned to the corner of the image and then a sliding window of size equal to the size of the ROI is used.



(a) (b)

Figure 5.13: A prediction for the same input for 2 identical NNs trained with 10,000 patches (a) and 28,000 (b). The two ROI predictions look very similar but we can observe that the maximum predicted value from the NN that was trained with the larger dataset is closer to the real value compared to the maximum predicted value from the NN that was trained with the smaller dataset.

Numerical example with 3 ellipses

Even though the CNN we trained seems to work well for the data it was trained on we do not expect the same level of accuracy as we depart from this dataset, although the method is fully non-parametric and the trained NN can make prediction for any unseen micro and macro geometries. Specifically, we would expect a decrease in accuracy in the following cases:

1. Spatially fast varying macroscale stress field, generated by macroscale features not present in the training dataset.

2. Microscale features not present in the training dataset, for instance non-circular holes.

3. Patterns of microscale features not present in the training dataset, for instance different distribution of circular holes.

To tackle this problem we created a new, more interesting, family of data with the expectation that this would add more complexity. As can be seen from [Fig 5.14] this new family of data has 3 ellipses as macroscale features and more disks as microscale features. At first, we used the old CNN to make predictions on the new dataset. We observed that the accuracy dropped from 96% to 72%. This implies two things. Firstly, the drop in accuracy means that the new dataset contains information that the network had never seen before or was unable to learn from (due to the sparsity of the examples), thus we can assume that training in this dataset will help the CNN to generalise better. Lastly, the concept of making the knowledge transferable seems to be working as we were able to make reasonable, but not perfect, predictions on a new family of data. This suggests that we managed to learn interactions between microscale features and the macro stress field and not just the structures themselves.

Training a CNN with the new dataset proved to be more challenging. By using 23,000 patches as training set (almost as many as with the original case) and 5,000 patches as a validation set, we obtained, with the same settings, a validation accuracy of 74% in contrast to the 96% in the first case. We believe that this happens not only because more microscale features are present in each case but also because the 3 ellipses are creating a much more complicated macro stress field. From experiments we found out that, as more and more new patches are added, the accuracy tends to increase

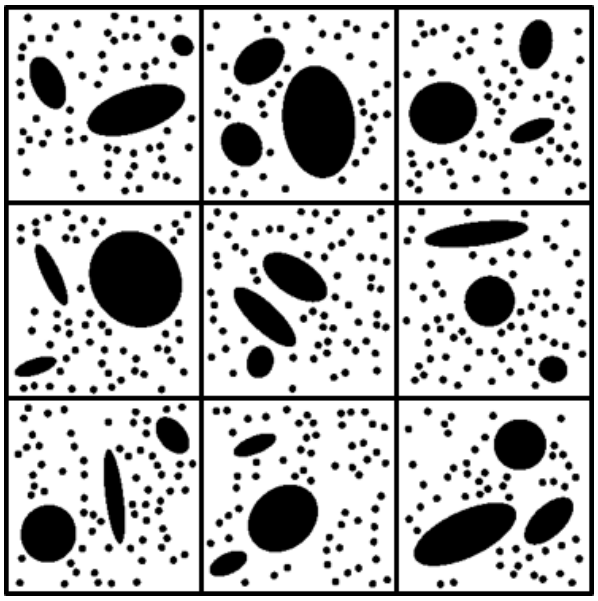


Figure 5.14: Nine random examples from the "3 ellipses" dataset

slower and slower. This happens because the new patches added tend to contain less and less new information.

When we tried to use this CNN to make predictions on the 1 Ellipse dataset the accuracy slightly improved, compared to the CNN trained with the 1 Ellipse dataset, from 96% to 96.7%. This small increase was expected since this CNN is trained with a dataset that contains all the necessary information to make predictions on the 1 Ellipse dataset and even more information that may or may not be useful. The very small increase in accuracy implies that the mispredicted cases are underrepresented in both datasets.

A common technique used to improve the performance of CNNs is data augmentation. Common data augmentation techniques for image data are shifting, flipping, rotating and zooming. Here we use rotation as data augmentation technique. We rotate mechanically the stress tensor [Eq. 5.2] and we use standard rotations for the images that represent the geometry.

We started from an initial training set of 5,000 patches ($\approx 1/4$ of the full set) and we randomly rotated the dataset 6 and 12 times. After training with the same settings for all the cases, a validation accuracy of 62%, 80% and 82% was achieved for the 0, 6, 12 rotations dataset respectively for the 10% threshold [Fig 5.15]. Firstly, this means that we managed to outperform by 8% the model trained with the full dataset and secondly,

we realised that rotating from 6 to 12 times did not add a significant amount of new information even though the data is doubled. Once more, that was the motivation to start working with Selective Learning. We can see an example of a prediction with all 3 CNNs on the same input [Fig 5.16], where the prediction improves with the number of rotations. We can also see a prediction of the CNN trained with 6 rotations on 4 random patches [Fig 5.17].

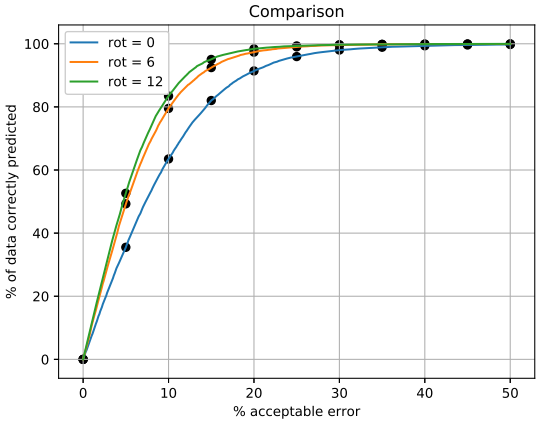


Figure 5.15: Comparison between 3 CNNs trained with the same settings but different datasets. Blue line corresponds to the original dataset with no rotation, orange line to a dataset with 6 rotations and finally the green line to a dataset with 12 rotations. We can observe that the accuracy increases as the number of rotations increases.

Moreover, we compare 2 CNNs with and without SE blocks. The 2 CNNs were trained with 27,000 training examples and validated on 3,000 validation examples. The CNN with the SE block reported accuracy of 78.98% while the CNN without the SE Block 68.39%. This clearly shows that adding the SE block in the Residual Blocks of the CNN substantially improves the performance.

Additionally, we investigate the smoothness of the solution. We compare the CNN prediction in 2 ROIs that share a common area. In [Fig 5.18] we have highlighted with orange dashed lines the CNN prediction in the common area of the 2 ROIs and we can see that it is the same in both of them.

Lastly, we perform a cross-validation study to confirm that the accuracy is not dependent on our choice of test dataset. Specifically, we used a dataset of 30,000 patches and we divided it in 5 subsamples of size 6,000 patches. We run 5 tests. Each time we used 1 of these 5 subsamples as a validation set and the rest as training test.

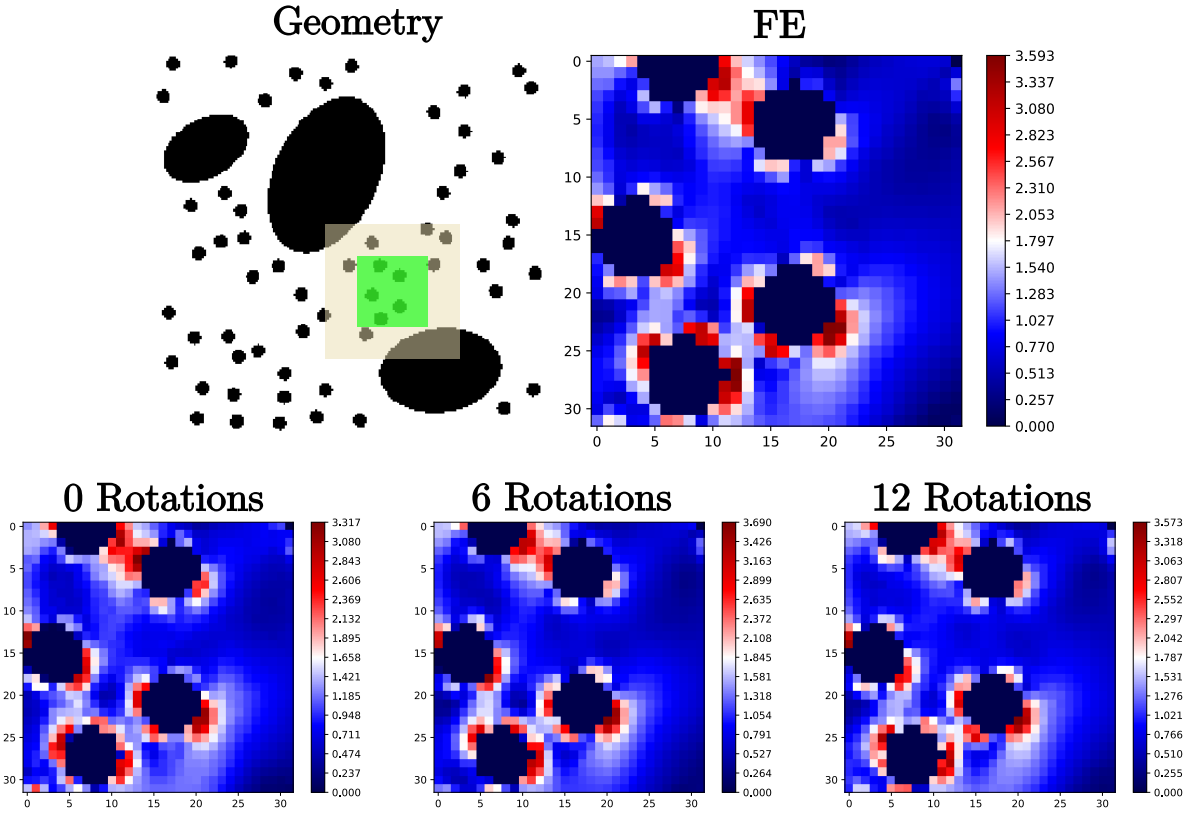


Figure 5.16: Top left corner the structure, the patch (with light brown) and the ROI (with green) for the prediction. Top right corner the scaled Tresca stress field in the ROI computed by FEA and converted into an image. Bottom from left to right, prediction in the ROI from a NN trained with a dataset with 0, 6 and 12 rotations respectively. We observe that even though those 3 images look quantitatively very similar, the predicted maximum value approaches the one calculated by the FE simulation as the number of rotations increases.

The mean accuracy for the validation set is 0.7813 (78.13%) and the standard deviation is 0.0174. The small value of the standard deviation implies that the CNN is stable and gives consistent results independent of the choice of test set, as long as the test set is large enough.

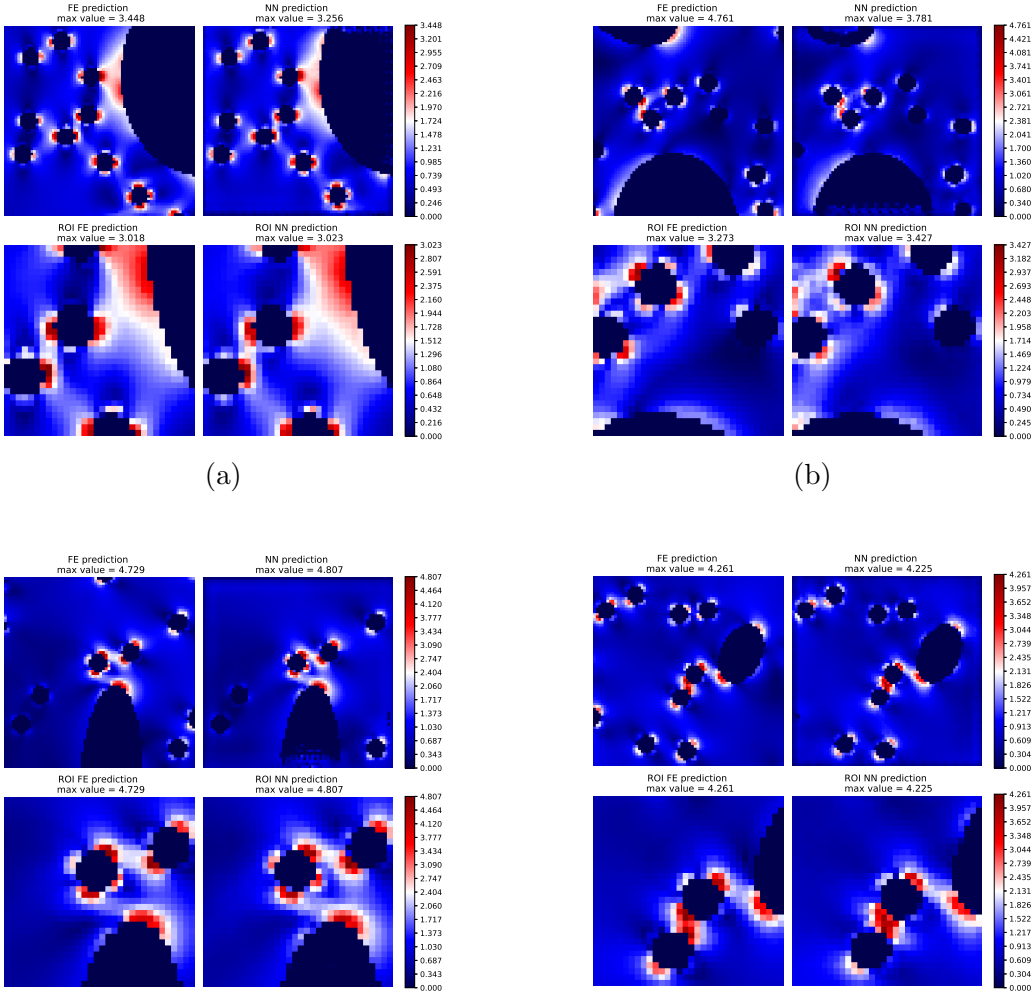


Figure 5.17: Evaluation of the performance of the CNN trained on the 6 rotation dataset on 4 patches. In each of the 4 images, on the first row we can see the scaled Tresca stress field for the entire patch computed by FEA and converted into an image on the left and the NN prediction for the entire patch on the right. On the second row we can see a zoom in the ROI of the above images.

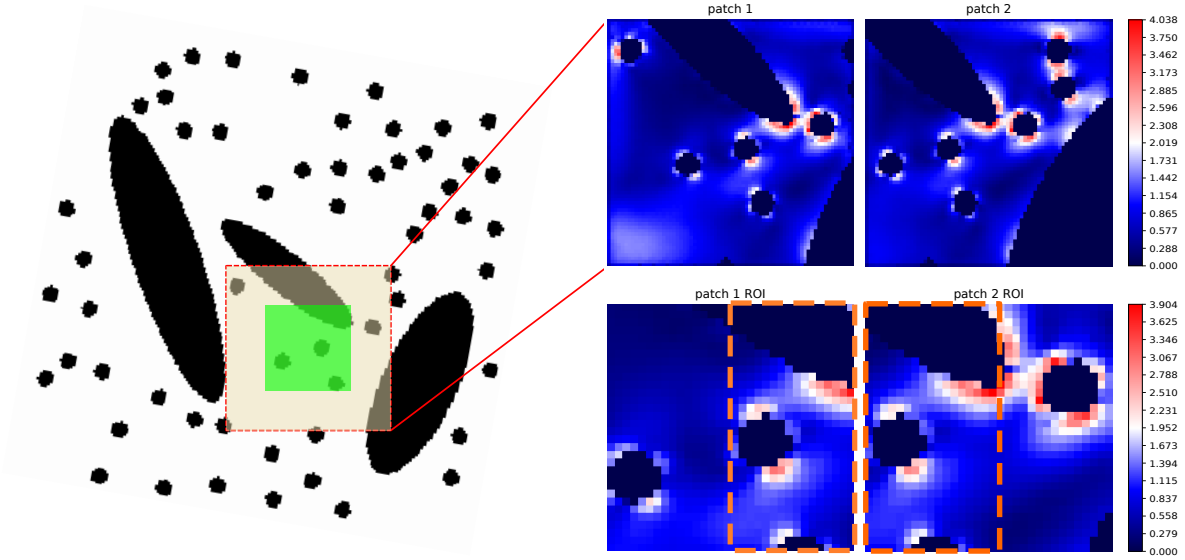


Figure 5.18: Prediction of the CNN in two patches with overlapping ROIs. On the left we see the geometry where we solve the FE problem on and we can also see the patch and the ROI. For clarity we only show 1 of the 2 patches. The second patch will be created by sliding to the right a sliding window with size half the size of the ROI. On the right we see 4 plots, the 2 top plots are the CNN prediction on the entire patch and the 2 bottom plots the CNN prediction in the ROI. The orange discontinues boxes correspond to the common area of the 2 ROIs. The CNN prediction in both boxes is the same.

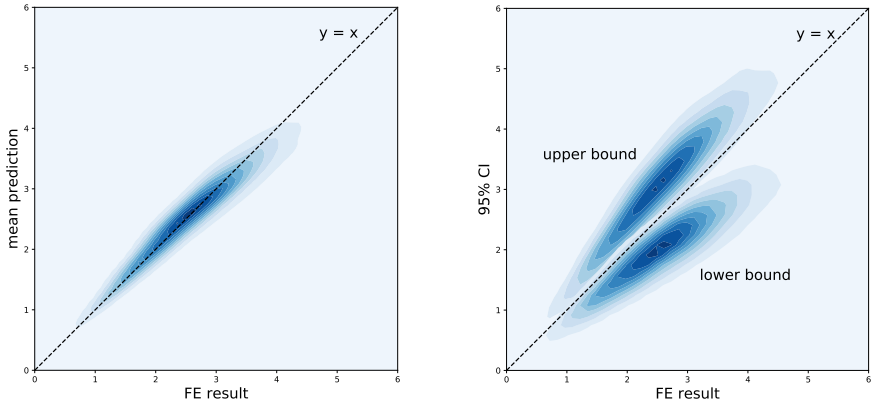
Numerical example using a Bayesian Neural Network

Until now we have used a deterministic neural network for the predictions. In this section we will present results corresponding to the use of the Bayesian NN. We trained the BNN with the same 5,000 patches as in section [5.3.1] for 600 epochs and validated on 10,000 patches. That requires 2.1 times more computational time compared to the deterministic case. The accuracy of the prediction is 72% for the 10% threshold compared to 62% for the deterministic case. In the Bayesian CNN case the accuracy is calculated using the mean network prediction. The mean and the variance of the BNN prediction are calculated by drawing the weights of the network from the posterior distribution 100 times and performing inference for every input.

The results for a BNN where the prior was optimised during training can be found in [Fig 5.19]. We can see from the first image, [Fig 5.19a], that the mean prediction is very close to the real value. We can also observe that for higher values we get higher absolute error. This is expected because those cases are represented to a lesser extent in the dataset. In the second image, [Fig 5.19b], we can observe that the $y = x$ line is almost always, and specifically for 92% of the patches, between the upper and lower 95% CIs. This means that the true solution is bounded by the 95% CIs for 92% of the patches.

Lastly, we trained a BNN where the prior parameters were not optimised during training. From [Fig 5.20a] we can see that the mean prediction is very good, a slight decrease of 2% is observed in the accuracy compared to the optimised prior BNN. Nevertheless, from [Fig 5.20b] and [Fig 5.20c] we can see that the uncertainty fails to explain the error as there are many cases where the $y = x$ line is either above the upper 95% CI or below the lower 95% CI. Specifically, the true value is bounded by the 95% CIs in 82% of the cases, a decrease of 10% compared to the optimised prior BNN.

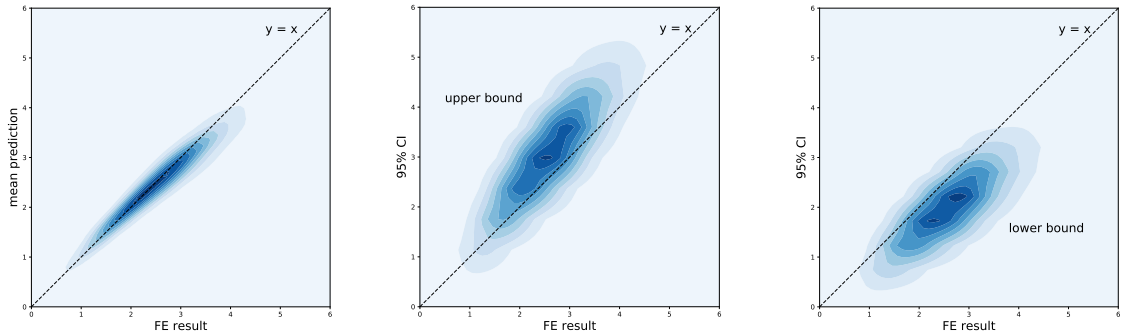
Results from the uncertainty estimation on image level can be found in [Fig 5.21]. On the top 2 cases [Fig 5.21a, 5.21b] we can see some examples of good mean predictions where there are clear interactions between multiple microscale features. The middle images [Fig 5.21c, 5.21d] are examples of good mean predictions where interactions between multiple microscale features and a macroscale feature can be seen. We can observe that the uncertainty, expressed as 1.96 $\times$ standard deviation, is higher in the vicinity of the higher error pixels indicating that the BNN has successfully identified the unseen interactions (interactions that where not in the training dataset or were underrepresented). [Fig 5.21e] is an example of a case where the maximum value is



(a) (b)

Figure 5.19: In these 2 figures we see point densities where darker colours correspond to higher point density. On the left (a), a diagram showing the relationship between BNNs' mean prediction and FE results for the maximum value in the ROI. We can clearly see that most of the points are on or near the $y = x$ line. On the right (b), a diagram showing the upper and lower 95% CIs for the prediction. We can observe that for most of the points the $y = x$ line is between the upper and lower 95% CIs.

mispredicted with a large error of about 1 unit. Fortunately, we can observe that the uncertainty is also very large, specifically 1.96 $\times$ standard deviation has a value of about 1.5 unit meaning that the true maximum value is between the mean prediction and the 95% CI. Image [Fig 5.21f] is an example of a case with low uncertainty and low error. This means that the CIs are very tight and the BNN is very confident about the prediction. That was an expectable result in the sense that this is a very simple case, 2 circular microscale features are weakly interacting, and we would expect from the BNN to handle it without a problem because the training dataset contains a very large number of these examples.



(a) Mean prediction (b) Upper 95% CI (c) Lower 95% CI

Figure 5.20: Three diagrams, depicting point densities where darker colours correspond to higher point density, corresponding to a BNN where the prior was not optimised during training. The prior distributions are Gaussian initialised as: $N(0, 1)$. First diagram (a) is the BNNs' mean prediction against the FE results for the maximum value in the ROI. Most of the points are on or near the $y = x$ line so the NN was able to provide good mean estimations. The next two diagrams correspond to the upper 95% CI (b) and the lower 95% CI (c). Ideally the point densities should not intersect with the $y = x$ line. The high percentage of points below the $y = x$ line for (b) and above the $y = x$ line for (c) indicates that the network was not able to successfully quantify the uncertainty.

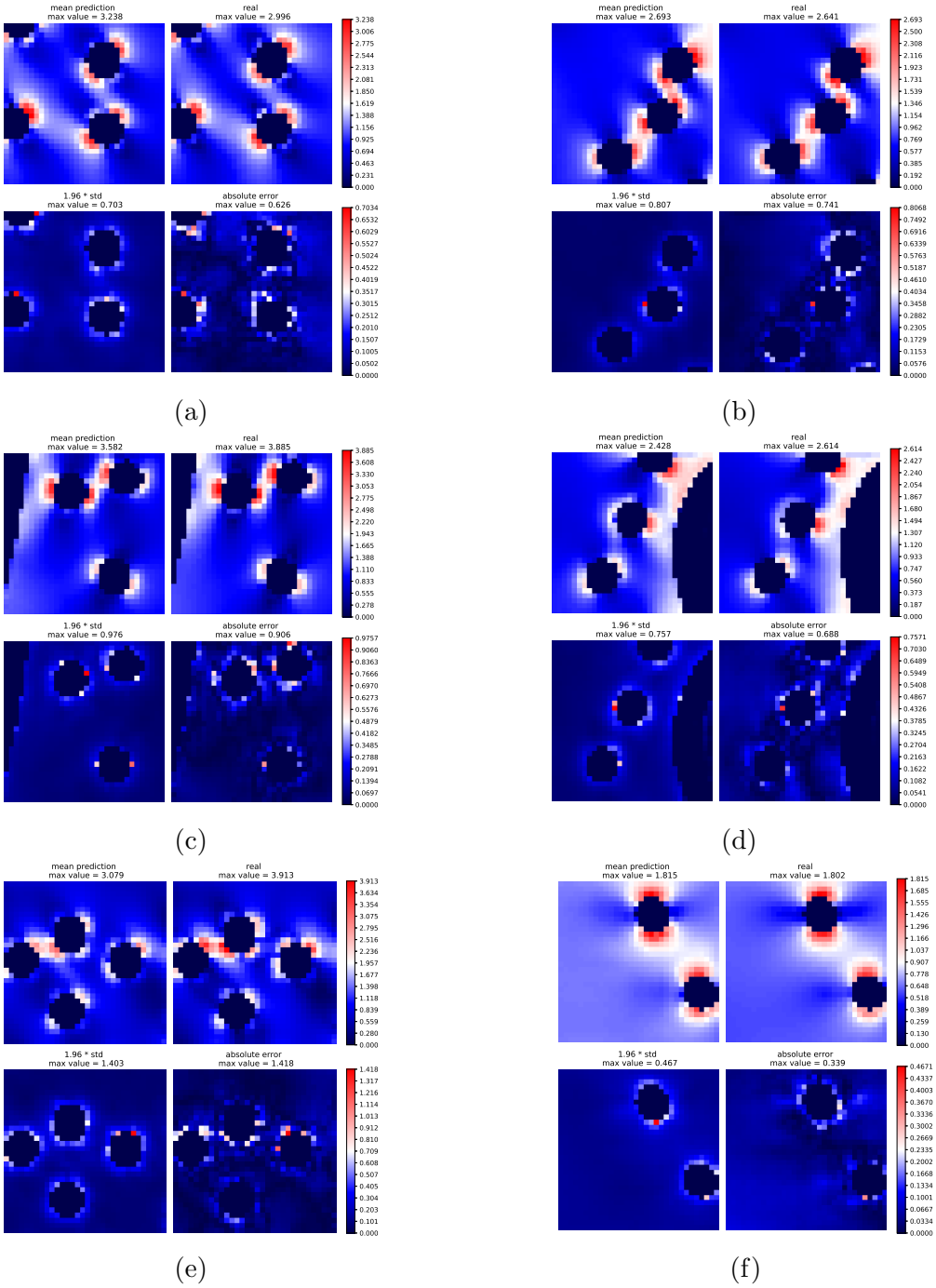


Figure 5.21: Examples of BNN predictions. All images correspond to the ROI of the patches. For each of the 6 images the first row corresponds to the NN mean prediction on the left and to the scaled Tresca stress field computed by FEA and converted into an image on the right. The second row corresponds to the NN uncertainty, expressed as 1.96 $\times$ standard deviation, on the left and to the absolute error between the NN mean prediction and the FE results on the right.

5.3.2 Nonlinear elasticity

In this section we demonstrate the applicability of our method for multiscale problems in finite strain elasticity. Compared to the previous section, the macroscale geometry of the dataset is constrained. This example can be used to showcase the ability of the framework to tackle microscale-informed stress analysis during the macroscale design of an engineering component.

Training dataset

The structure that we study in this section is rectangular with one macroscale geometrical feature and a distribution of disks as microscale features. The macroscale feature whose parameters can be optimised is composed of 2 disks with random radii and centres connected by their common external tangents [Fig 5.22]. Multiple instances of this structure are created by randomly choosing values for the macroscale parameters and changing the distribution of the microscale features. After training, the CNN will be able to evaluate the micro stress distribution in different macroscale geometries (of the same family) under any realisation of the microscale random texture.

To create the training dataset we choose length 5 and height 1 as overall dimensions for the structure. For the boundary conditions, the structure is clamped at $x = 0$ and a random displacement along the $-y$ direction ranging from 0 to 2 is applied to the other end, at $x = 5$, while the displacement along the x direction is zero. An example of the deformed structure for a displacement of size -1.93 can be found in [Fig 5.23]. The radius of the disks is 0.02 units or 4 pixels as in the linear elasticity case.

The FE solution is mapped into an image of size [80 $\times$ 400]. The patch and the ROI have the same size as the linear elasticity case [72 $\times$ 72] and [32 $\times$ 23] respectively.

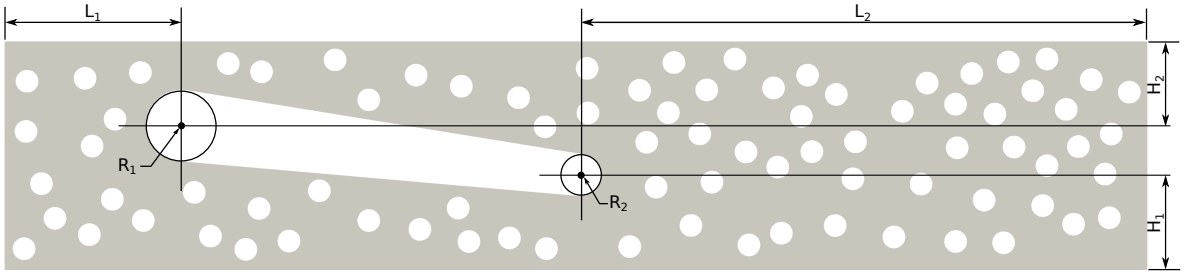


Figure 5.22: Structure in the reference (undeformed) configuration. $L_1 = 0.76$, $R_1 = 0.06$, $L_2 = 2.45$, $R_2 = 0.035$, the width of the structure is 5 and the height 1

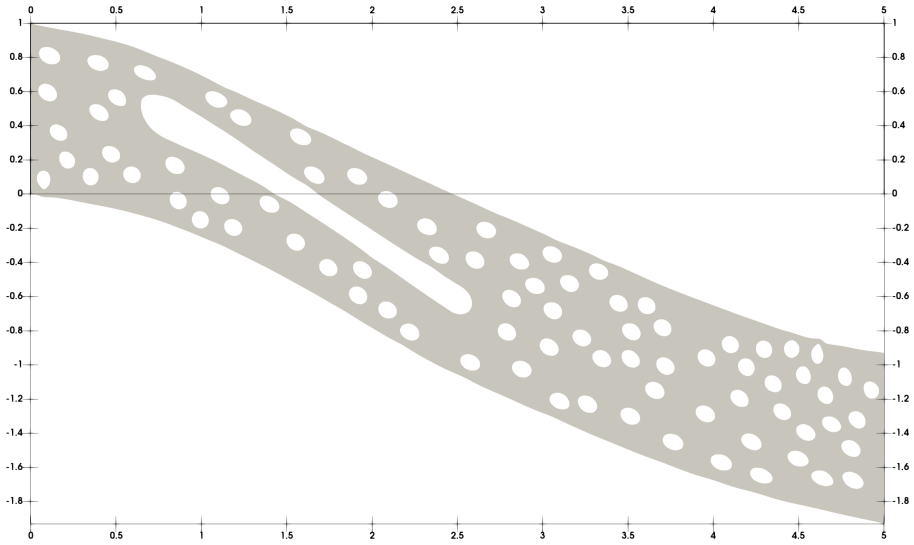


Figure 5.23: Structure in the deformed configuration (Finite Strain elasticity theory)

Numerical results in finite strain elasticity

We performed 200 FE simulations, which took 16 hours on an Intel® Core™ i7-6820HQ CPU, and we extracted 12,000 patches. Of those, 1,200 were used as a test set and 10,800 as a training set. We trained the same CNN as in the linear elastic case, [section 5.3.1], without scaling the data as a pre-processing step. Training with the Adam optimizer for 300 epochs required 3 hours on an NVIDIA T4 GPU. The results can be found in [Fig 5.24]. The accuracy for the 10% threshold is 73%.

Firstly, in [Fig 5.25] we show the difference between a structure modelled with linear elasticity and the same structure modelled with non-linear elasticity as has been calculated with FE simulations for the structure described in [Fig 5.23]. We observe that in general the 2 predictions are different and specifically that the linear elastic model underestimates the stress magnitude in regions of very large deformations.

Additionally, in [Fig 5.26] we see the prediction of the CNN in 4 patches. The two top rows include interactions between macroscale and microscale features. In both cases the maximum values are computed with a relative error less than 10% and the stress distribution is also correctly predicted. The two bottom rows include interactions between microscale features. In both cases the maximum values are computed with an error of less than 10% and the stress distribution is also correctly predicted.

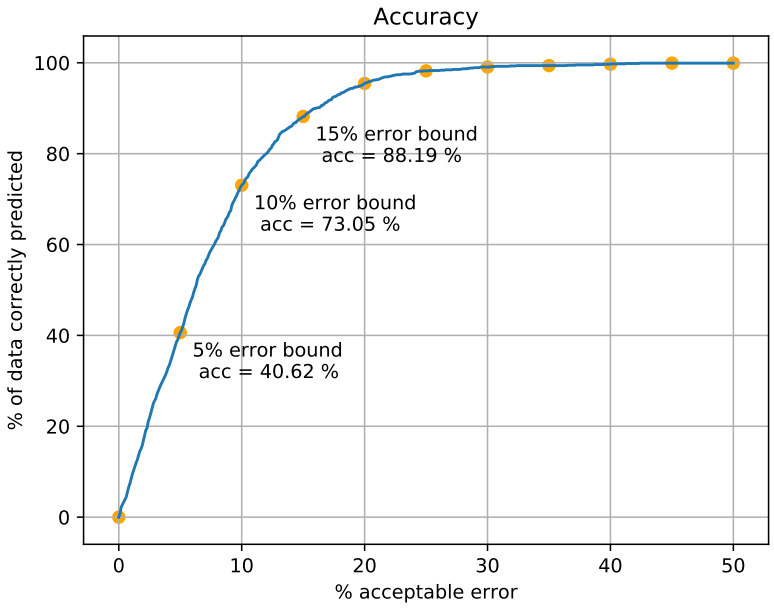


Figure 5.24: Accuracy as function of the threshold. The accuracy is defined as the percentage of patches in the dataset for which the relative error between the maximum NN prediction and the maximum FE result in the ROI is less than a predefined threshold. The accuracy for threshold values 5%, 10% and 15% is 40%, 73% and 88% respectively. For threshold values higher than 20% the accuracy is more than 95%.

Lastly, we show a comparison between the CNN prediction and the FE prediction in the entire structure. We remind the reader that the CNN prediction happens at patch level and then the patches are rearranged to create the entire solution field as has already been described in [Fig 5.12]. In [Fig 5.27] we show the comparison between the CNN prediction and the FE prediction for the non-linear elasticity case. We observe that the maximum values are very close with a relative error between the maximum values less than 4% and also that the stress distribution is correctly predicted. We also see the same comparison for another structure in [Fig 5.28] where we can observe a similar behaviour.

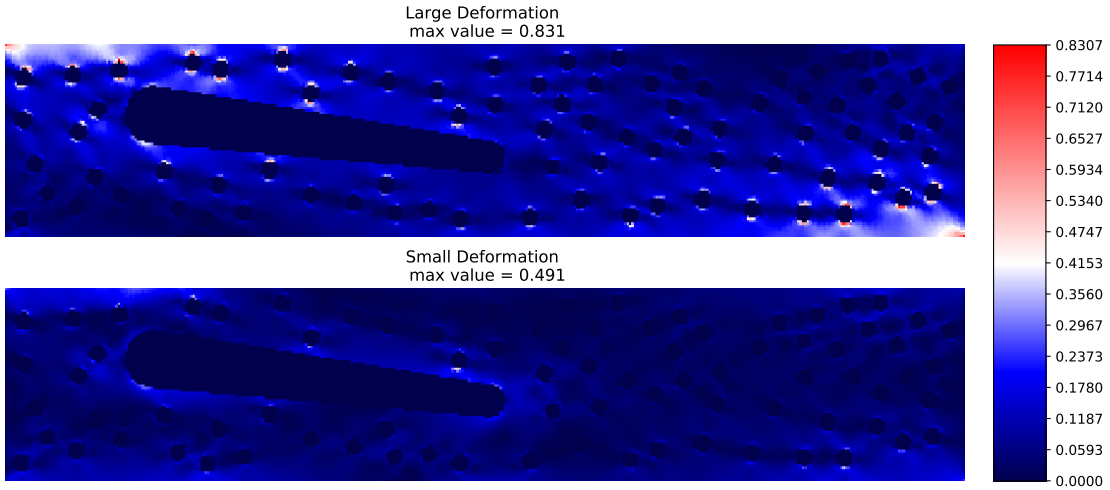
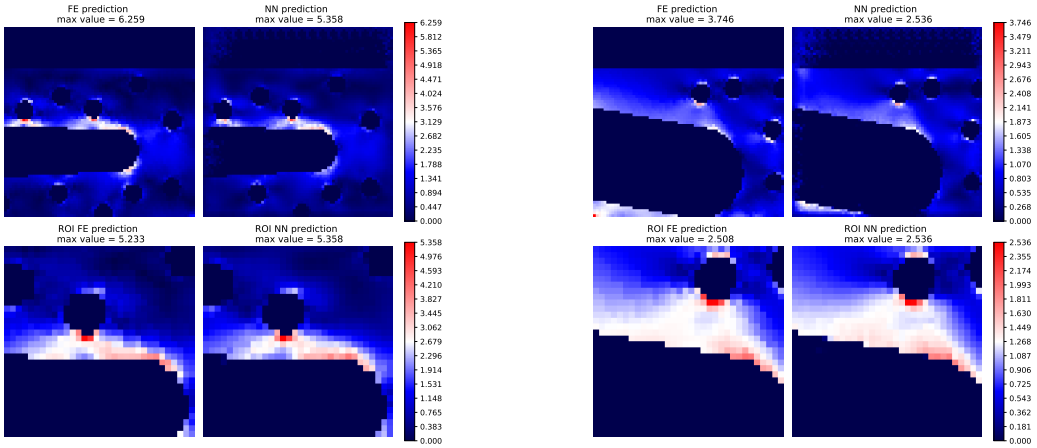
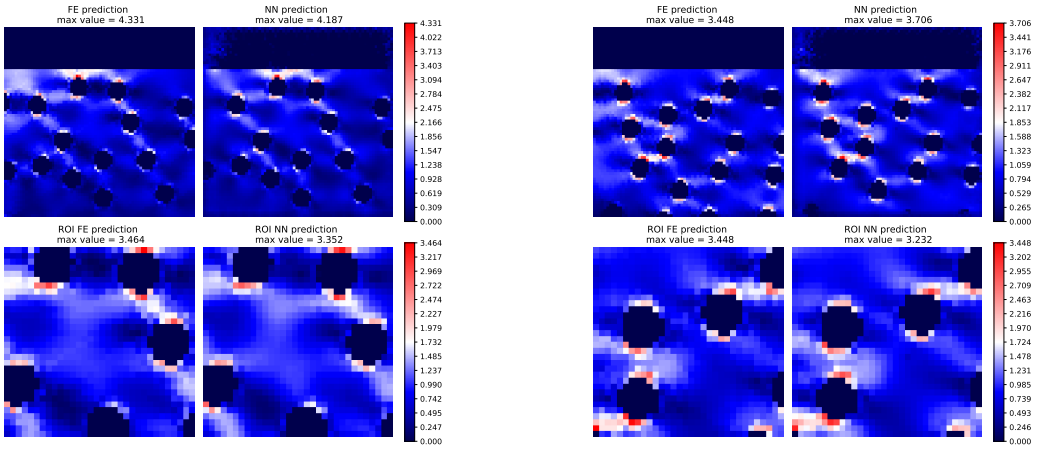


Figure 5.25: Comparison between the Tresca stress field computed by FEA and converted into an image for the nonlinear elasticity case, on top, and the Tresca stress field computed by FEA and converted into an image for the linear elasticity case, on the bottom, for the structure described in [Fig 5.23].



(a) (b)



(c) (d)

Figure 5.26: Comparison between CNN and FE prediction in 4 patches. In each of the 4 images the first row from left to right corresponds to the scaled Tresca stress field in the patch computed by FEA and converted into an image and the CNN prediction in the patch. The second row from left to right is the scaled Tresca stress field in the ROI computed by FEA and converted into an image and the CNN prediction in the ROI.

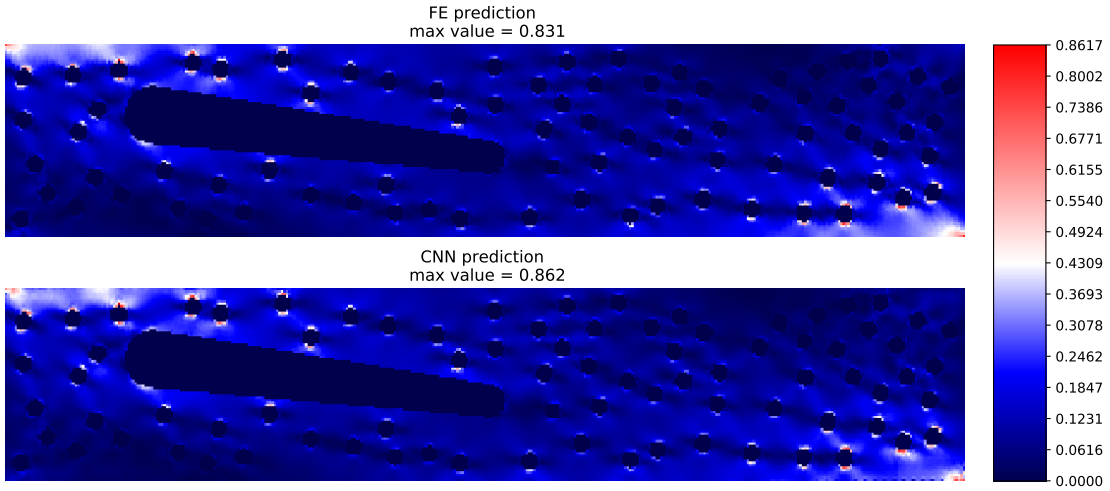


Figure 5.27: Comparison between the Tresca stress field computed by FEA and converted into an image, on the top, and an image reconstructed using the CNN predictions, on the bottom, for the structure described in [Fig 5.23].

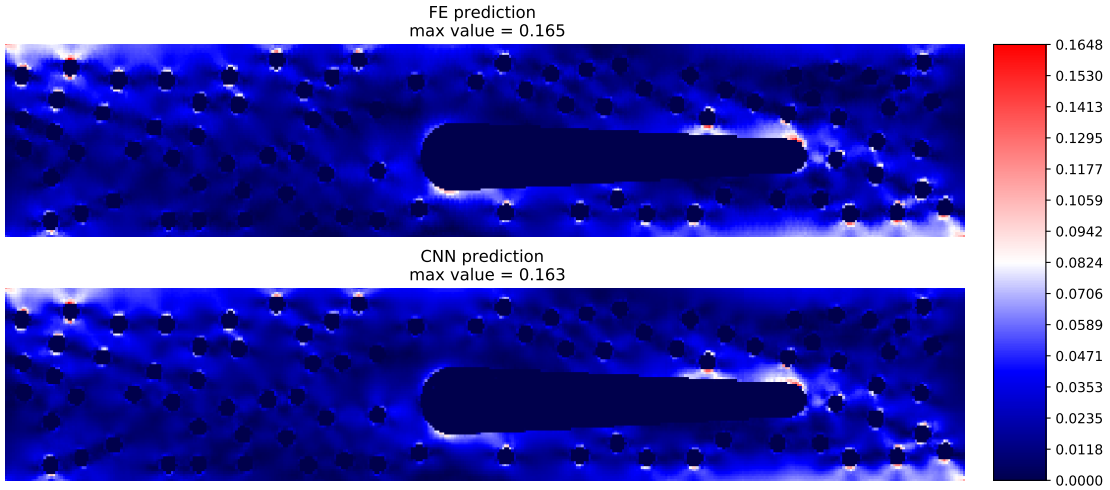


Figure 5.28: Comparison between the Tresca stress field computed by FEA and converted into an image, on the top, and an image reconstructed using the CNN predictions, on the bottom.

5.3.3 Selective learning

As already discussed in [section 5.3.1] not all the training data provide useful information for the network to learn. As the size of the training dataset increases, the training time increases proportionally, while the gain for the network is disproportionally small. Consequently, in this section we investigate the idea of Selective Learning to reduce the labelled data requirements for training the BNN by selecting only the data that contain new information for the network to learn. We need an initial dataset with labelled data so we can initially train the BNN, a larger dataset with only unlabelled data and finally a validation set with labelled data. The principles of this framework are described below.

1. We use an acquisition function to select small batches from the unlabelled dataset
2. We label the selected data points and "move" them to the training set
3. We train the BNN with the new training dataset
4. We measure the accuracy of the BNN using the validation set
5. We repeat the same process until the accuracy converges or we label the entire unlabelled set.

The data that will be used in this section come from the linear elasticity problem [5.3.1]. We designed a small experiment to validate our approach, inspired by [Gal et al., 2017]. Here we make a comparison between a network trained using the maximum uncertainty acquisition function, choosing first the patches with higher uncertainty, and a second one trained using the random acquisition function, that chooses patches randomly. For the random selection approach, we repeated the experiments 5 times and presented the mean and the 95% confidence interval. We used the following setup: 2,500 patches for the initial set, 2,500 patches as the unlabelled set and 11,000 patches as validation set. We trained each network for 50 epochs, we performed 50 forward passes for the uncertainty estimation and we added 500 patches in the labelled set at each iteration. The number of data added in each iteration is called query rate, so in this case the query rate = 500. The accuracy is calculated from the mean prediction of the network. The results can be found in [Fig 5.29]. We observe that the results produced by using the maximum uncertainty acquisition function, to select the training samples, consistently present higher accuracy. Specifically, with this unlabelled data

set we can reach an accuracy of about 75%. This can be achieved using 1,500 patches with the maximum uncertainty acquisition function but requires all the 2,500 patches if we choose them randomly. This means that we reduced the labelled data requirement by 40%.

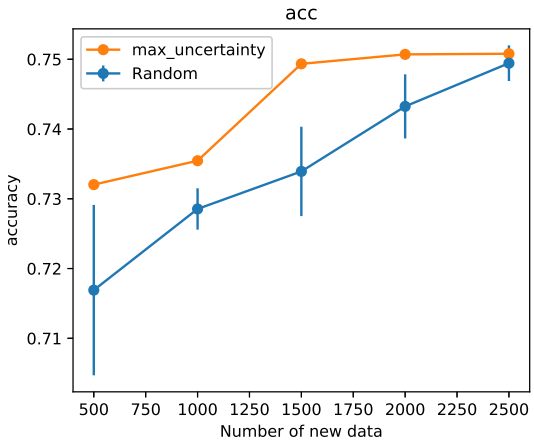


Figure 5.29: Results of Selective Learning for an initial set of 2,500 patches. 50 epochs per training, 50 forward passes for the uncertainty quantification and 500 added data in each iteration. The orange line corresponds to the maximum uncertainty acquisition function and the blue to the random acquisition function. For the random acquisition function we repeated the experiment five times and reported the mean values and the 95% confidence interval.

Now we will use a larger unlabelled dataset consisting of 10,000 patches. We compare again the maximum uncertainty acquisition function and a random acquisition function. The initial training set has 5,000 patches. We train for 150 epochs every network. We perform 100 forward passes for the uncertainty quantification and we label 2,000 unlabelled patches at each iteration (we calculate the microscale stress field for them), query rate = 2,000. The results can be found in [Fig 5.30]. The accuracy increases faster for the maximum uncertainty acquisition function and also the loss function is decreasing faster until it reaches 6,000 new patches. At this point the accuracy practically stops increasing and the loss gradually approaches the same value as with the random acquisition function. Using the maximum uncertainty acquisition function we can reach the maximum accuracy using 6,000 patches while we need all the 10,000 patches when randomly choosing new data. Again we have a decrease of 40% in the labelled data requirement.

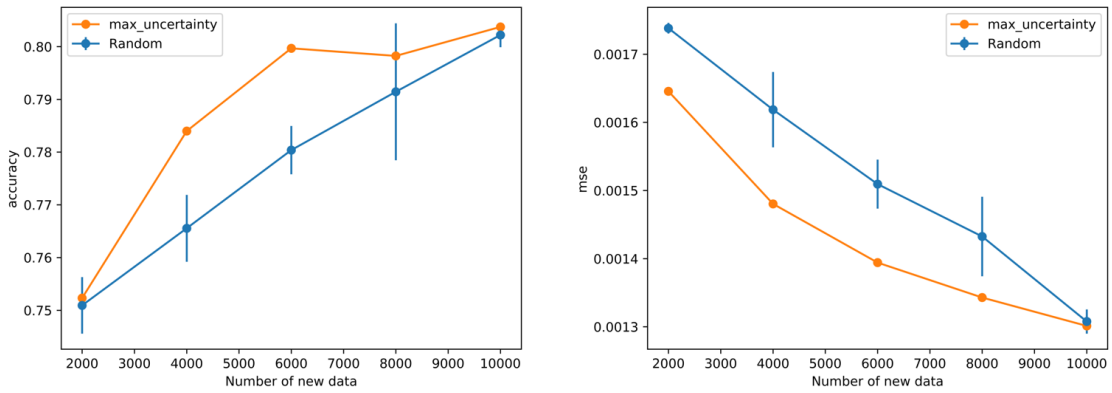


Figure 5.30: Demonstration of the selective learning framework for an unlabelled set of size 10,000 patches. On the left a diagram depicting the accuracy and on the right a diagram depicting the loss. The orange line corresponds to the maximum uncertainty acquisition function and the blue to the random acquisition function. For the random acquisition function we repeated the experiment five times and reported the mean values and the 95% confidence interval.

This time we want to perform a similar experiment but we are interested in examining the effect of query rate on the results. Specifically, we will use an initial set of 5,000 patches and we will perform Selective Learning on an unlabelled dataset of 4,000 patches. We will repeat the experiment 3 times, with query rates 500, 1,000 and 2,000. A similar experiment was conducted by [Islam, 2016], where he concluded that using very small query rates results in sub-optimal performance, higher simulation times and noisy behaviour. This can be attributed to the following two reasons. Firstly, adding only a few patches compared to the size of the initial dataset might result in overfitting and secondly, these patches might get smoothed out in the loss function. The simulation time increases because the network needs to be retrained a considerable number of times. On the other hand using too large query rates also results in worse results because the weights of the network are not updated frequently enough so new information is rarely incorporated in the network and we end up again labelling and training on patches that do not contain new information. The results of our experiment can be found at [Fig 5.31]. We have reached the same conclusions. When query rate is 1,000 we have the optimal behaviour, when we double it we observe slower convergence and when we use a small query rate we observe noisy sub-optimal behaviour.

After validating the Selective Learning framework we will now use it without the

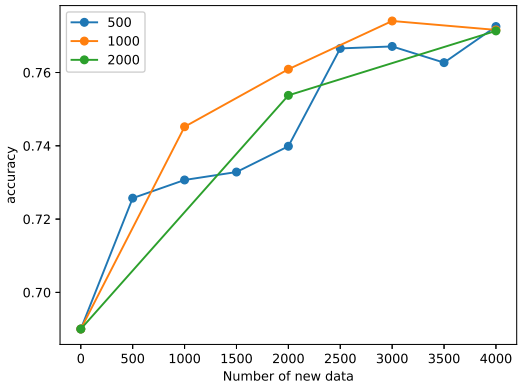


Figure 5.31: Selective learning with query rates of different sizes. The green line corresponds to a large query rate (of size 2,000 patches), the orange line to a medium query rate (of size 1,000 patches) and the blue line to a small query rate (of size 500 patches). The small and larger query rates result in sub-optimal behaviour and specifically small query rates result in noisy results.

random acquisition function as baseline. We will use all the 30,000 available data to train the network. As an initial set we will use again 5,000 patches. We will query 5,000 unlabelled patches at each iteration chosen by the maximum uncertainty acquisition function. We will train for 300 epochs and perform 100 forward passes for the uncertainty quantification. The results can be found in [Fig 5.32]. It is clear that the accuracy is not improving after the third iteration, 15,000 patches, but we continued labelling points only for demonstration reasons. The mean squared error decreases for the first 3 iterations and then stops decreasing as well. In this example we could reach the maximum accuracy using 15,000 out of the 30,000 patches, so we managed to reduce the labelled data requirements by 50%.

5.3.4 Out of distribution study

Lastly, we test the BNN in data outside of the training set. One way to realise this study is to keep the same microscale features but create new distributions of them. That could be done using the "1 Ellipse Dataset" from section [5.3.1] and drawing patches from the "3 Ellipses Dataset" from section [5.3.1], to obtain out-of-distribution samples. Instead, we choose to completely change the microscale features as we believe that this will be more challenging for the network to predict.

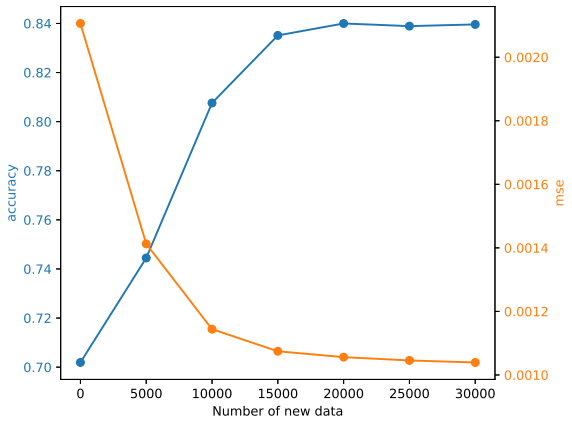
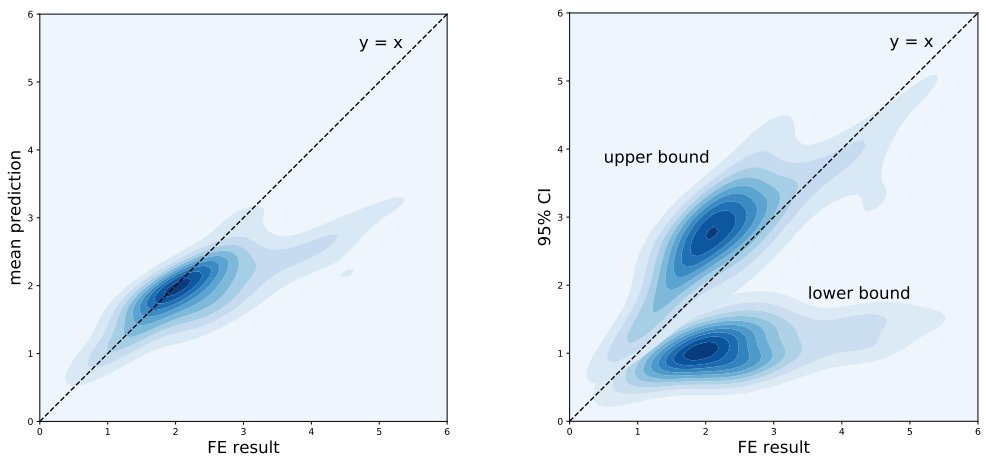


Figure 5.32: Accuracy and mean squared error plots with respect to train data chosen by a maximum uncertainty acquisition function. The blue line corresponds to the accuracy and the orange one to the MSE.

In this study we will use ellipses as micro features to get out of distribution samples. Neural Networks are notoriously bad at extrapolating. What we are hoping for is that the BNN will understand that the ellipses are not in the training dataset and will assign high variance to most of the patches.

We solve the same linear elastic problem we solved in the linear elasticity section [5.3.1] with the same BCs. We created 500 patches and made a prediction with the BNN from [5.3.3]. The results can be found in [Fig 5.33]. From the first plot [Fig 5.33a] we can see that the mean prediction from the BNN for the maximum values in the patch is not close to the real maximum value for a substantial percentage of the data, accuracy $\approx 50\%$, but is not unreasonable. On the other hand, the second plot [Fig 5.33b] shows that in most cases, $\approx 80\%$, the true maximum value is indeed inside the 95% CI. Even more encouraging is the fact that higher uncertainty corresponds to higher error as can be seen from [Fig 5.34]. This also implies that selective learning is very promising in this case.

We can also see examples of predictions in 6 patches of this new dataset. In [Fig 5.35a, 5.35b] we can see 2 examples of cases where the error in maximum values is relatively high and even though the 95% CIs are very broad they fail to contain the real value. In [Fig 5.35c, 5.35d] we can see 2 examples of cases where the error is high but inside the 95% CI. Lastly in [Fig 5.35e, 5.35f] we can see 2 examples where the mean prediction of the BNN is very close to the real value. Some error is present in other areas of the patch but this error is captured by the uncertainty of the BNN.



(a) (b)

Figure 5.33: In these 2 figures we see point densities where darker colours correspond to higher point density. On the left a diagram (a) showing the relationship between NNs' mean prediction and FE results for the maximum value in the ROI. We can observe that the maximum value is underestimated in a lot of patches from the BNN. On the right (b) a diagram showing the upper and lower 95% CIs. We can observe that in most cases the real maximum value is inside the 95% CIs of the prediction.

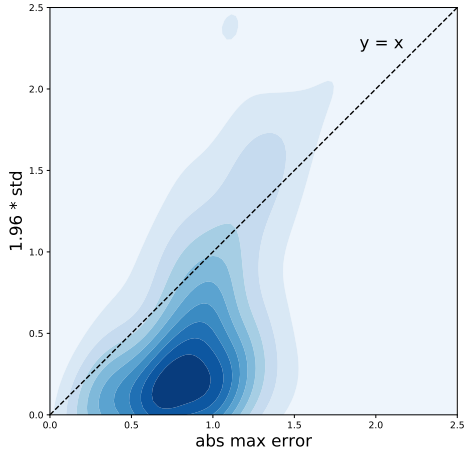


Figure 5.34: A diagram where the x axis is the absolute error between the real maximum value in the ROI and the predicted one and the y axis is 1.96 $\times$ the standard deviation. We can observe a correlation between high uncertainty and high error.

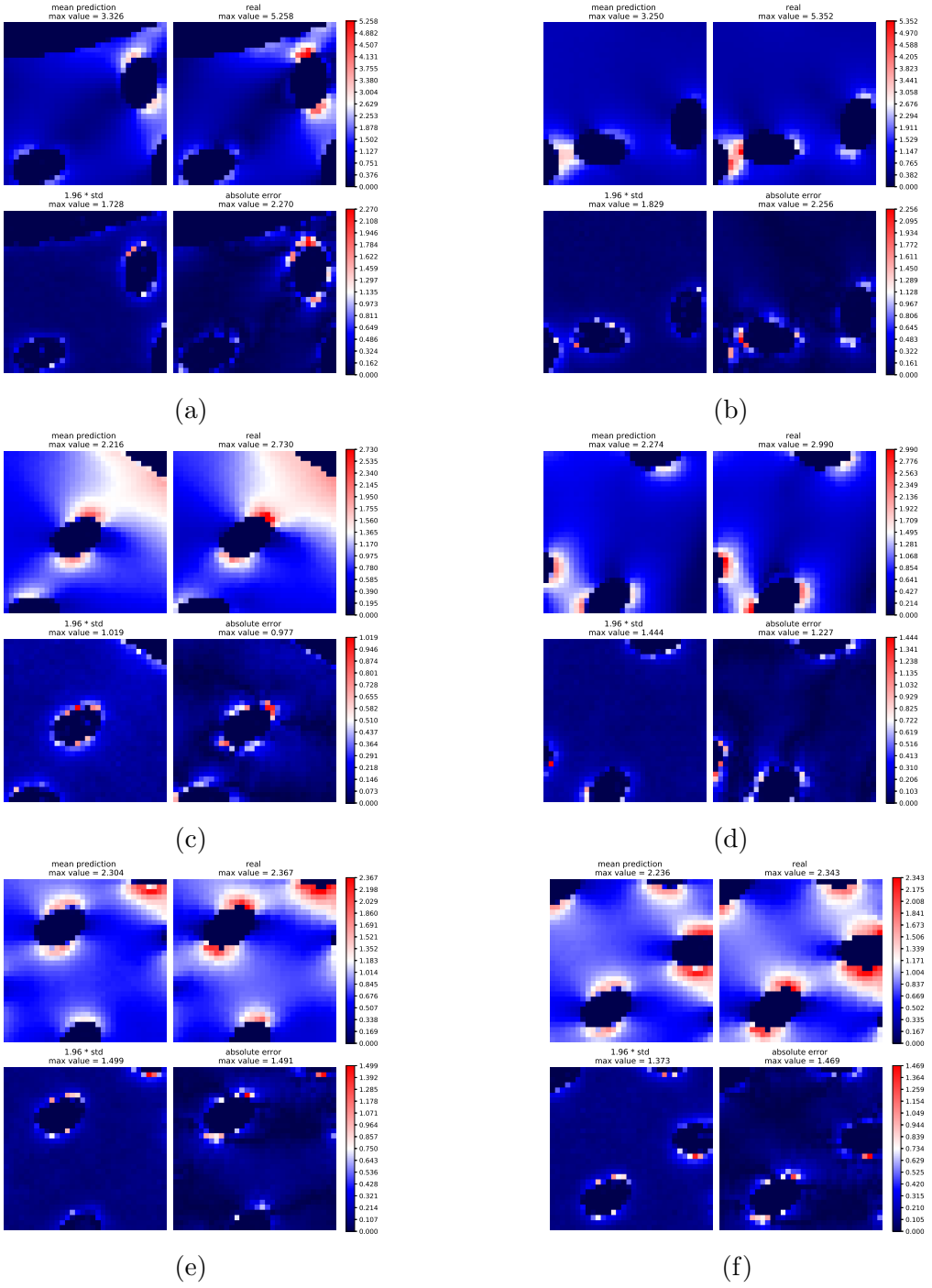


Figure 5.35: BNN predictions on out of distribution examples. All images correspond to the ROI of the patches. In all of the 6 images the first row corresponds to the NN mean prediction on the left and to the scaled Tresca stress field computed by FEA and converted into an image on the right. The second row corresponds to the NN uncertainty, expressed as 1.96 $\times$ standard deviation, on the left and to the absolute error between the NN mean prediction and the FE results on the right.

5.4 Comparison to homogenisation

In elasticity, homogenisation works well if the size of the microscale features is much smaller than the size of macroscale stress concentrators. This is clearly not the case in the examples proposed in this thesis. Conversely, our method is only useful in cases where scales are not well separated, as it will be bettered by homogenisation-based approaches when they are applicable.

We propose to illustrate the qualitative point made above. To this end, we train a CNN that takes as input the average stress field over the patch and not the entire stress field, hence mimicking the effect of spatial averaging commonly found in homogenisation schemes. We call this network a homogenisation CNN. We compare this CNN with the CNN we proposed earlier in this chapter. Our goal is to show that our CNN outperforms the homogenisation CNN and specifically that this happens in patches where there are interactions between microscale and macroscale features. We trained the 2 CNNs with the same architecture and training dataset, 27,000 training data and 3,000 validation data. In [Fig 5.36] we see that the first CNN achieves higher accuracy, specifically for the 10% threshold our CNN achieves 79% accuracy while the homogenisation CNN 42%. Additionally, in Fig [Fig 5.37] we compare the prediction of the 2 CNNs in a patch where the macro stress field is not constant. We see that in contrast to our CNN the homogenisation CNN fails to predict the correct stress field. We can also see that the higher absolute error between the homogenisation CNN and both our CNN and the FE solution is close to the ellipse where the macro stress field varies faster. Lastly, in Fig [Fig 5.38] we compare the prediction of the 2 CNNs in a patch where the macro stress field is constant. We see that both CNNs manage to predict the correct stress field.

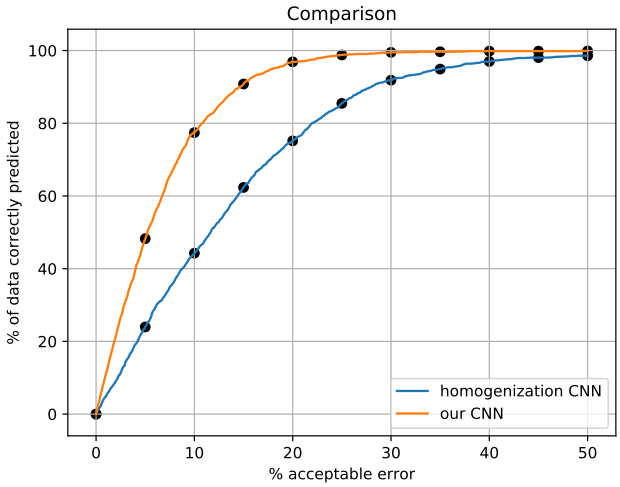


Figure 5.36: Comparison between 2 CNNs trained with the same architecture and dataset but one takes as input the average stress in the patch and the other the entire stress field. Blue line corresponds to the CNN with the average stress as input and the orange line to the CNN trained with the full stress as input. The x-axis of the diagram corresponds to the threshold level used to define the accuracy and the y-axis to the accuracy.

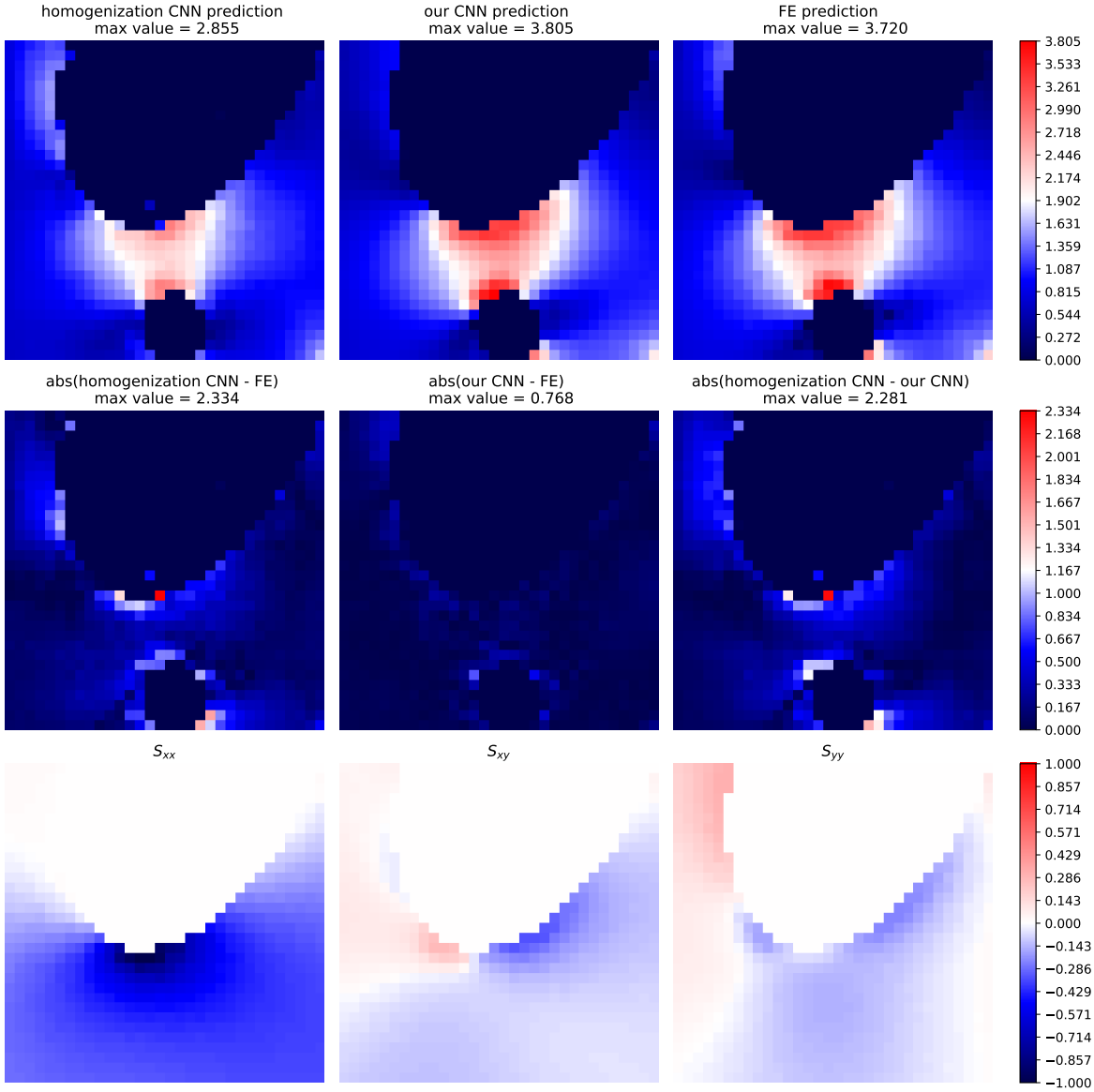


Figure 5.37: A figure where we compare the homogenisation CNN to our CNN for a patch where the macro stress field is not constant. In the first row we see from left to right, the homogenisation CNN prediction, our CNN prediction and the scaled Tresca stress field computed by FEA and converted into an image. We observe that the homogenisation CNN fails to predict the correct stress distribution in contrast to our CNN. In the second row from left to right we see the absolute error between the homogenisation CNN and the FE results, our CNN and the FE results, and the homogenisation CNN and our CNN. In the last row from left to right we see the xx, xy and yy components of the macro stress tensor.

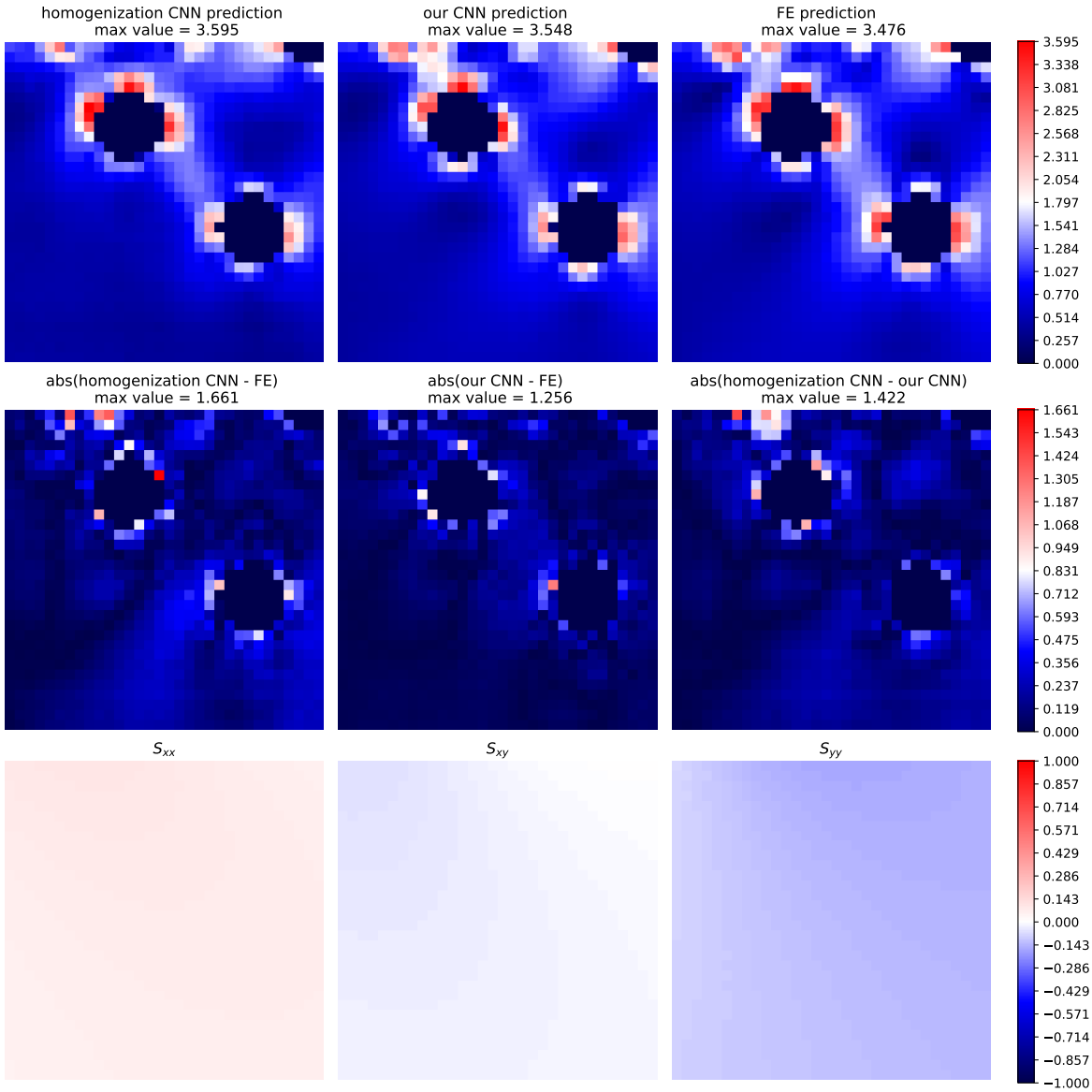


Figure 5.38: A figure where we compare the homogenisation CNN to our CNN for a patch where the macro stress field is constant. In the first row we see from left to right, the homogenisation CNN prediction, our CNN prediction and the scaled Tresca stress field computed by FEA and converted into an image. We observe that both the homogenisation CNN and our CNN manage to predict the correct stress distribution. In the second row from left to right we see the absolute error between the homogenisation CNN and the FE results, our CNN and the FE results, and the homogenisation CNN and our CNN. In the last row from left to right we see the xx, xy and yy components of the macro stress tensor.

5.5 Assumptions and limitations

In this section we aim to remind the reader of the assumptions that we made and also highlight the limitations of the methodology that we developed.

Firstly, we assumed that the local macroscale fields that we use as input to our CNN are sufficient to predict all the microscale features, for all the structural problems over which training is performed. This is an assumption of locality that cannot in general be used in the context of non-diffusive problems (e.g. wave propagation, crack propagation).

A second limitation of the methodology is its rather poor extrapolation ability. Indeed, when the CNN that was trained with disks as microscale features was used to make predictions on data with elliptically-shaped pores as microscale features, the accuracy decreased substantially. However, the uncertainty interval remained reasonably accurate and could be used to indicate that the requested predictions are too far from the training set, for instance in a selective learning framework. Additionally, when the CNN that was trained on the one-ellipse dataset was used to make predictions on the three-ellipses dataset, the accuracy also decreased, albeit not as sharply as in the former case. To summarise, the method does not generalise well outside the set of examples (microscale or macroscale) of the problem over which training was performed, which is unsurprising. A corollary to the previous statement is that it is necessary to restrict training to relatively narrow families of boundary value problems, as the space of heterogeneous structures with randomly distributed microscopic components is arbitrarily vast.

Lastly, many problems arising in computational mechanics are history dependent, for instance elastoplasticity or viscoelasticity. Unfortunately, our current multiscale CNN architecture is not able to tackle all history dependent problems in general, but it could be used for a limited set of problems where one-to-one mapping between the macro stress and the microscale correction exists. For instance, in [Rocha et al., 2021], the authors replaced constitutive relations in multiscale plasticity by instantaneous Gaussian Processes, the strain history being discarded. If we want to address a broader set of history dependent problems, we could add extra channels to the input that would correspond to snapshots of the stress distribution from the past [Yan et al., 2020]. Another common approach for time series prediction is the Long Short-term Memory Networks. Recently [Wang et al., 2021] developed a convolutional-aided bidirectional Long Short-Term Memory network to predict the sequence of maximum stress until material failure.

This architecture combines CNNs and LSTMs and leverages the advantages of both.

5.6 Conclusion and perspectives of this chapter

The goal of this chapter was to develop a CNN based multiscale surrogate modelling technique that can be used to perform fast stress predictions in 2D structures exhibiting spatially random microscopic features, without prior assumption of scale separability, and without prior parametrisation of the multiscale problem.

We have shown that the CNN is able to predict how macroscale solution fields should be corrected by taking into account the existence of microscale pores, interacting arbitrarily with one another and with the boundary of the computational domain. The framework is designed in such a way that it does not require any knowledge of the PDE system to generate microscale corrections, and therefore is not intrusive. Moreover, the methodology is not a priori limited to ad hoc parametrisations of the multiscale boundary value problem, as it treats the geometry of the microscopic patterns as arbitrarily large images. Incidentally, the method is Bayesian and provides credible intervals for the microscale field predictions.

Experiments with data from linear elastic simulations with a variety of macroscale structures, of the same family, with randomly distributed circular pores under different boundary conditions showed good performance in terms of mean values and uncertainty intervals. This suggests that the method generalises well in new realisations of known macroscale structures and microscale distributions. We also proved that the method readily extends to nonlinear elasticity.

Lastly, we investigated two other features of our framework. We addressed the problem of limited labelled data using mechanically consistent rotations as data augmentation technique. Furthermore, to reduce the large computational cost associated with the creation of labelled data (i.e. multiscale FEA simulations) we used selective learning to choose and label only the data that contains new information for the network.

Chapter 6

Graph Neural Networks for the prediction of stress in 3D porous structures

6.1 Introduction

Stress prediction in porous materials and structures is challenging due to the high computational cost associated with direct numerical simulations. In the previous chapter (Chapter [5]), we proposed Convolutional Neural Network (CNN) based architectures as FE surrogate models to approximate the solution of 2D multiscale simulations. CNN methodologies are usually limited to 2D problems due to the high computational cost of 3D voxel-based CNNs. In this chapter, we propose a novel geometric learning approach based on Graph Neural Networks (GNNs) that efficiently deals with 3D problems by performing convolutions over 2D surfaces only. The purpose of this chapter is to show that

- A GNN can be used to automatically add local fine scale stress corrections to an inexpensively computed coarse stress prediction in the porous structure of interest. In contrast to the relevant literature, the examined structures have multiple fine scale features, intersecting with each other, with the macroscale features and with the boundaries of the structures.
- The Bayes by Backprop method can be used to convert the deterministic GNN to a probabilistic one. In contrast to the GNNs found in the relevant literature,

this probabilistic GNN generates densities of stress fields, from which credible intervals can be extracted. This increases the confidence in the prediction which is crucial in realistic engineering applications.

- The uncertainty information extracted from the GNN can be utilised by an online stress correction technique, namely Ensemble Kalman method, to update the output distribution of the Bayesian GNN so that it respects the appropriate physical laws. This methodology alleviates the effect of undesirable biases observed in the outputs of the uncorrected GNN, and improves the accuracy of the predictions in general.

6.2 Geometric learning for multiscale stress analysis

In this section we outline the assumptions utilised to apply the proposed geometric learning framework to multiscale stress analysis problems. Additionally, we analyse the GNN we use to reproduce the FE results of the 3D problem. Lastly, we discuss the input and output of the network as well as its architecture.

6.2.1 Assumptions and justifications

As can be seen from the literature review, researchers can take advantage of the flexibility that GNNs offer, to reduce the computational cost by modelling their system in more meaningful ways that stem from the physical understanding of the underlying problem, as for example [Vlassis et al., 2020] who modelled the behaviour of a polycrystal by considering every crystal as a node of the graph instead of operating on the volume mesh. Following this paradigm, for reasons we explain in the next paragraphs, we choose to work only on the surface mesh of the porous medium and not the volume mesh.

First of all, we try to evaluate to what degree the choice to work only on the surface mesh is going to affect the maximum stress in the structure, which is the quantity of interest. We conduct an experiment where we perform 100 FE simulations with 100 realisations of the dogbone structure. For the boundary conditions we apply a displacement along the x and $-x$ direction of the same magnitude but with opposite direction. Afterwards, we extract from the volume mesh the surface mesh with the

stress tensors encoded on the nodes of the surface mesh (see Appendix A). Lastly, we calculate the maximum Von Mises stress both on the volume and surface mesh and we compare the two. From [Fig 6.1] we can conclude that indeed the maximum volume stress is the same as the maximum surface stress. We do not claim that this would be the case for every possible distribution of pores, for every sample geometry and for every macroscale loading condition but we will make this assumption for computational efficiency reasons.

Additionally, we hypothesise that the surface mesh is a comprehensive representation of the geometry of the specimen. Lastly, we assume that the macroscale stress over the surface mesh is sufficient to inform the GNN of the macroscopic stress state over the patch. We prove these 2 hypotheses numerically in the context of this thesis.

Consequently, we choose to work only on the surface mesh since we are primarily interested in predicting the maximum stress. This greatly reduces the memory requirements and the training time.

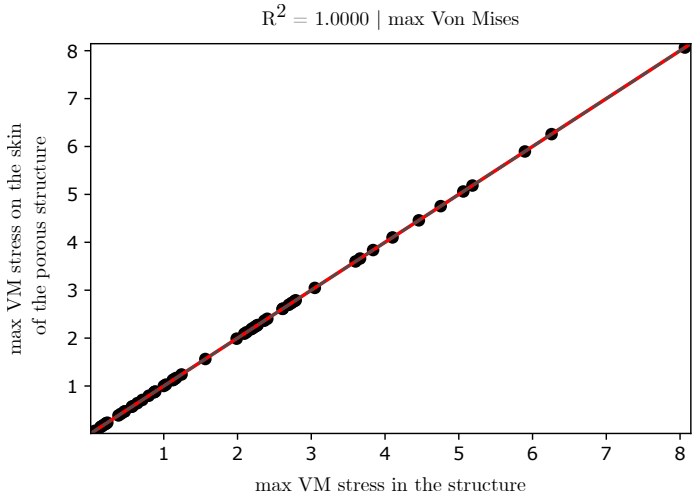


Figure 6.1: The x axis corresponds to the maximum Von Mises stress on the volume mesh and the y axis to the maximum Von Mises stress on the surface mesh for each of the 100 performed simulations. The red line is the $y = x$ line. We can observe that all the points lay on this line and thus the maximum volume and surface stress values are the same.

6.2.2 Graph construction

In our attempt to perform geometric learning on the surface mesh of a dogbone structure we consider a graph G that can be described by a set of nodes $\boldsymbol{V} : \{V_1, V_2, ...V_N\}$ where N is the number of nodes of G and a set of edges $\boldsymbol{E} : \{E_1, E_2, ...E_M\}$ where M is the number of edges of G.

The nodes of the graph $\boldsymbol{V}$ coincide with the nodes of the surface mesh, since their position is optimised by the mesh generation software to best describe the geometry of the structure. The edges of the graph are used to pass messages between the nodes and they define the connectivity of the graph. Specifically, the edges of the graph should create connections between interacting areas of the surface mesh, that are not necessarily connected. Consequently, the edges of the graph cannot coincide with the edges of the surface mesh, since the surface mesh has disconnected areas. This means that it is possible for areas of the surface mesh that are close to each other to not share an edge, for example two interacting but non intersecting defects. This does not allow passing of information between the two and thus their interaction cannot be modelled. To overcome this problem a different approach to create the neighbourhood of a point needs to be followed. Below, we define 2 different neighbourhoods [Fig 6.2].

- Geodesic: Given a node (that we refer to as central node), the Geodesic neighbourhood, or 1-ring neighbourhood, includes the nodes that share an edge with this central node. This is equivalent to the neighbourhood dictated by the surface mesh connectivity.

- Euclidean: The Euclidean neighbourhood in this work is defined using the radius neighbours. Every node V_i, is connected with every node that falls into a sphere with a predefined radius r and with centre V_i. The predefined radius r that is used for this work is 4 times the radius of the spherical voids as suggested in [Krokos et al., 2021]. In practice that results in a very high number of neighbours and thus into a very expensive computational problem. To tackle this problem we need to restrict the number of neighbours for each node by setting a threshold and randomly choosing neighbours so that the total number of neighbours is below the threshold. In [Appendix B.5], we discuss the most appropriate value for this threshold.

To overcome the problem of message passing between disconnected areas of the surface mesh, in this work we use either Euclidean or a combination of Euclidean and

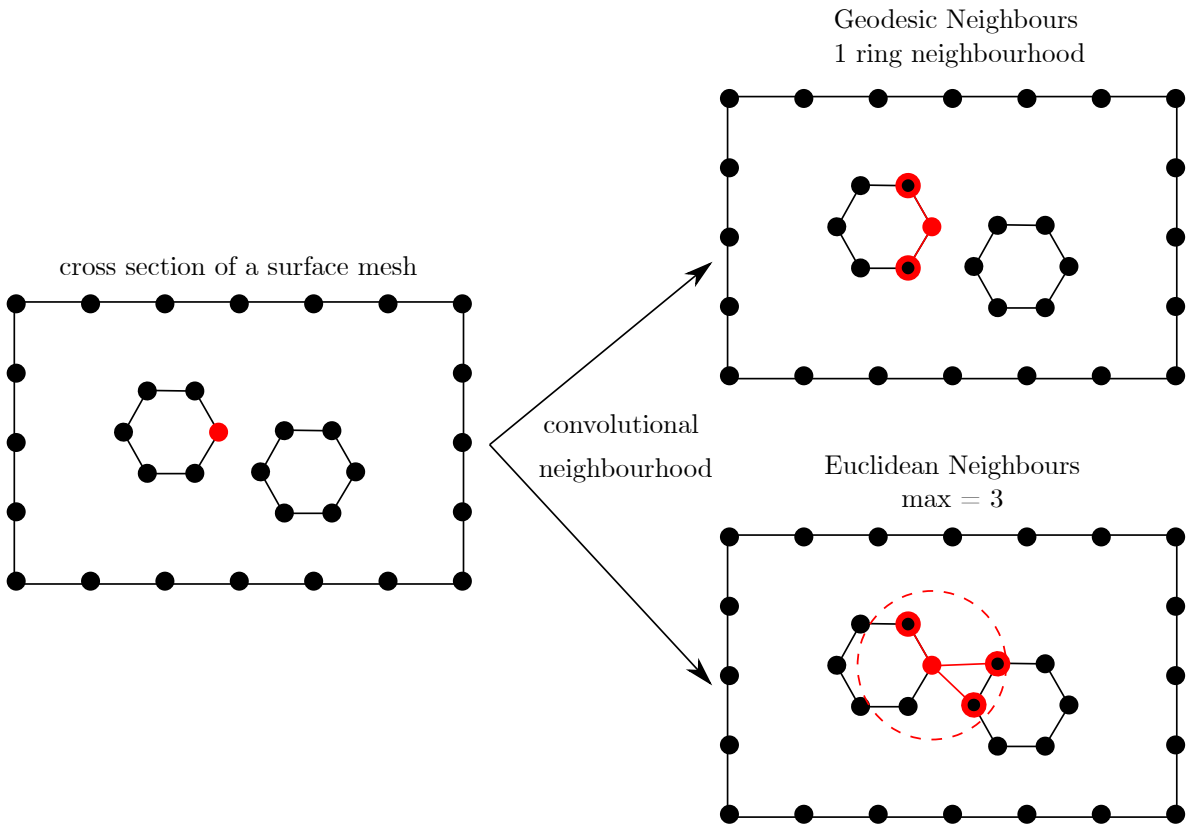


Figure 6.2: On the left we show a cross section of the surface mesh depicting a rectangle with 2 spherical features. With red with see the node that is considered the central node for the convolutional neighbourhood. On the right we can see 2 possible neighbourhoods for the central node. The Geodesic (1-ring) neighbourhood on top and the Euclidean neighbourhood at the bottom with a threshold for the maximum number of neighbours equal to 3. In the Geodesic neighbourhood there is no edge between the two spherical features and thus message passing between the two is not possible.

Geodesic neighbours, as proposed by [Schult et al., 2020], both of which allow the modelling of interactions between non-intersecting parts of the mesh.

The edges of the graph are bidirectional so that the information can be exchanged both ways between two connected nodes. This means that for every edge $E_{k,l}$ passing a message from node V_k to V_l there exists the edge $E_{l,k}$ passing a message from node V_l to V_k.

6.2.3 Input-Output

The framework we use requires the input and output graphs to be isomorphic, meaning they have the same number of nodes, edges, and the same edge connectivity. Consequently, we will use the microscale mesh (mesh with spherical voids) as a graph to represent both microscale and macroscale 3D FE solutions. For the macroscale solution, the FE problem is solved using the macroscale mesh (mesh without spherical voids) and then the solution is interpolated into the microscale mesh.

For the input of the GNN we encode node features, v_i where $i = 1, 2, \dots N$, on the nodes of the graph and edge features, $e_{i,P(i)}$ where $P(i)$ is the neighbourhood of i, on the edges of the graph. The input node features are the independent components of the macroscale stress tensor along with the microscale feature indicator, a single integer indicating if the node corresponds to a microscale feature or not. The input edge features are the relative position between the 2 nodes that the edge is connecting along with the distance between the 2 nodes. This is summarised in [Fig 6.3].

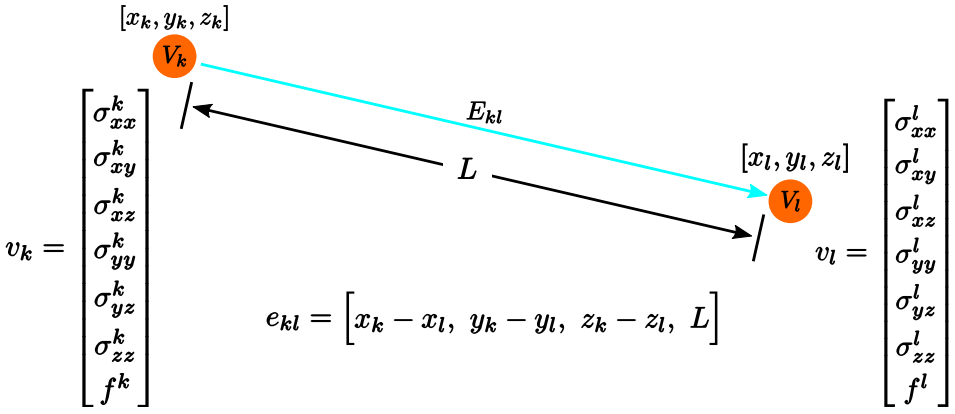


Figure 6.3: Example of input node features, u_k and u_l, and edge features, e_{kl}. With orange we see the nodes and with light blue the directed edge. f is the micro indicator, a single integer that indicates if the node is a macroscale or microscale feature. $\sigma_{i,j}$ corresponds to the i, j component of the macroscale stress tensor. L is the distance between the two nodes.

The output of the graph contains information only on the nodes of the mesh. Specifically, the output node features are the components of the microscale stress tensor.

We sum up in [Fig 6.4] the input and output of the GNN. We have only included the node features since the edge features are calculated as a pre-processing step from the connectivity of the mesh.

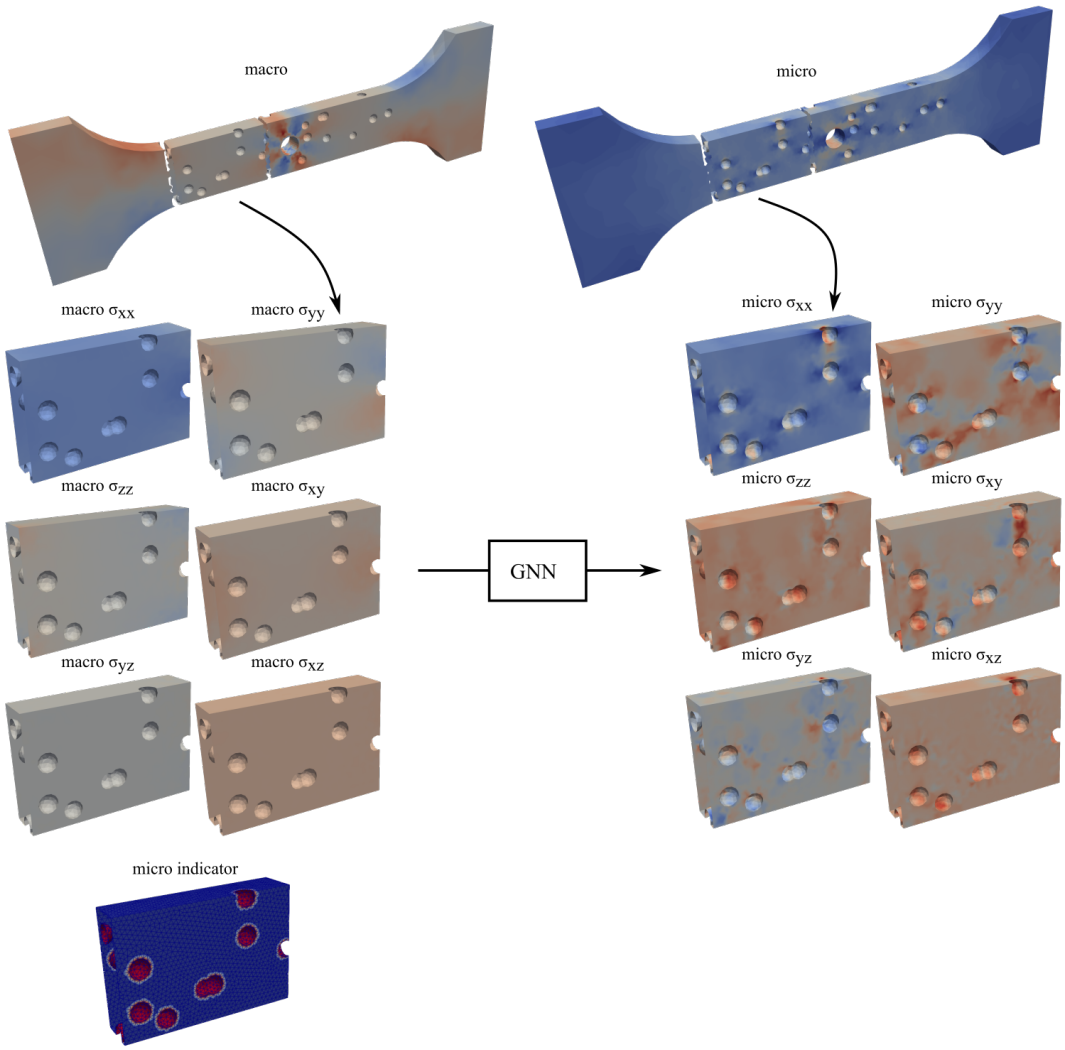


Figure 6.4: In this figure we can see the input and output of the GNN. The input consists of the 6 stress components of the macroscale stress tensor along with the micro indicator, a single integer per node that determines if the node belongs to a microscale or a macroscale geometrical feature. For the patch corresponding to the micro indicator the FE mesh is visible. The output of the GNN is the 6 stress components of the microscale stress tensor. At the top of the figure, we can see the full structures from which the patches are extracted both for the input and the output. We stress that the macro stress is computed without the pores, i.e. stress gradients in the gage section are solely created by the cylindrical hole, before being projected onto the microscale mesh to be provided as input to the GNN.

6.2.4 Loss function

The loss function used for the training of the deterministic networks in this chapter is the Mean Squared Error (MSE) between the microscale stress tensor in the ROI of every patch predicted by the GNN and the one calculated using direct FEA. For the Bayesian networks the ELBO loss is used, as described in equation [2.15].

6.2.5 GNN model

GN Block

The GN block is the basic building block of the GNN and it is used to map an input graph to an isomorphic output graph with updated node and edge features. The node and edge features are jointly passed through an edge update MLP and a node update MLP in a 2-step procedure described in [Battaglia et al., 2018] that sequentially updates the edge features first and the node features later. A sketch of this procedure can be found in [Fig 6.5].

For the edge update step, if we consider an edge $E_{k,l}$ with edge features e_{kl} that sends a message from the sender node V_k with node features v_k to the receiver node V_l with node features v_l then the updated edge feature, e^*_{kl}, is calculated by passing through the edge MLP the concatenation of these features, $e^*_{kl} = \text{MLP}(\text{concatenate}(e_{kl}, v_k, v_l))$. If M is the number of edges of the graph, NNF is the number of node features and NEF is the number of edge features then the input of the edge update MLP is of size $[M \times (2NNF + NEF)]$.

For the node update step, the updated edge features, e^*, are aggregated per node. In this work we use mean aggregation although elementwise summation, maximum or any other symmetric operation can be used. After the aggregation step, we have a single edge feature, e^{**}, corresponding to each node feature v. The updated node features, v^*, are calculated using a node MLP. The input to this node MLP is the concatenation of the original node features with the updated and aggregated edge features corresponding to this node, $v^* = \text{MLP}(\text{concatenate}(v, e^{**}))$. If N is the number of nodes of the graph, NNF is the number of node features and F is the dimensionality of the updated edge features then the input of the node update MLP is of size $[N \times (NNF + F)]$.

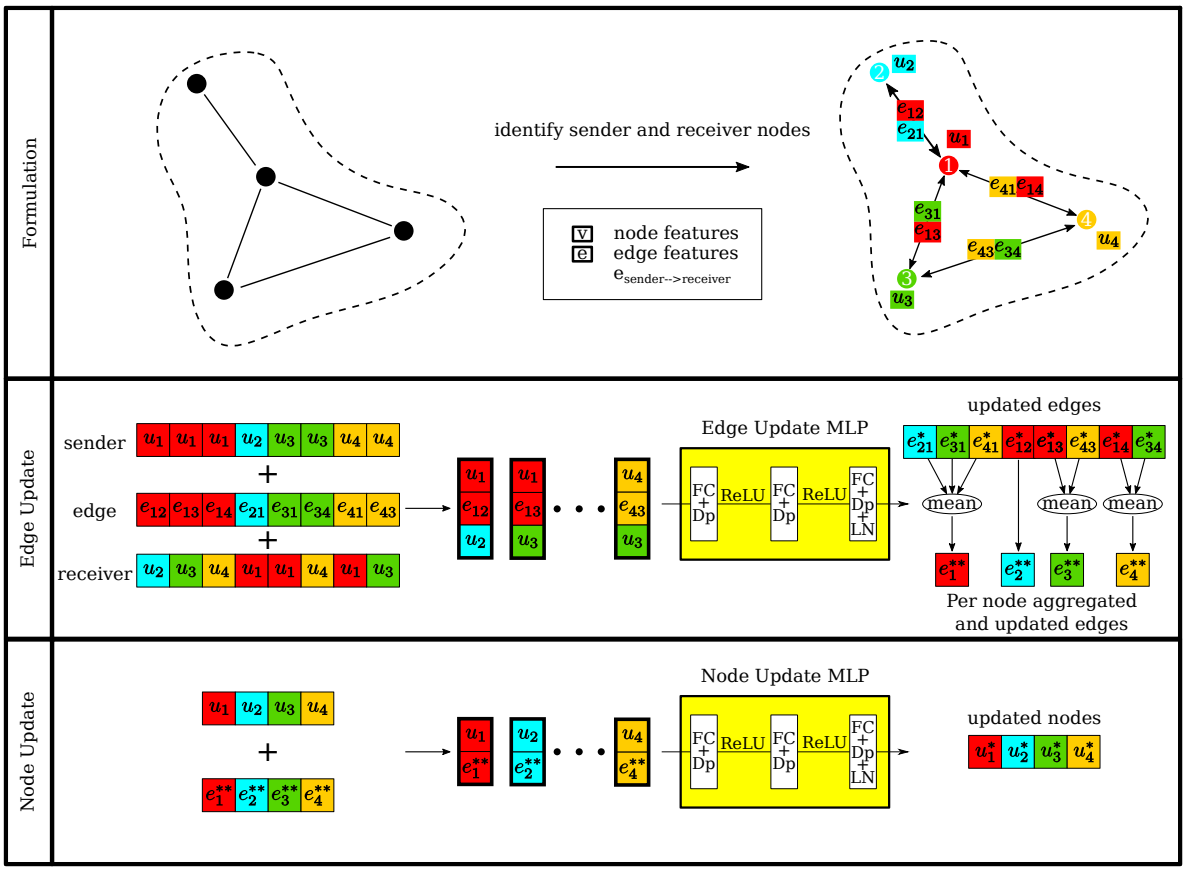


Figure 6.5: Structure of the GN block used in this work. The update of node and edge features happens through a 2-step procedure as described in this figure. FC stands for fully connected layer, Dp for dropout and LN for layer normalisation.

Architecture

In this work we deploy an Encode-Process-Decode GNN as described by [Battaglia et al., 2018]. The encoder encodes the node and edge features in a latent space of higher dimensions. The processor updates these node and edge features and finally the decoder, decodes these features from the latent space to the output space.

- **Encoder**: The node and edge features are independently encoded into a latent space of 128 dimensions using 2 distinct MLPs with the same structure. The MLP has 3 hidden layers. Each hidden layer is followed by a Dropout layer [Hinton et al., 2012] and a ReLU activation function. A Layer Normalisation layer is placed after the output layer, which improves training stability and decreases training time in recurrent NNs [Ba et al., 2016].

- **Processor**: The processor is composed of 5 residual GN Blocks. By using residual blocks the NN itself can choose its depth by skipping the training of a few layers using skip connections. This has been shown to be advantageous in training deep NNs [He et al., 2016; Kim et al., 2016; Zagoruyko and Komodakis, 2016; Lim et al., 2017] and it also applies to GNNs [Sanchez-Gonzalez et al., 2020; Pfaff et al., 2021; Mylonas et al., 2022]. The structure of the MLPs of the Processor is the same as the structure of the encoder MLPs. The processor combines information from edges and nodes and passes messages between nodes that share an edge.

- **Decoder**: The decoder operates only on the node features as the output of the GNN is the microscale stress field on the nodes. A single MLP is used to decode the node features from the latent space to the output space. The decoder MLP is similar to the MLPs used in the encoder and the processor but there is no Layer Normalisation layer applied after the output layer

There is a skip connection that connects the macroscale stress tensor, from the input, with the output of the Decoder. That way the network can learn how the microscale stress field diverges from the macroscale stress field. In [Appendix B.1], we demonstrate that this leads to smoother and faster convergence. A sketch of the GNN can be found below [Fig 6.6].

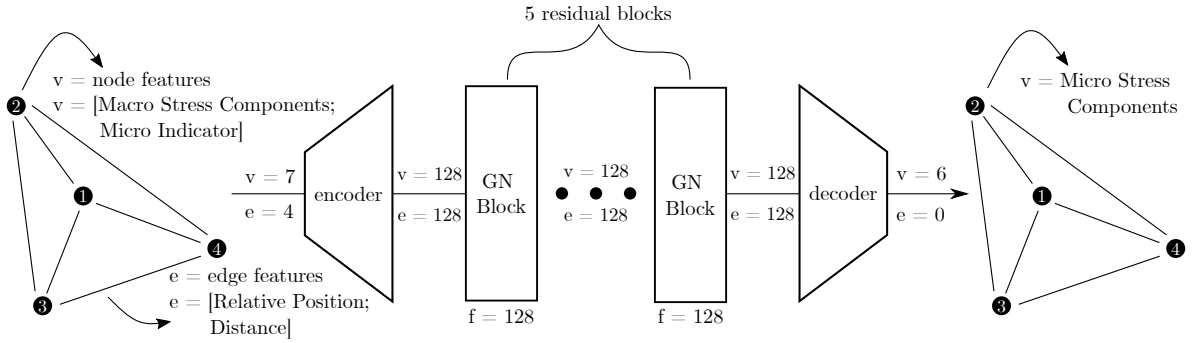


Figure 6.6: Architecture of the GNN

6.3 Physics-based corrections of the NN predictions: enforcing Neumann conditions online via an ensemble Kalman approach

In this section we introduce an online Bayesian method to constrain the prediction of the GNN. This method allows us to improve the network prediction without creating more data nor retraining the network. We impose Neumann conditions (which are homogeneous in all our numerical examples), which are the only equations that may be enforced on the stress field after restricting it to the surface of the porous structure. These conditions are not imposed while training the network, even though the training examples satisfy them up to the FE accuracy.

To apply this method, we require a statistical distribution for the GNN predictions, and thus we use the output of the Bayesian GNN (BGNN). The posterior estimation of the BGNN for the stress field over the surface of the structure will then serve as a prior for the online correction step, which is formulated as a standard Bayesian data assimilation problem. More precisely, the Neumann conditions are applied by considering them as partial information of the stress field, yielding a Bayesian posterior update.

The ensemble Kalman method is used to update the prior probability density of the state of a system, represented by vector $\mathbf{x} \in \mathbb{R}^n$, taking into account noisy and partial observations of that state. In the present context, the state vector corresponds to the microscale stress components predicted by the BGNN at every node, j, of the surface mesh. Those are concatenated in a vector of $6J$ components, where J is the number of surface nodes of the inference mesh where Neumann boundary conditions are to be enforced. The state vector reads as

$$\mathbf{x} = \begin{pmatrix} \hat{\mathbf{x}}^1 & ... & \hat{\mathbf{x}}^{\mathrm{J}} \end{pmatrix}^T \tag{6.1}$$

where

$$\hat{\mathbf{x}}^{\mathrm{j}} = \begin{pmatrix} S^{\mathrm{j}}_{xx} & S^{\mathrm{j}}_{yy} & S^{\mathrm{j}}_{zz} & S^{\mathrm{j}}_{yz} & S^{\mathrm{j}}_{xz} & S^{\mathrm{j}}_{xy} \end{pmatrix} \tag{6.2}$$

The Neumann boundary conditions are treated as linear observations of the state vector, through the formal definition of observation operator $\mathbf{H}$ as

$$\mathbf{H}\mathbf{x} = \begin{pmatrix} \mathbf{S}^1 \cdot \mathbf{n}^1 & ... & \mathbf{S}^{\mathrm{J}} \cdot \mathbf{n}^{\mathrm{J}} \end{pmatrix} \tag{6.3}$$

where $\mathbf{S^j}$ is the $[3 \times 3]$ symmetric microscale stress tensor at node j and $\mathbf{n^j}$ is the $[3 \times 1]$ unit normal vector

$$\mathbf{S^j} = \begin{pmatrix} S^j_{xx} & S^j_{xy} & S^j_{xz} \\ S^j_{xy} & S^j_{yy} & S^j_{yz} \\ S^j_{xz} & S^j_{yz} & S^j_{zz} \end{pmatrix}, \quad \mathbf{n^j} = \begin{pmatrix} n^j_x \\ n^j_y \\ n^j_z \end{pmatrix} \tag{6.4}$$

Following the standard Bayesian approach to data assimilation, random noise is added to the observations so that a Gaussian likelihood is implicitly defined. The data reads as

$$\mathbf{d} = \mathbf{H}\mathbf{x} + \epsilon \qquad \epsilon \sim \mathcal{N}(0, \mathbf{\Sigma}_\epsilon) \tag{6.5}$$

If $\mathbf{x}$ were normally distributed then we would have an explicit formula, through Bayes's rule, for the posterior state vector, $\mathbf{x}^\star$. In that case, the posterior state would be given by

$$\mathbf{x}^\star = \mathbf{x} + \mathbf{G}(\mathbf{d} - \mathbf{H}\mathbf{x}) \tag{6.6}$$

where $\mathbf{G}$ is the so-called Kalman gain defined as $\mathbf{G} = \mathbf{\Sigma}\mathbf{H}^T(\mathbf{H}\mathbf{\Sigma}\mathbf{H}^T + \mathbf{\Sigma}_\epsilon)^{-1}$, $\mathbf{\Sigma}$ being the prior covariance matrix.

Our prior state density, which is the output of the BGNN, is of course non-Gaussian. However, the idea of the Ensemble Kalman method, which is at the heart of the Ensemble Kalman filter, is to approximately sample the posterior density using the above formula, i.e. claiming that the prior (and therefore the posterior density as well as the observations are Gaussian-perturbed linear observations of the state) is Gaussian. This is achieved by taking N samples from the BGNN posterior and creating an ensemble of state vectors

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}^1 & \dots & \mathbf{x}^\mathrm{N} \end{pmatrix} \tag{6.7}$$

The formula for the posterior update can be rewritten in terms of the N samples as

$$\mathbf{X}^\star = \mathbf{X} + \tilde{\mathbf{G}}(\mathbf{D} - \mathbf{H}\mathbf{X}) \tag{6.8}$$

where the perturbed data matrix reads as

$$\mathbf{D} = \begin{pmatrix} \mathbf{d}^1 & \dots & \mathbf{d}^N \end{pmatrix} \tag{6.9}$$

where $\mathbf{d}^i \sim \mathcal{N}(\mathbf{d}, \mathbf{\Sigma}_\epsilon)$, and the approximated Kalman gain matrix, $\tilde{\mathbf{G}}$, is defined by

$$\tilde{\mathbf{G}} = \tilde{\mathbf{\Sigma}}\mathbf{H}^T(\mathbf{H}\tilde{\mathbf{\Sigma}}\mathbf{H}^T + \mathbf{\Sigma}_\epsilon)^{-1} \tag{6.10}$$

In the equations above, the prior covariance matrix used in the standard Gaussian case has been substituted by the ensemble covariance matrix defined as

$$\tilde{\mathbf{\Sigma}} = \frac{\mathbf{X}\mathbf{X}^T}{N-1} \tag{6.11}$$

We note that that the observation matrix $\mathbf{H}$ does not need explicit defining. Only its action needs to be evaluated for every prior sample. For additional (and classical) algorithmic details, the reader may refer to [Appendix C].

To summarize, the online stress correction procedure that we employ here can be described using the following steps. A prior state ensemble is created by drawing N samples from the BGNN posterior. Then, the Kalman update scheme described above transforms the prior empirical distribution into an approximation of the posterior empirical distribution that satisfies the homogeneous Neumann conditions up to a certain tolerance encoded by the spherical covariance matrix $\mathbf{\Sigma}_\epsilon$.

This tolerance may be interpreted as a discretisation error, i.e. the Neumann conditions should not be exactly enforced as (A) the BGNN is trained on the output of finite element code, which delivers stress fields that do not satisfy the Neumann conditions exactly and (B) the normal vector at a vertex is not uniquely defined (we choose here to calculate them for each triangle and then average them at shared points). These are qualitative observations, which did not help us choose the level of noise. In practice, we adjusted the level of noise (i.e. a scalar parameter) manually so that the posterior update is of appropriate quality on a validation set.

6.4 Numerical examples

6.4.1 Numerical example with cubical heterogeneous material

In this section we want to test our methodology using a dataset containing samples from a heterogeneous material. We will demonstrate how we can use our methodology

to

- predict the microscale stress distribution
- predict the maximum microscale Von Mises stress over the patches
- quantify the uncertainty of the prediction
- improve the BGNN prediction using the online correction technique introduced in section [6.3]

FEA set-up and generation of the training dataset

For the purpose of training the model we will work with a synthetic heterogeneous material from which we extract and examine cubical specimens. The size of each specimen is 2 units along each spatial dimension. The specimen has two macroscale elliptical pores, with random position, size and orientation and a distribution of 50 to 100 microscale spherical pores with the same radius, $R = 0.08$ units. The spherical and elliptical pores are generated without intersecting. For the macroscale simulations, input of the BGNN, only the elliptical pores are taken into account and the spheres are ignored. The Young's modulus and the Poisson ratio of the structure are 1 and 0.3 respectively. Two realisations of the cubical specimen are shown in [Fig 6.7].

As already explained in [section 4.5], the input to the GNN is a patch of the geometry, not the entire structure, and the prediction of the GNN happens in a sub region of the patch that we called ROI. The patch and ROI size depend on the radius of interaction of the microscale spherical pores. Following the same logic as in the 2D case we choose a patch size of $[18R \times 18R \times 18R]$ and a ROI size of $[8R \times 8R \times 8R]$, where R is the radius of the spherical voids, as can be seen in [Fig 5.6].

The patches that are extracted never intersect with the exterior boundaries of the FE mesh. Additionally, we apply homogeneous Dirichlet boundary conditions away from the material specimen, as represented in [Fig 6.8], [Eq. 6.12].

$$u = \begin{bmatrix} E_{xx} & E_{xy} & E_{xz} \\ E_{xy} & E_{yy} & E_{yz} \\ E_{xz} & E_{yz} & E_{zz} \end{bmatrix} (X - X_0)^\top \tag{6.12}$$

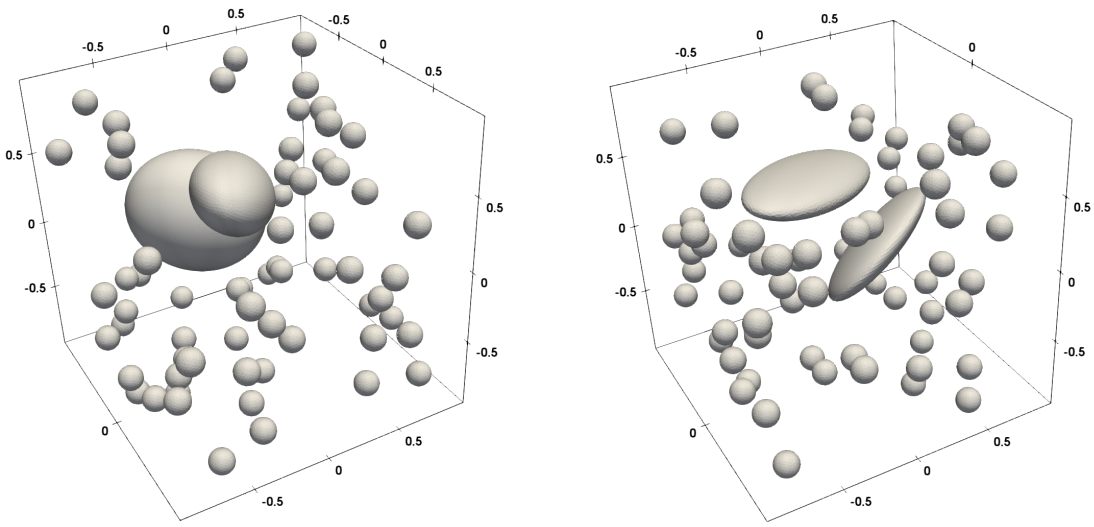


Figure 6.7: Two realisations of a cubical specimen from a heterogeneous material with a bimodal distribution of random pores: large elliptical pores and smaller equally-sized spherical pores.

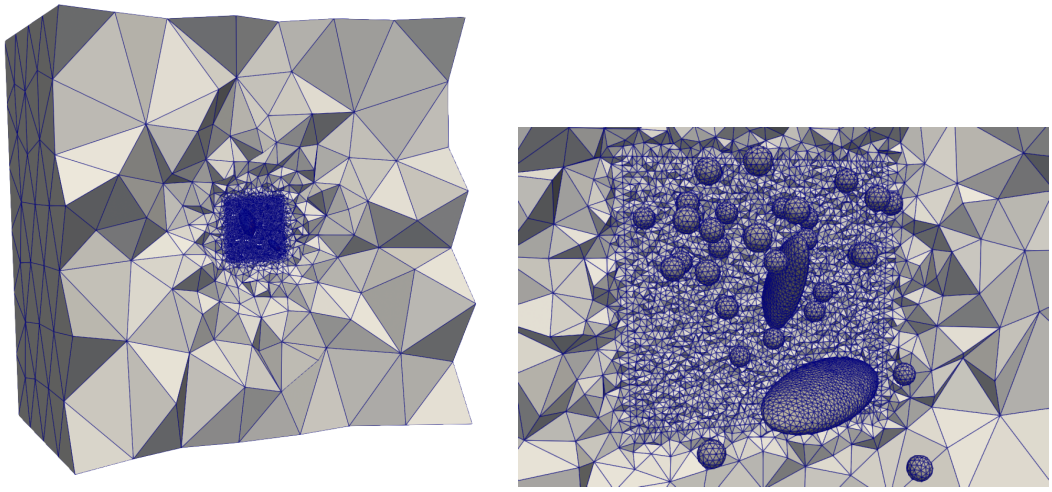
where E_{xx}, E_{xy}, E_{xz}, E_{yy}, E_{yz}, E_{zz} are far-field load parameters, X is the position of a point in $\mathbb{R}^3$ and X_0 is the initial position of the centre of the body in $\mathbb{R}^3$.

For data pre-processing we use a simple linear rescaling by dividing the input stress components with the maximum stress component in the training dataset.

The dataset we use consists of 200 FE simulations completed in 8.7 hours on 5 cores on an Intel® Xeon® Gold 6148 CPU @ 2.40GHz CPU using the Hawk Supercomputer located at Cardiff University. From every FE simulation we extract 5 patches, at random locations in the cubic specimen, resulting in 1,000 data points, 900 for training and 100 for testing.

Training parameters

For the training of the network we used a batch size of 2 and the Adam optimiser with an initial learning rate of $1 \cdot 10^{-4}$ decreasing by 5% every 50 epochs. Additionally, we performed several tests to identify key parameters that affect the training and performance of the GNN and determined their optimum values. We concluded that for achieving optimum behaviour we will use a GNN with 5 GN blocks, residual connection, independent encoder, 128 filters and a maximum of 10 neighbours per node. The total number of trainable parameters for the network is 842,654. More details can be found



(a) (b)

Figure 6.8: On the left (a) we can see a volume mesh of the multiscale structure. The buffer area where the mesh is coarse is visible. In the centre we can see the dense mesh area from where we extract the patches. The shape of the dense mesh area is a cube, but here only one face of this cube is visible. On the right (b) we can see a zoom in the dense mesh region. We stress that the fine scale features visible are inside the dense mesh area.

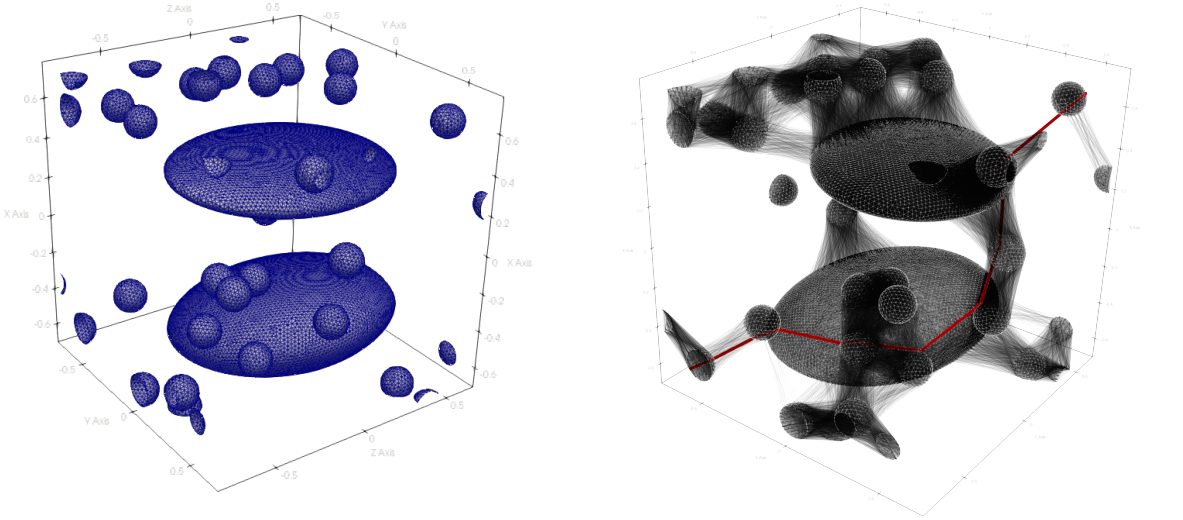
in [Appendix B].

In the GNN architecture examined in this chapter, there are two types of layers that have learnable parameters, namely the linear and layer normalisation layer. For both layers PyTorch Geometric's default initialisation methods are being used. For the linear layer, the weights are initialised using the Kaiming uniform method, as described by [He et al., 2015], with the default parameters used by PyTorch, namely $a = \sqrt{5}$. The biases are initialised using uniform initialisation, where values are drawn from a uniform distribution between 0 and 1. For the layer normalisation the weights are initialised to 1 and the biases to 0. The same methods are applied to all the GNNs trained in this work.

Training on 900 patches using the parameters identified above requires 6 hours on an NVIDIA V100 GPU with 16GB of RAM with a maximum memory requirement of 4.3GB using the Hawk Supercomputer located at Cardiff University.

GNN graph

In this section we provide information about the graph used for the GNN training. The graph is constructed using Euclidean edges as described in [Section 6.2.2]. An example can be found in [Fig 6.9], where both the surface mesh and the graph can be seen. A threshold of 10 neighbours per node is being used and a radius of 0.32 for the radius neighbours. The patch has 20,113 points. The shortest path to connect 2 points of the graph located on opposite corners is 24 steps.



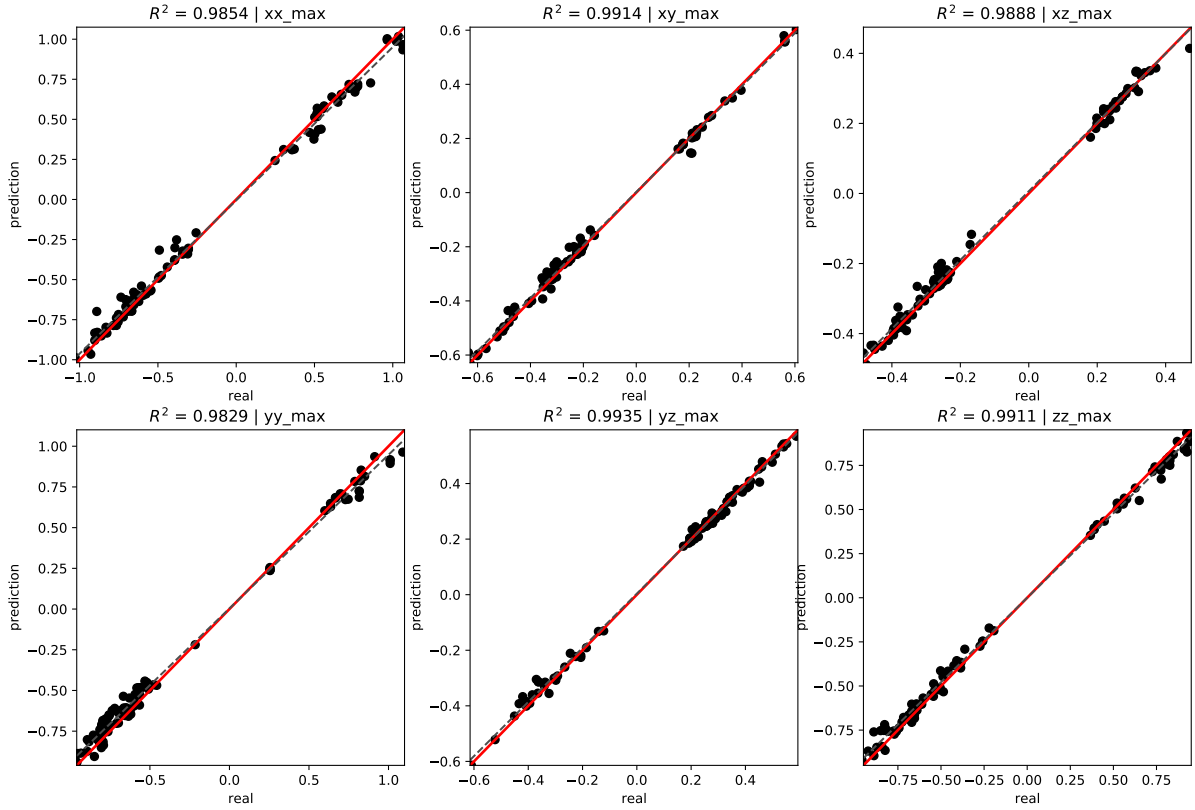
(a) FE mesh of the patch (b) Graph of the patch used for GNN training

Figure 6.9: On the left (a) we can see the surface mesh of a patch. On the right (b) we can see the graph of the same patch that will be used for the GNN training. The graph connects the nodes of the surface mesh using Euclidean edges. With red we can see the shortest path that connects two points of the graph located on opposite corners.

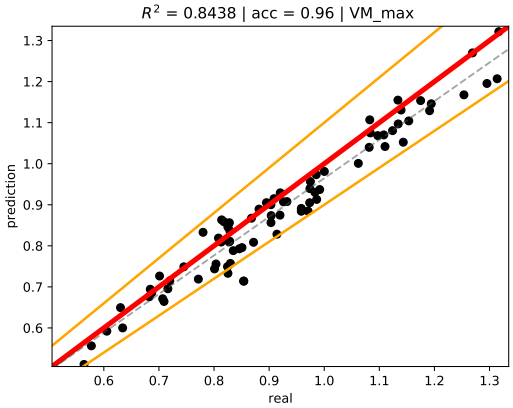
BGNN mean prediction

In this section we will examine the mean prediction provided by the BGNN, without referring to the uncertainty estimation.

In [Fig 6.10a] we observe that the maximum stress components predicted by the BGNN are in a good agreement with the ones calculated by the FEA. Specifically, the coefficient of determination, R^2, for all the 6 stress components is larger than 0.98. In [Fig 6.10b] we observe similar results for the maximum Von Mises stress. Although the points are more scattered and the coefficient of determination dropped to 0.84, the accuracy is 96%.



(a) maximum microscale components



(b) maximum microscale Von Mises stress

Figure 6.10: In both subfigures the x-axis corresponds to the FE prediction in the ROI of each patch and the y-axis to the BGNN prediction in the ROI of each patch. In the top subfigure, (a), we observe 6 diagrams corresponding to each of the maximum absolute microscale stress components. In the bottom subfigure, (b), we observe the maximum Von Mises stress.

Additionally, we examine the stress distribution on the FE mesh. In [Fig 6.11] we compare the microscale Von Mises stress in the ROI of a patch as predicted by the GNN with the one calculated by FE simulations. We observe that the predicted stress distribution is very close to the ground-truth.

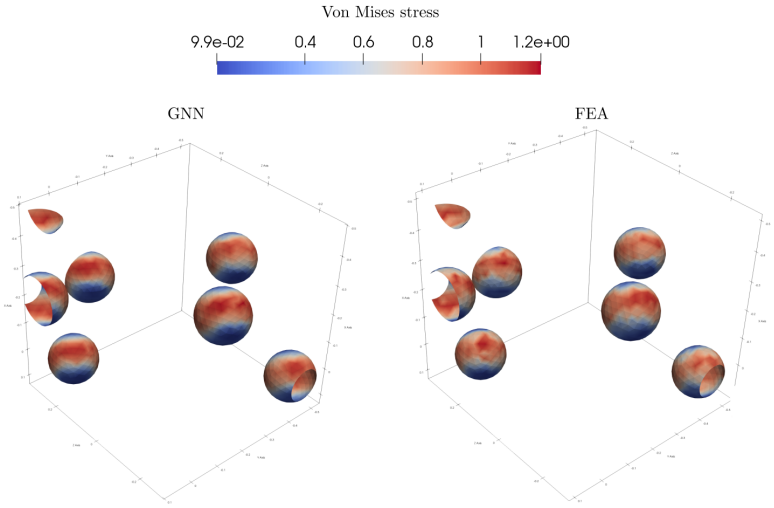


Figure 6.11: Comparison between the Von Mises stress distribution as calculated by FEA (right) and the Von Mises stress distribution as predicted by the GNN (left).

BGNN uncertainty estimation

After studying the quality of the mean predictions provided by the BGNN, here we evaluate the ability of the BGNN to quantify the uncertainty of the prediction and provide CIs that reflect the error in the prediction. From [Fig 6.12] we can see that the percentage of points inside the 95% CIs is 92%, which is satisfactory.

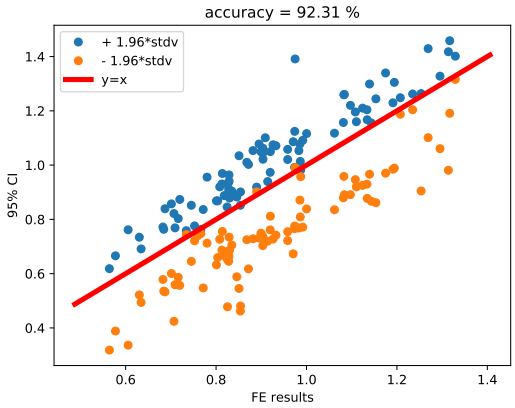


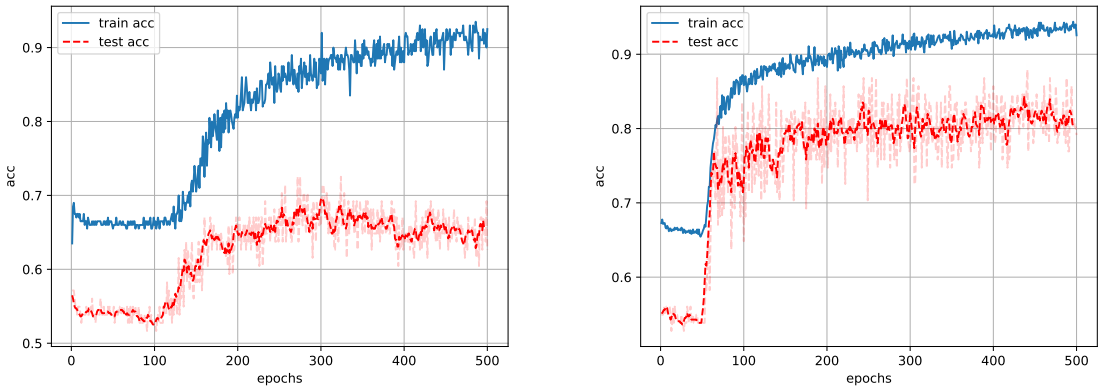
Figure 6.12: In this figure we see the upper and lower 95% CIs for the maximum Von Mises stress prediction, blue and orange points respectively.

6.4.2 Online stress correction

In this section we will demonstrate the use of the Ensemble Kalman method, [section 6.3], to improve the BGNN prediction using physics information. We will examine 2 realistic cases; the first case is to improve the prediction of an under trained network and the second case is to improve the prediction of a network in an extrapolation regime.

Under trained GNN

With the phrase under trained we refer to a network for which we use a small dataset for training. In [Fig 6.13] we show the accuracy plots for the mean BGNN prediction as a function of the epochs for 2 GNNs trained with the same parameters but the one on the left uses data from 32 FE simulations for training and the one on the right uses data from 100 FE simulations for training. The BGNN trained with more data performs better than the undertrained network.



(a) 32 FE simulations

(b) 100 FE simulations

Figure 6.13: In both diagrams we see accuracy curves defined using the mean BGNN prediction for the maximum Von Mises stress. Specifically, we see the accuracy as function of the training epochs for the training set (blue) and the test set (red). In the diagram on the left (a) the GNN is trained with 32 FE simulations, 160 patches, and in the diagram on the right (b) the GNN is trained with 100 FE simulations, 500 patches.

Using the Ensemble Kalman method we want to update the output distribution of the BGNN, which we consider to be the prior, and obtain the posterior. In [Fig 6.14] we can see the posterior and the prior for the prediction of the maximum Von Mises stress. The coefficient of determination between the posterior and the FE results has a value of 0.3074 which is larger than the coefficient of determination between the prior

and the FE results that has a value of 0.0176. Also, we can see that best line that fits the posterior data is closer to the $y = x$ line (ideal result) compared to the best line that fits the prior data, i.e. the output of the BGNN, without the online correction. Therefore, the posterior mean better fits the data compared to the prior mean and thus the online stress correction resulted in an improvement of the mean BGNN prediction.

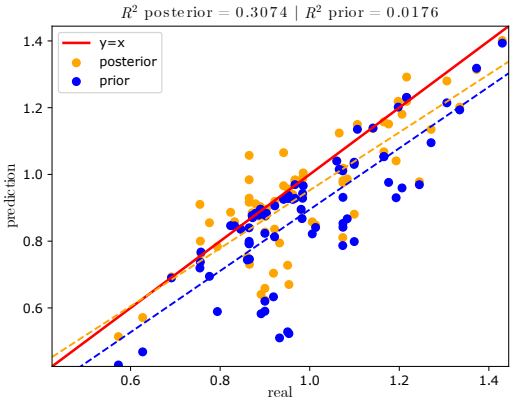
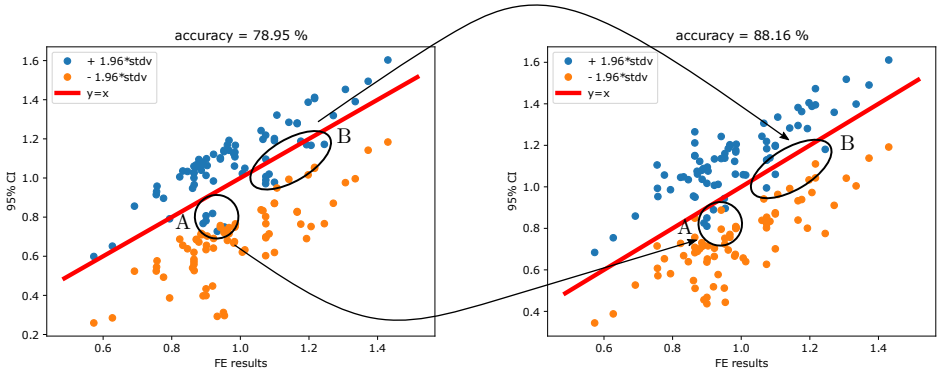


Figure 6.14: A diagram showing the posterior (yellow points) maximum Von Mises prediction and the prior (blue points) maximum Von Mises prediction. We observe that the posterior, after the stress update, has a larger coefficient of determination compared to the prior.

Furthermore, we want to examine the posterior distribution and not only the posterior mean. In [Fig 6.15] we see the comparison between the prior 95% CIs (left) and the posterior 95% CIs (right). Firstly, we observe that even in the prior case, where we have only used 32 FE simulations, the 95% CIs give a good estimation of where the real value is. We see that the percentage of points that are inside the 95% CIs is 79%. Additionally, we see that this value climbs to 88% in the posterior case. By using the online stress update we improved the BGNN prediction without adding more data to the training dataset.

Lastly, in [Fig 6.16] we compare the mean Von Mises stress distribution on the mesh before and after the stress update with the one calculated through FE simulations. We can see that the posterior, middle image, is closer to the FE results, image on the left, compared to the prior, image on the right.



(a) 95% CIs before the stress update

(b) 95% CIs after the stress update

Figure 6.15: Comparison between the prior and posterior 95% CIs for the maximum Von Mises stress in the ROI of the patches for a BGNN trained with only 32 FE simulation data. The diagram on the left, (a), is showing the upper and lower 95% CIs, blue and orange points respectively, before the stress update. While the diagram on the right, (b), is after the stress update. Two clusters, A and B, are marked before and after the stress correction highlighting that blue points, +95% CIs, which were below the $y = x$ line for the prior case have moved towards the $y = x$ line and in a lot of cases surpassed it, in the posterior case.

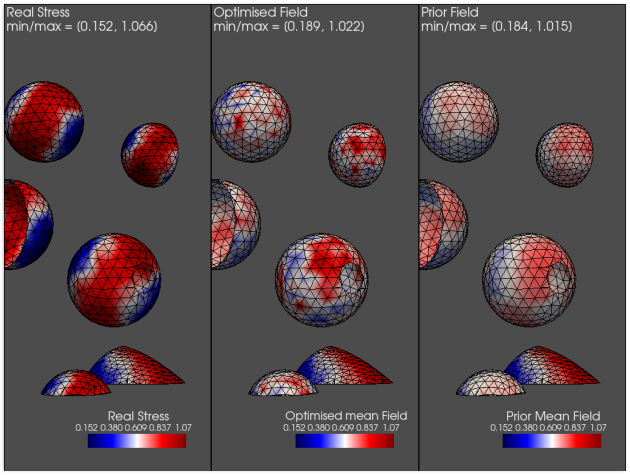
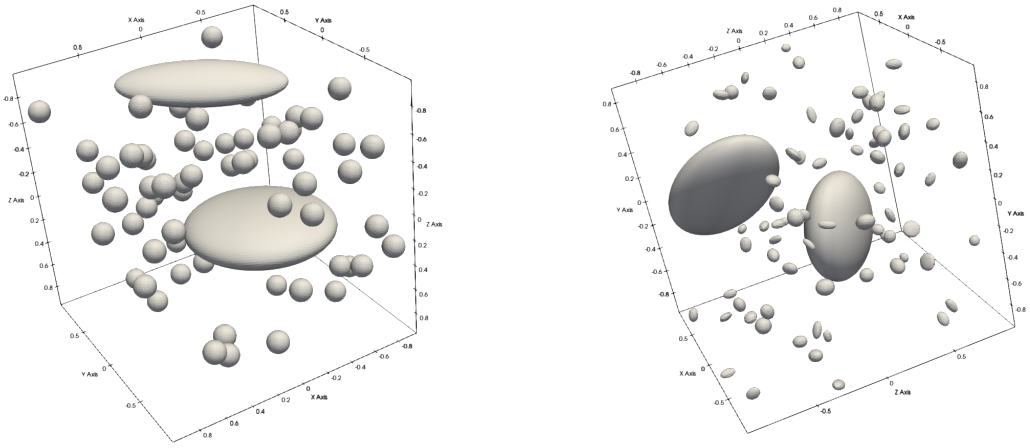


Figure 6.16: Comparison between the Von Mises stress before and after the online stress update with the one calculated through FE simulations. On the left we see the FE results, in the middle the posterior mean and on the right the prior mean.

Prediction for out-of-training microstructural inputs

In this section, with the phrase "out-of-training" we refer to a network trained with spherical defects but evaluated in cases that have elliptical defects. The BGNN is extrapolating which is a particularly undesirable but common scenario for real applications. In [Fig 6.17] we show a realisation of the training dataset on the left and a realisation of the test dataset on the right.



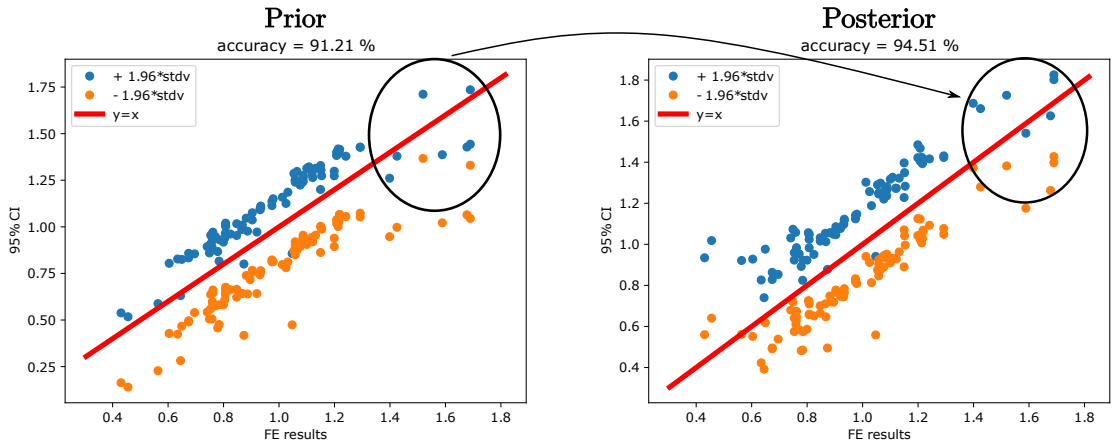
(a) a sample from the training dataset (b) a sample from the test dataset

Figure 6.17: On the left (a) we see a realisation of the training dataset and on the right (b) of the test dataset. For the training dataset the porous phase is composed of spheres while for the test set of ellipsoids.

In [Fig 6.18] we see the comparison between the prior 95% CIs (left) and the posterior 95% CIs (right). Firstly, we can observe that even in the prior case, where the BGNN extrapolates, the 95% CIs give a good estimation of where the real value is. We can see that the percentage of points that are inside the 95% CIs is 91%. Additionally, we see that this value climbs to 95% in the posterior case. Most importantly we see that using the ensemble Kalman method we managed to avoid underpredicting the maximum values. This clearly means that by using the Ensemble Kalman method we were able to improve the BGNN prediction without adding more data to the training dataset.

We stress that the results of this section do not suggest that the GNN framework could be used for extrapolation away from the dataset. Instead, they indicate that

given a reasonably accurate prediction where a downward bias is observed for the highest stress values, the proposed online stress correction technique may be deployed to correct this bias upward.



(a) 95% CIs before the stress update

(b) 95% CIs after the stress update

Figure 6.18: Comparison between the prior and posterior 95% CIs for the maximum Von Mises stress in the ROI of the patches for a BGNN trained on a dataset with spheres as microscale features and tested on a dataset with ellipses as microscale features. The diagram on the left, (a), is showing the upper and lower 95% CIs, blue and orange points respectively, before the stress update. While the diagram on the right, (b), is after the stress update. A cluster of points is marked before and after the stress correction highlighting that blue points, +95% CIs, which were below the $y = x$ line for the prior case have moved towards the $y = x$ line and in a lot of cases surpassed it, for the posterior case.

6.4.3 Numerical example with dogbone data

After demonstrating the applicability of our GNN and studying the online correction technique, in this subsection we apply the proposed deep learning methodology to a more challenging case. We also provide multiple examples of the Von Mises stress distribution on the surface of the mesh.

Training dataset

The geometry that we choose to model is a dogbone with a single macroscale hole in the middle that does not change from one realisation to another, and with a random distribution of 50 to 100 microscale spherical pores. The size of the dogbone is 2 units along the x axis (length), 0.6 units along the y axis (height) and 0.08 units along the z axis (width). The radius of the hole is 0.032 units, and the radius of the spherical pores is 0.016 units. The spherical pores may intersect with each other, with the boundaries of the geometry and the hole. For the macroscale simulations, input of the BGNN, only the hole is taken into account and the spherical pores are ignored. In [Fig 6.19] we show 2 examples of the geometry. For the boundary conditions we apply displacements on those faces of the dogbone specimen that are perpendicular to the x axis. The prescribed displacement has the same magnitude for both sides, and is applied along the positive outer normal direction. Zero displacement is applied along the other two spatial directions. Lastly, The Young's modulus and the Poisson ratio of the structure are 1 and 0.3 respectively.

We performed 500 FE simulations completed in 5.7 hours on 10 cores on an Intel® Xeon® Gold 6148 CPU @ 2.40GHz CPU. From every FE simulation we extract 10 patches resulting in 5,000 data points, 4,000 for training and 1,000 for testing. As explained in section [6.4.1] the patch is of size $[18R \times 18R \times 18R]$ and the ROI is of size $[8R \times 8R \times 8R]$, where R is the radius of the spherical pores.

GNN parameters

For the GNN training we used the Adam optimiser with an initial learning rate of $1 \cdot 10^{-4}$ decreasing by 5% every 50 epochs. Additionally, we started with the parameters that we identified in [6.4.1], namely 5 GN blocks, residual connection, independent encoder, 128 filters and a maximum of 10 neighbours per node. After experiments we concluded that in this case the GNN can benefit from a maximum of 20 neighbours per node but all

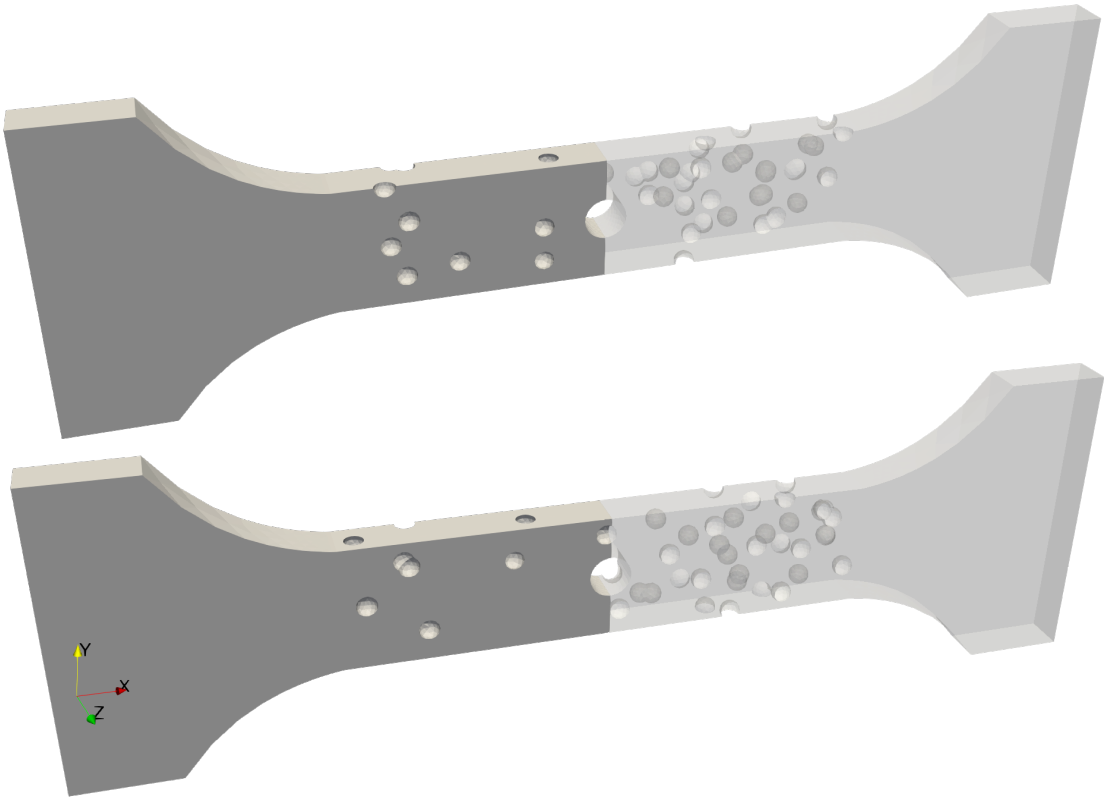


Figure 6.19: Two realisations of the dogbone used to train the GNN. Each dogbone has a cylinder-shaped hole and the porous material is defined via a random distribution of spherical voids.

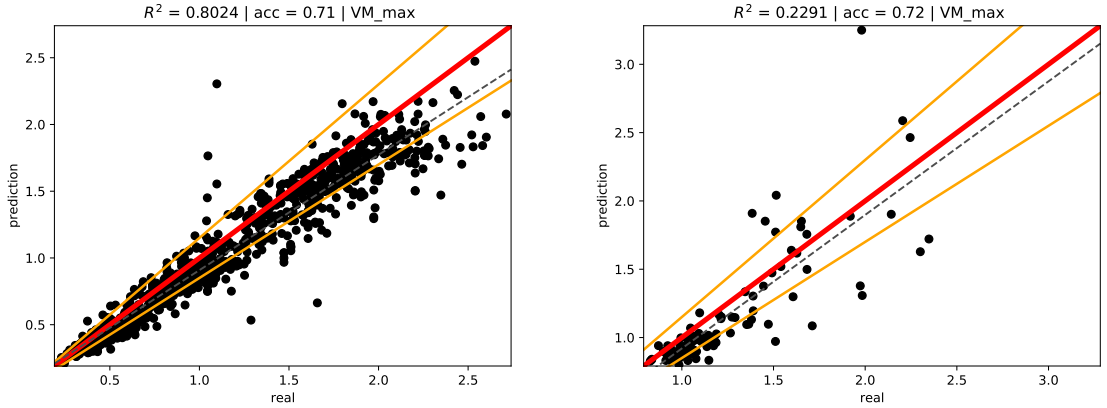
the other hyperparameters remained unchanged. Training of the GNN was performed on an NVIDIA V100 GPU with 16GB of RAM.

In this more challenging problem, we investigate the idea of using dual convolutions. Instead of only considering the Euclidean neighbourhood we also consider the Geodesic neighbourhood of every node in order to perform joint Geodesic and Euclidean convolutions as explained in [6.2.2]. After experiments, we concluded that in our case the optimum value for the ratio between Geodesic and total convolutions is 75%. For more details the reader can refer to [Appendix B.6].

Additionally, throughout this chapter we made the choice to predict the full stress tensor instead of the Von Mises stress and to work on patches of the structure instead of the entire structure. Using data from the dogbone dataset we performed experiments to support these choices in [Appendix B.7] and [Appendix B.8] respectively.

Numerical examples with deterministic GNN

As can be seen in [Fig 6.20a] when training with 4000 patches the accuracy for the maximum Von Mises stress is 71% which is considerably lower than the accuracy reported for the cubical heterogeneous material case, 96%. In [Fig 6.20b] we can also evaluate the performance of the GNN on the entire dogbone and not only at patch level. The accuracy does not change considerably in this case, but the data are more spread around the $y = x$ line which is indicated by the lower coefficient of determination R^2.



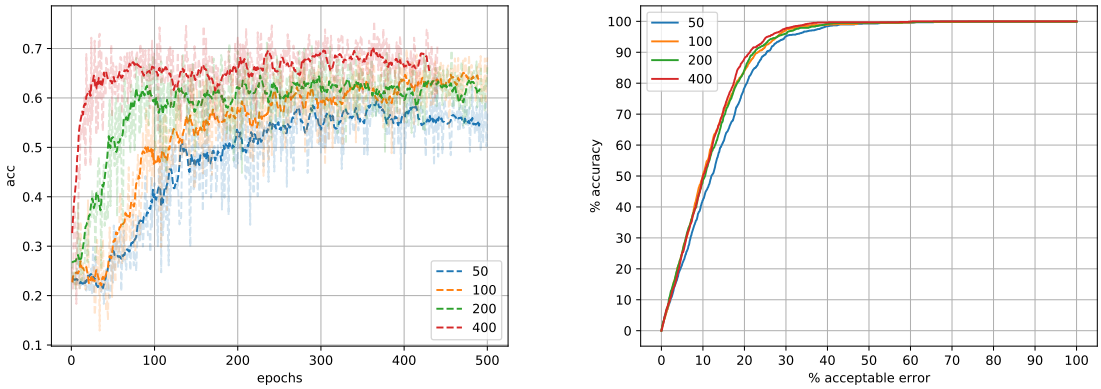
(a) evaluate on patches

(b) evaluate on entire structure

Figure 6.20: In both subfigures the x-axis corresponds to the maximum Von Mises stress as calculated by FEA and the y-axis as calculated by the GNN. For the left subfigure, (a), we see results for the ROI of the patches while for the right subfigure, (b), for the entire dogbone.

Additionally, we investigate the performance of the network with respect to the size of the training dataset. We train 4 GNNs with patches generated from 50, 100, 200 and 400 FE simulations. The results can be found in image [Fig 6.21]. From [Fig 6.21a] we can observe that the accuracy increases as we increase the size of the training dataset, from 0.5750 for the GNN trained with 50 FE simulations to 0.7060 for the GNN trained with 400 FE simulations. This strongly suggests that the accuracy can further increase given more data, although in this work we try to focus on realistic scenarios where the training data are not abundant. To support our hypothesis, we create 400 additional FE simulations, and we train a GNN with all the 800 FE simulations, but we refrain from further increasing the size of the dataset. The accuracy of the GNN is 82% as can be seen in [Fig 6.22]. This GNN was trained using as initial weights the ones determined by the GNN trained with 400 FE simulations. In [Fig 6.21b] we investigate

the dependency of the accuracy on the relative error threshold. We observe a sharp increase of the accuracy with respect to the threshold values, as long as the threshold is less than 20%. After 20% the accuracy slowly converges to 100%. We note that with only 50 direct FE simulations, and for the 20% threshold, the accuracy reaches 80%. Although such levels of accuracy are relatively loose, this somewhat contradicts the widespread idea that NNs require too much training data to be of use in areas of science where data is sparse and/or expensive to acquire.



(a) Accuracy

(b) Accuracy VS error threshold

Figure 6.21: In the diagram on the left (a) we see accuracy curves defined using the maximum Von Mises stress and a 15% threshold for the relative error. In the diagram on the right (b) we see accuracy curves with respect to the threshold used for the relative error. For both diagrams, coloured lines correspond to GNNs trained with datasets of different size, namely patches extracted from 50, 100, 200 and 400 FE simulations.

Furthermore, we demonstrate the quality of the stress distribution. In [Fig 6.23] and [Fig 6.24] we show that the stress field predicted by the GNN is in good agreement with that provided by direct FEA.

Lastly, we are interested in studying the predicted stress distribution in the highly stressed regions of the specimen. In [Fig 6.25a] we have zoomed in a region where a spherical void interacts strongly with the cylinder-shaped hole, and in [Fig 6.25b] we have zoomed in at a strong interaction between 2 spherical voids. We observe that in both cases the GNN successfully located the area where the maximum Von Mises occurs, predicted its value and the stress distribution around it.

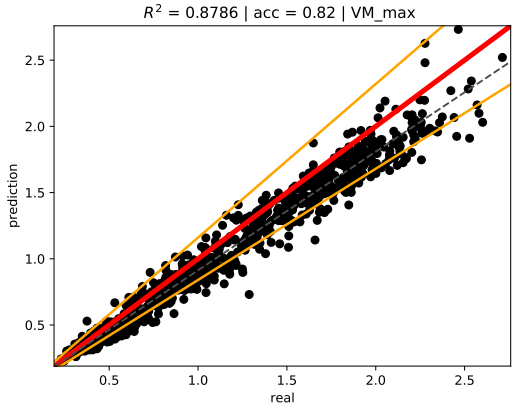


Figure 6.22: In this diagram the x-axis corresponds to the maximum Von Mises stress as calculated by FEA and the y-axis as calculated by the GNN. The accuracy, defined by the 15% relative error threshold, is 82%.

GNN prediction dependency on the input mesh

In this section we aim to examine the dependency of the GNN prediction on the density of the input mesh. To this end, we propose to mesh one single porous dogbone realisation with four different mesh densities. The four different meshes have typical edge lengths 0.008, 0.009, 0.010 and 0.011. In [Fig 6.26] we can see the results obtained by applying the NN to the patch. We observe that the stress distribution does not change considerably from the 0.008 to the 0.009 case but as the mesh gets coarser, like in the 0.011 case, the GNN does not manage to capture the highly stressed area correctly. We highlight that the GNN was trained using examples meshed with a typical edge length of 0.008.

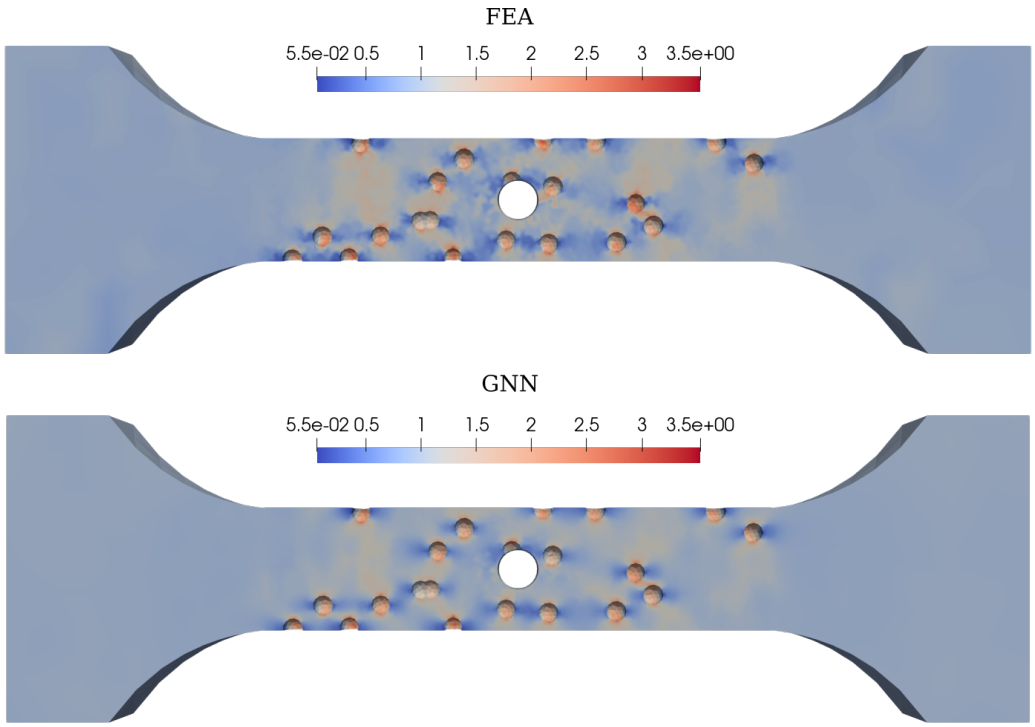


Figure 6.23: Comparison between the Von Mises stress distribution as calculated by FEA (top) and Von Mises stress distribution predicted by the GNN (bottom). The GNN result is reconstructed from the union of multiple patch predictions where only the ROI is extracted.

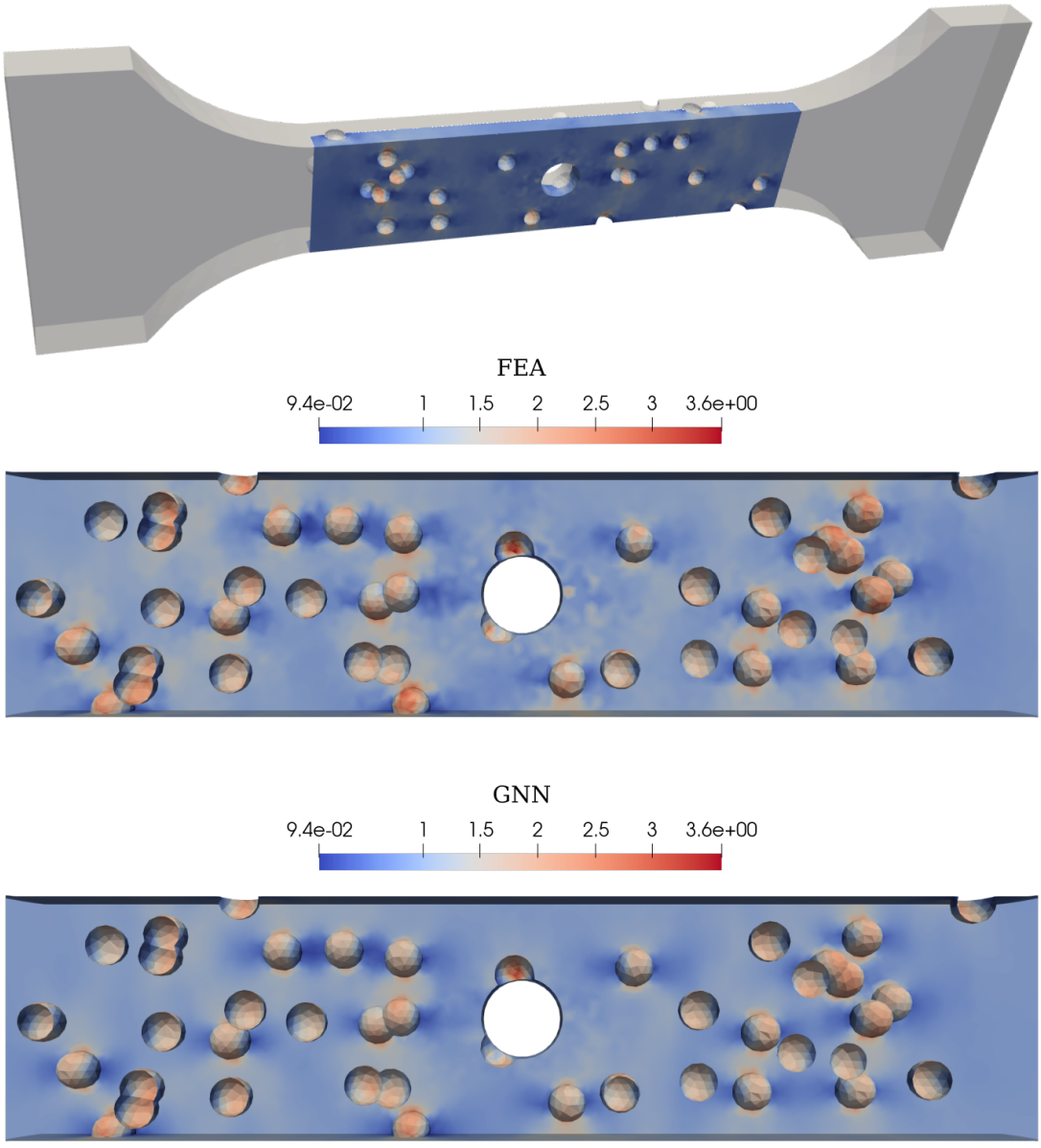
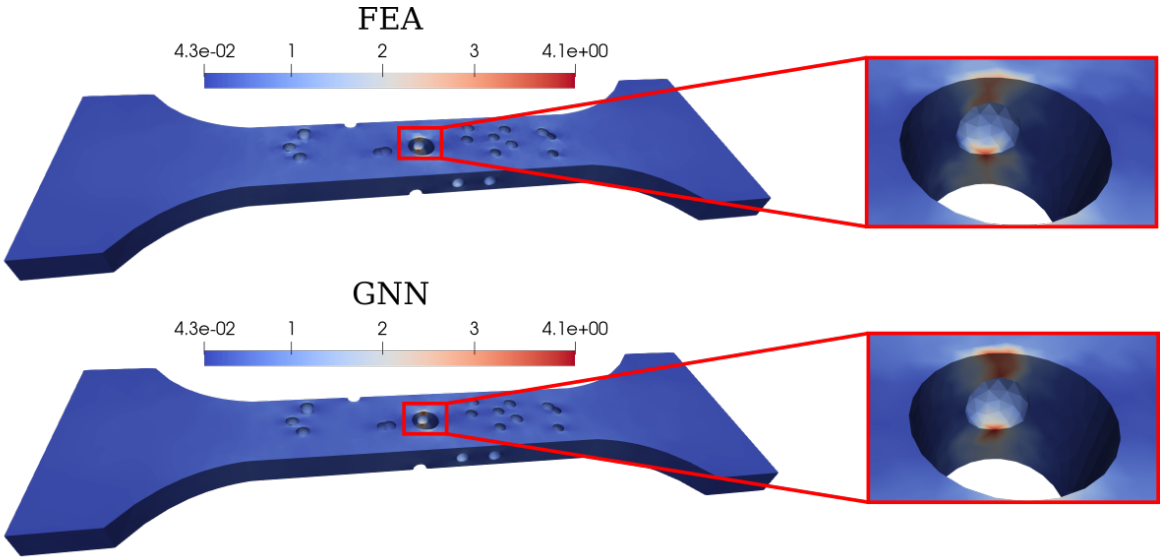
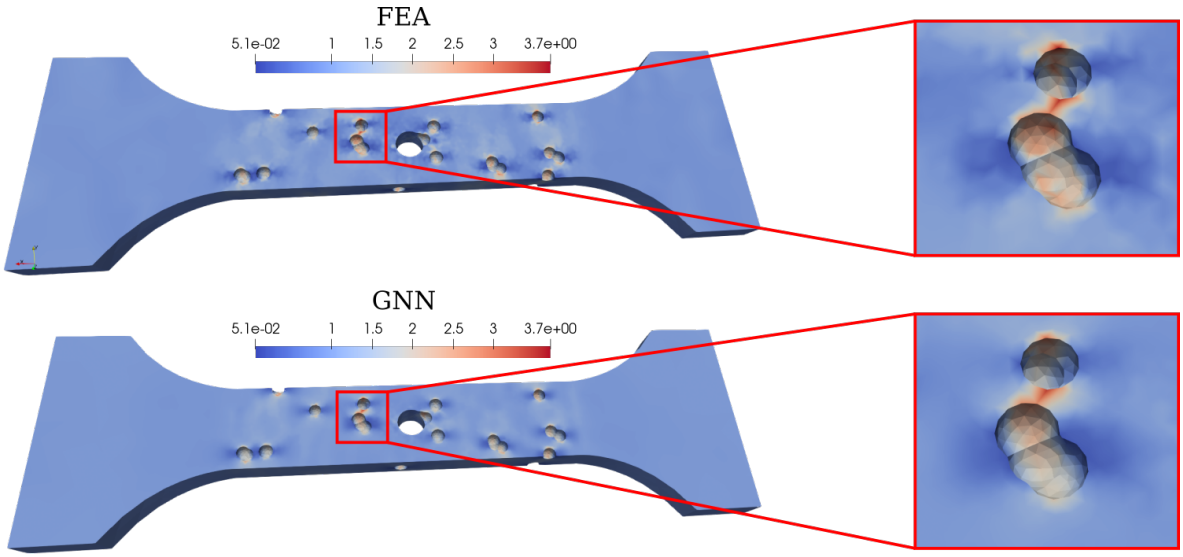


Figure 6.24: Comparison between the Von Mises stress distribution, extracted from a cross section of the structure (top), as calculated by FEA (middle) and as predicted by the GNN (bottom). We can observe that the two distributions are qualitatively very similar. The GNN result is reconstructed from the union of multiple patch predictions where only the ROI is extracted.

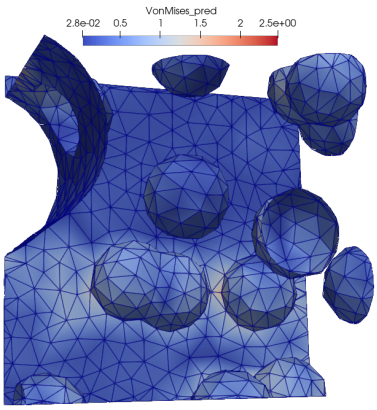


(a) spherical void - cylinder-shaped hole interaction

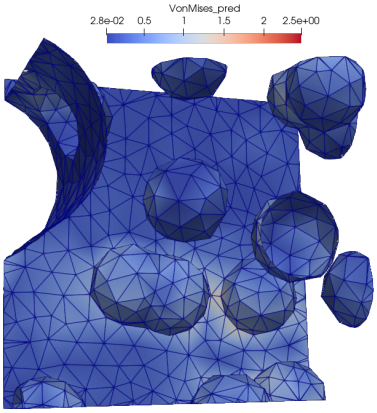


(b) spherical void - spherical void interaction

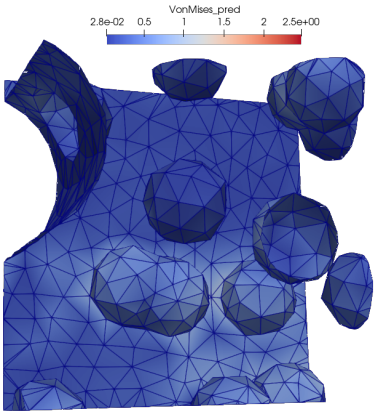
Figure 6.25: In both subfigures we compare the maximum Von Mises stress as calculated by FEA and as predicted by the GNN. In the top subfigure (a), the maximum is due to an interaction between spherical voids and the cylinder-shaped hole while in the bottom, (b), due to an interaction between 2 spherical voids.



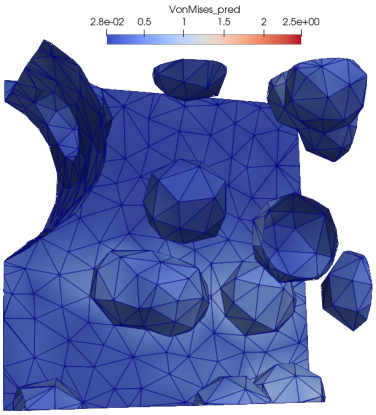
(a) edge length = 0.008



(b) edge length = 0.009



(c) edge length = 0.010



(d) edge length = 0.011

Figure 6.26: Evaluation of the performance of the GNN on input meshes of different density. In all of the figures the microscale Von Mises stress is plotted.

6.4.4 Variable dimension dogbone

In this section we demonstrate the ability of our method to be used for exploration of responses in a space of shapes described by a few geometrical parameters. To this end, we perform geometric learning in a family of dogbone shapes where not only the microscale spherical void distribution will be different from realisation to realisation but also the macroscale cylindrical-hole and the dimensions of the dogbone. We focus on the dependence of the quality of the results with respect to the size of the dataset both in terms of maximum Von Mises stress and stress distribution.

Training dataset

To create this dataset we follow the same process as in the "Dogbone" section, [6.4.3]. The difference lays on the fact that here the dimension of the specimen is not constant. The length of the dogbone is equal to 2, the width varies from 0.06 to 0.12 and the height varies from 0.24 to 0.7. Additionally, the number of cylindrical holes varies from 1 to 4 in every realisation, and their radii from 0.024 to 0.06. Two realisations of the variable dogbone geometry can be found in [Fig 6.27].

Numerical example with variable dimension dogbone data

We perform a similar experiment as in the section [6.4.3], where we investigate the dependency of the accuracy on the size of the dataset [Fig 6.28]. The accuracy increases as we increase the size of the dataset, from 50.21% for the GNN trained with 50 FE simulations to 64.30% for the GNN trained with 400 FE simulations. Compared to the previous family of dogbone specimen, where the dimensions of the structure and the cylindrical hole did not change from one realisation to another, we can observe that the accuracy that can be achieved using 400 FE simulations decreases from 71% to 64% for the variable dogbone case. This result was to be expected since the space of shapes that we attempt to learn has more parameters than before. In order to achieve similar accuracy, we create additional 600 FE simulations and we train the GNN for 100 epochs using as initial weights the ones determined by the GNN trained with 400 FE simulations. The accuracy achieved for this case is 70%.

Additionally, in order to examine the stress distribution, in the following figure we have extracted the upper part of the structure so that we can observe the stress distribution inside. This can be found in [Fig 6.29], where we can see that the GNN

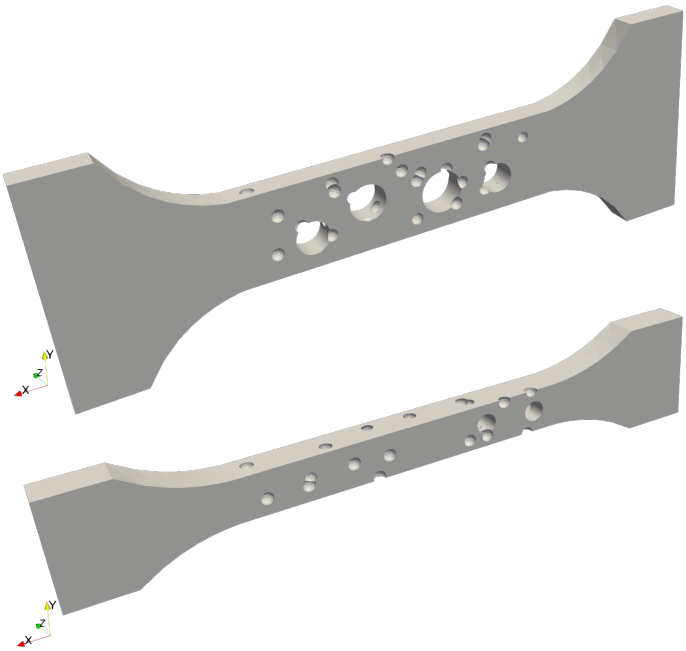
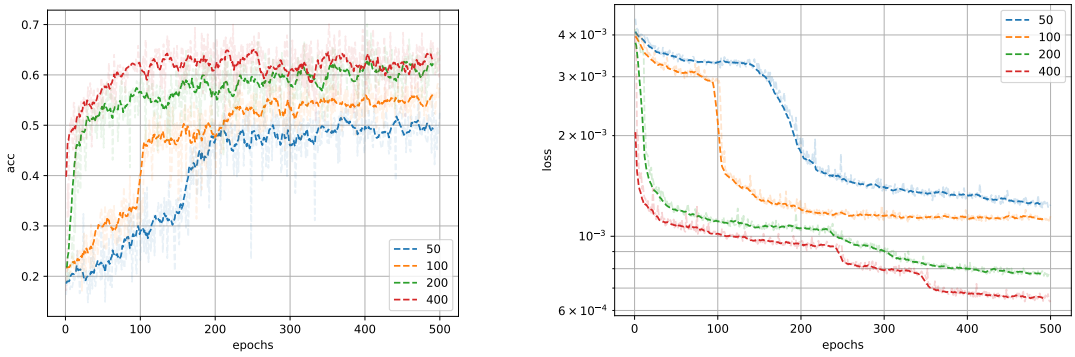


Figure 6.27: Two realisations of the variable dogbone structure. In the top we observe a dogbone with large height, small width and 4 cylindrical holes. In the bottom we observe a dogbone with small height, large width and 2 cylindrical holes.

successfully reconstructed the Von Mises distribution.

Lastly, we want to examine the quality of the stress distribution with respect to the size of the training dataset. In [Fig 6.30] we see the Von Mises prediction for a case where not many strong interactions are present. We can observe that all the GNNs even the one trained with 100 FE simulations successfully predicts the correct Von Mises stress distribution. On the other hand, in [Fig 6.31] we can see a case where two intersecting defects, located at the corner of the domain, strongly interact and we can observe that the GNNs trained with less than 400 FE results fail to predict this interaction.



(a) Accuracy (b) Loss

Figure 6.28: In the diagram on the left (a) we see accuracy curves defined using the maximum Von Mises stress and a 15% threshold for the relative error. Specifically, we see the accuracy as function of the training epochs. In the diagram on the right (b) we see the loss function as a function of the epochs. For both diagrams, coloured lines correspond to GNNs trained with datasets of different size, namely patches extracted from 50, 100, 200 and 400 FE simulations.

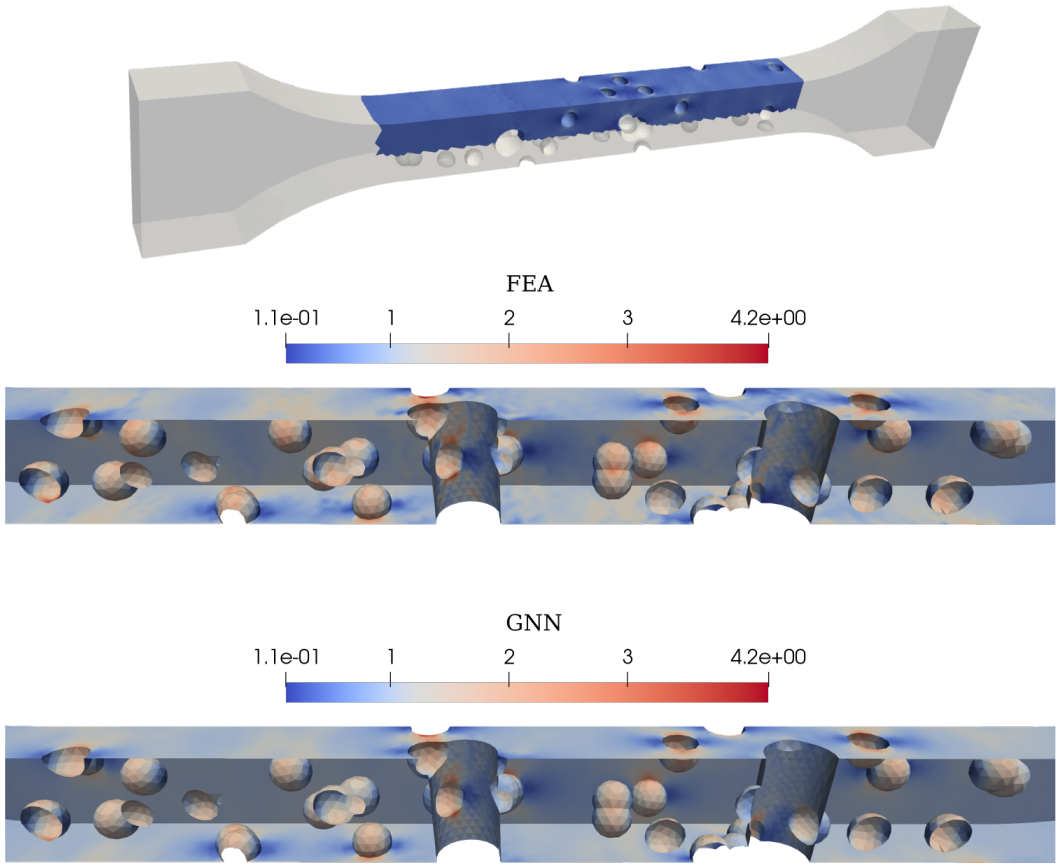


Figure 6.29: Comparison between the Von Mises stress distribution, extracted from the top of the structure (top), as calculated by FEA (middle) and as predicted by the GNN (bottom). We can observe that the two distributions are qualitatively very similar. The GNN result is reconstructed from the union of multiple patch predictions where only the ROI is extracted.

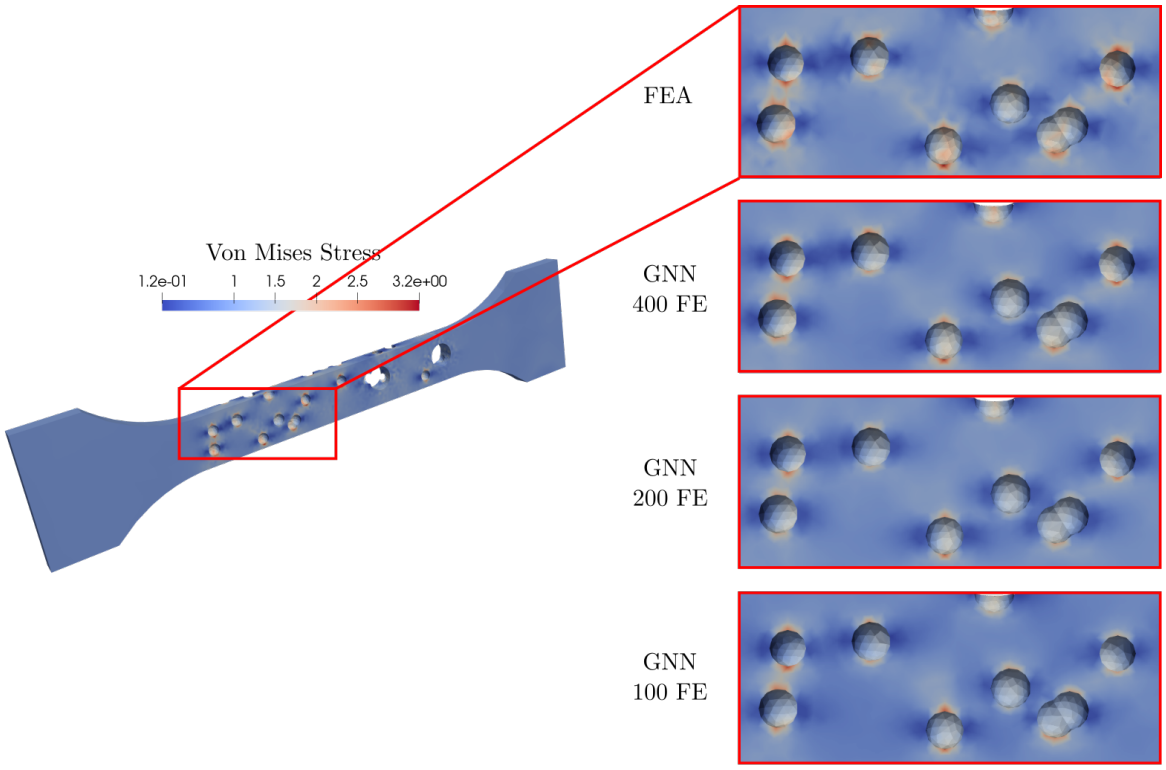


Figure 6.30: On the left the Von Mises distribution on the surface of a dogbone structure as calculated by FE simulations. On the right, we can see a zoom of an area of the dogbone. From top to bottom we see the FE simulation results, the GNN prediction for a GNN that was trained with patches extracted from 400, 200 and 100 FE simulations. We can observe that all the GNN predictions are very close to the FE results.

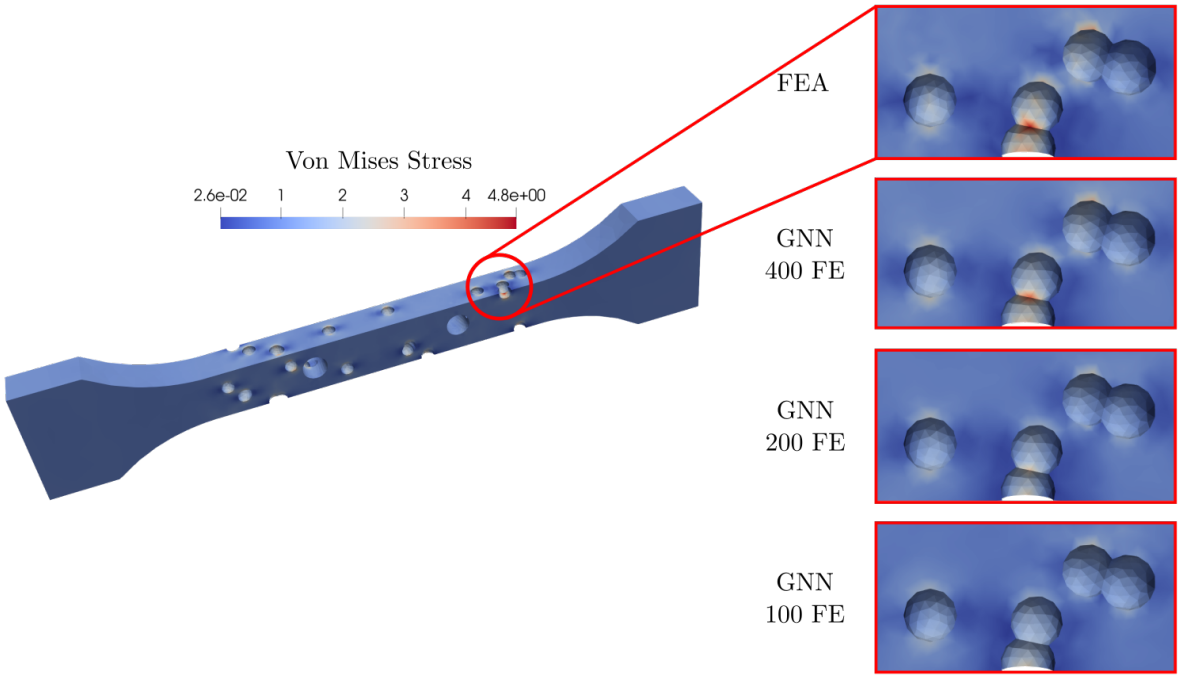


Figure 6.31: On the left the Von Mises distribution on the surface of a dogbone structure as calculated by FE simulations. On the right, we can see a zoom of the maximum Von Mises stress area. From top to bottom we see the FE simulation results, the GNN prediction for a GNN that was trained with patches extracted from 400, 200 and 100 FE simulations. We can observe that only the GNN that was trained using 400 FE simulations was able to correctly predict the maximum Von Mises stress that is created from the interaction between two spherical voids.

6.5 Conclusion and perspectives of this chapter

The goal of this chapter was to extend the 2D Convolutional Neural Network (CNN) based multiscale framework developed in [Chapter 5] to 3D problems and to ultimately create a 3D Graph Neural Network (GNN) surrogate model to circumvent the need for 3D direct numerical analysis of stress simulations in porous materials and structures. To this end we show that given an inexpensively computed macroscopic approximation of the stress field, where the pore network is ignored, and a fine scale representation of the geometry of the structure and of the network of microscale pores, a 3D GNN is able to correct the macroscopic stress field to produce fields that emulate the output of the direct numerical simulation.

The developed framework is based on geometric learning. We develop a GNN to perform convolution-based learning over the surface of the porous structure, which is represented by a mesh of triangles. This surface mesh represents both the structure boundary and the surface of the pores. An advantage of using geometric learning for the surrogate modelling of porous structures is that we eliminate the need to perform unnecessary algorithmic operations associated with the analysis of bulk information, which would typically be required in a standard voxel-based CNN approach. In addition, the GNN architecture allows us to operate on a variable resolution mesh of the geometry rather than a representation with fixed spatial resolution. Our choice to deploy geometric learning over the surface of the porous structure is supported by three observations.

Firstly, the surface representation is a comprehensive representation of the geometry of the specimen. Secondly the macroscale stress over the surface mesh is empirically found to be a sufficiently rich information for the GNN to produce correct microscale corrections of the stress state in the porous material. Lastly, we have shown numerically, that the maximum microscale stress field in the porous network that we analysed in this chapter was always found at the surface of the porous structure.

The 3D graph-based GNNs developed in this chapter report similar trends in their prediction abilities, compared to the 2D pixel-based CNNs developed in [Chapter 5]. This includes the uncertainty quantification procedure, which is performed via a Bayesian formulation of the GNN. In particular, we have examined the ability of the proposed approach to predict the maximum Von Mises stress, in two different settings : (i) repeated structural calculations with random distributions of pores and (ii) a surrogate modelling of parameterised structural problems with random microstructures.

For application (i), we have shown that relatively few offline simulations were required to train the GNN. Training with 50 simulations already give predictions where 80% of new outputs are predicted with less than 20% relative error. However, our investigations also show that in applications where a lower error threshold is required, then the training dataset needs to be considerably larger. In this case the applicability of the method is limited to applications whereby (a) the surrogate model needs to be called at least thousands of times, or (b) results are to be delivered in quasi-real-time. (a) typically arises when studying all possible relative arrangements of micro-structural features within a structural component of fixed geometry, whilst (b) may arise when performing non-destructive testing from CT scans. Of course, it is always possible to revert to a strategy where the NN is only used to predict likely hot spots.

For application (ii), we have shown that introducing parameterised geometries vastly increases the complexity of the problem. Using global parameters leads to an exponential increase in training data requirements (the infamous curse of dimensionality). So, at the current stage of our understanding of the capabilities of deep learning algorithms, we recommend to either deploy this type of deep learning methodologies to predict stresses in random distributions of microscale pores over fixed macroscale geometries, or use a very small number of structural parameters.

Lastly, we have introduced an approach to enforce physics constraints at evaluation time. We choose to force the stress predictions delivered by the NN surrogate to satisfy the homogeneous Neumann conditions on the surface of the porous structure. This is done in a Bayesian setting. The Bayesian GNN predictions are considered as prior for this task, and posterior predictions that conform to the imposed Neumann conditions are computed via an ensemble Kalman update. We have used this method to address problems commonly reported with data-driven methodologies, namely lack of training data and poor extrapolation ability. We have shown that without this online correction, downward bias is observed for maximum equivalent stresses, which is problematic. This effect is alleviated thanks to the online physics-based correction.

Chapter 7

Centroid Graph Neural Network for the prediction of equivalent stress in 3D porous structures

7.1 Introduction

In the last two chapters, Convolutional Neural Networks (CNNs) operating on 2D structures and Graph Neural Networks (GNNs) operating on 3D structures were developed to circumvent the need for direct numerical analysis of stress simulations in porous materials. In this chapter we propose a lightweight model, based on the same multiscale approach, which operates on the centroids of the microscale features. This proposed model is in practice a GNN, referred to from now on as centroid GNN, in contrast to the GNN of [Chapter 6], referred to from now on as full GNN. We will demonstrate that the centroid GNN presents reduced memory and computational requirements compared to the full GNN. In practice this model can be trained on a laptop GPU and it reports accuracy comparable to the accuracy of the full GNN, trained on a GPU server, for specific cases. Conversely, there is a price that needs to be paid for this reduced computational cost. Specifically, this model does not have the ability to predict the stress distribution on the surface of the structure (since the surface mesh is discarded) and also it performs poorly in cases where the microscale features cannot be easily described with a few parameters (as for instance a sphere can in contrast to a defect of a random shape). To address the last issue, in this chapter we employ a Bayesian optimisation procedure with the purpose of trying to identify the defects with the highest

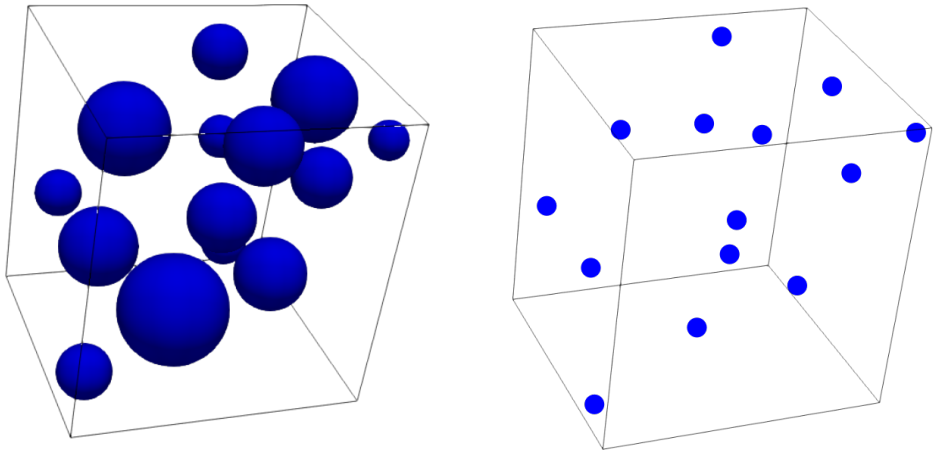
micro stress field in a dataset with multiple defects. We show that with only a few fine scale FE simulations we find the maximum stress in a dataset composed of hundreds of realistic defects. The purpose of this chapter is to show that

- A centroid GNN, trained on a laptop GPU, can be used to replace the expensive full GNN, trained on a GPU server, in specific cases.
- Transfer Learning can be applied to reduce the training data requirements, which is typically a major problem in applying NNs to realistic engineering applications.
- The Bayes by Backprop method can be used to extract meaningful uncertainty information from the centroid GNN, and thus a Bayesian NN can be obtained able to provide reliable stress predictions, as opposed to most state-of-the-art works in the relevant literature.
- The uncertainty information extracted from the Bayesian centroid GNN can be utilised by a Bayesian optimisation framework to identify the defects associated with the maximum stress and with only a few fine scale FE simulations the maximum stress in a dataset of hundreds of defects can be determined.

7.2 Input-Output

In contrast to the full GNN where we used patches from the FE mesh as input to the GNN, in the centroid GNN we convert the FE mesh to a point cloud, composed of the centroids of the microscale features, and use it as input. An example, using a realisation of the structures we used in section [6.4.1], but without ellipsoids, and with spheres of variable radius, can be found in [Fig 7.1].

This point cloud will be processed with a GNN. In order to allow message passing between the nodes (centroids) we create edges between the centroids. This results in a centroid network. Connecting each centroid with every other centroid creates a high computational load without significantly improving the predictive ability of the network. In this work we consider only the 3 nearest neighbours. We prove numerically in the next sections that this is enough to model the interactions between the centroids. On top of that, we only consider 2 defects as neighbours if the distance between their centroids is less than 5 times the maximum of their radii, which is in accordance with the assumptions we made in the previous chapters regarding the interaction length of the defects. Below we see an example of a centroid network [Fig 7.2].



(a) Surface of microscale features (b) Centroids

Figure 7.1: On the left (a) we can see the surface of the spherical microscale features. On the right (b) we can see the generated point cloud. Each point corresponds to a centroid from the spheres on the left.

For the input of the network, we utilise both node and edge features. Every node (centroid) represents the surface of one of the pores. The node features are the average macroscale stress on the surface of the pore each centroid represents along with the radius of the pore each centroid represents. On the edges of the network, we encode the relative position and the distance between the centroids. The network only predicts features on the nodes of the centroid network and thus it has no output edge features. The output node features are the maximum microscale Von Mises stress on the surface of the pore each centroid represents. A high-level visualisation of the workflow can be found in [Fig 7.3].

7.3 Loss function

The loss function used for the training of the deterministic networks in this chapter is the Mean Squared Error (MSE). Specifically, the GNN predicts the maximum microscale stress on the surface of each pore in the centroid network. Consequently, the loss

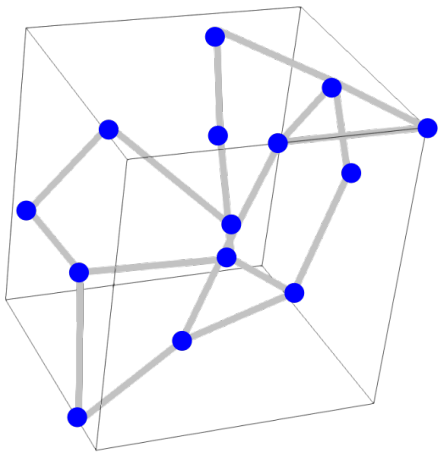


Figure 7.2: We can observe a centroid network, where we created edges between the centroids to allow for message passing between them.

function is the MSE between the maximum microscale Von Mises stress on the surface of each sphere as predicted by the GNN and the maximum microscale Von Mises stress on the surface of each sphere as calculated using direct FEA. For the Bayesian networks the ELBO loss is used, as described in equation [2.15].

7.4 Architecture

The architecture of the GNN is very similar to that described in section [6.2.5]. There are two main differences between the 2 GNN architectures.

- Number of Filters: The number of filters in the MLPs is here reduced from 128 to 64, since no improvement is reported when using a higher value
- Residual Connection: There is no residual connection connecting the input and output of the GNN

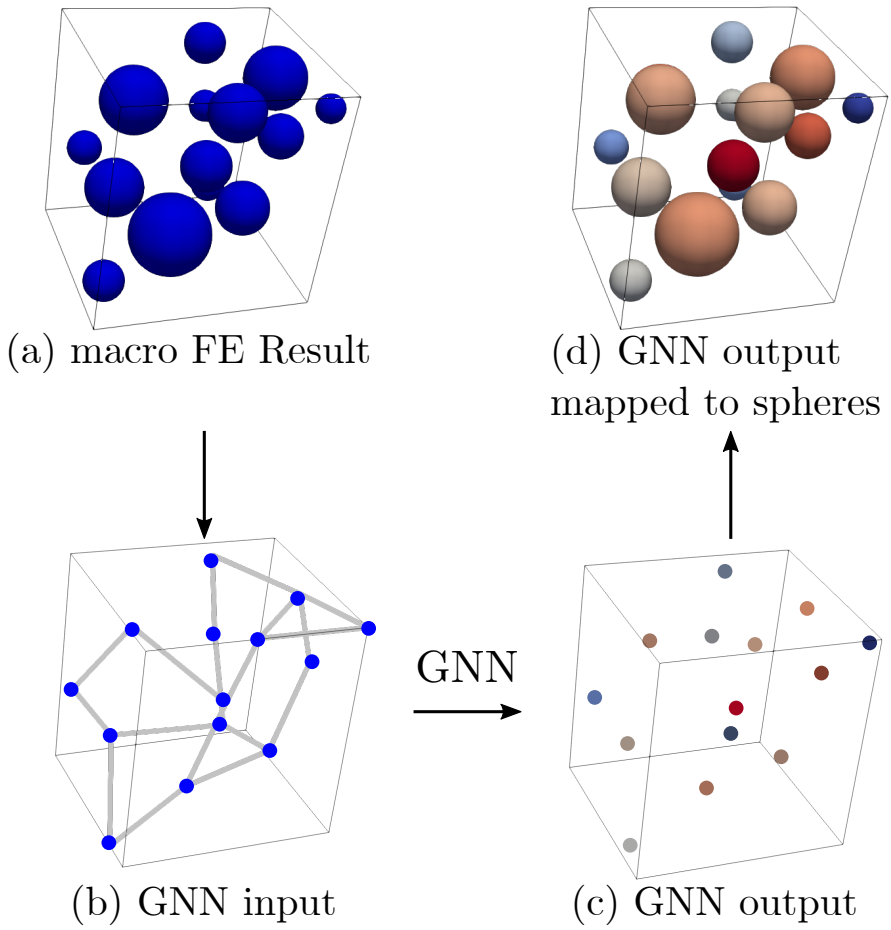


Figure 7.3: In (a) we see the output of the macroscale FE simulations. We can see that as expected the macroscale stress on the surface of the spheres is constant since these are ignored during the macroscale FE simulations. In (b) we see the centroid network, which is the input to the centroid GNN. On the edges of the network, we encode the relative position and the distance between the nodes. On the nodes of the centroid network, we encode the average macroscale stress and the radius of the spheres they represent. In (c) we see the output of the centroid GNN. The centroid GNN does not output any information on the edges, which are consequently not visualised. On the nodes of the centroid network, we see the maximum microscale stress on the surface of the spheres they represent. In (d) we can see, for visualisation purposes only, the maximum microscale stress, as predicted from the centroid GNN, on the surface of the spheres.

7.5 Bayesian Optimisation

Bayesian Optimisation is an iterative gradient free optimisation process commonly used to maximise a function that is computationally expensive to be evaluated. This function is usually referred to as objective function. For a set of points p, and an objective function f, the objective is to find $\mathbf{max}(f(p))$.

Bayesian Optimisation requires a probabilistic surrogate model, g, which is easy to evaluate, in contrast to the real function, f. This model puts a prior over the data p. In every iteration, i, the surrogate model is evaluated over all the points p. An acquisition function, which depends on the surrogate model, is used to indicate which point, $\hat{p_i}$, is more likely to maximise f. This is the point that maximises the acquisition function. Then the function f is evaluated at the indicated point $\hat{p_i}$ to determine its real value $f(\hat{p_i})$. This point is used to update the surrogate model using the Bayes Theorem. The updated model is a better approximation of the objective function. When the maximum of the acquisition function is less than the maximum of the observed points (objective function evaluations) the algorithm terminates. An example can be found in [Fig 7.4].

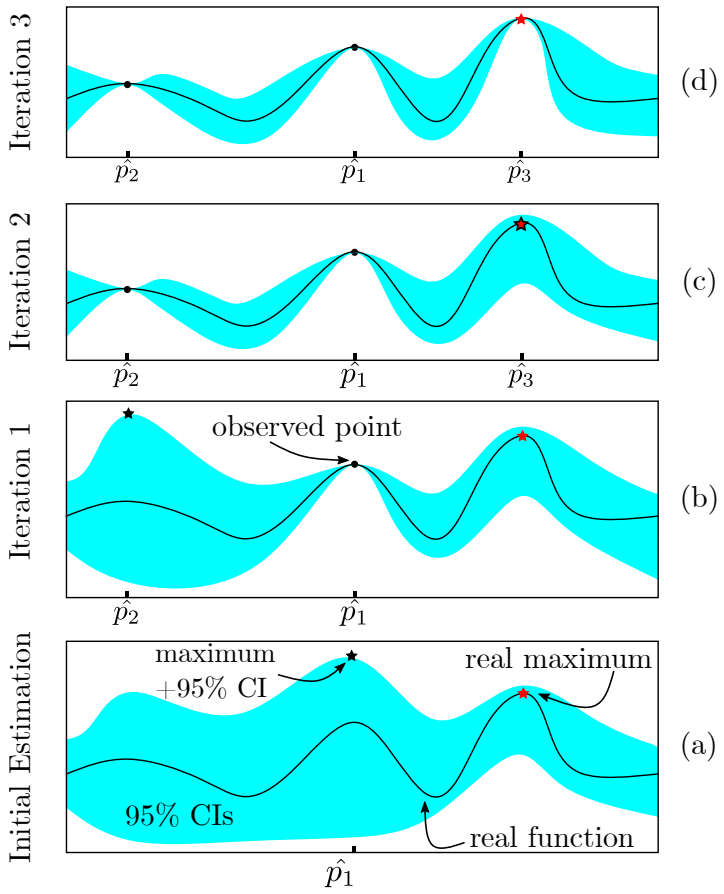


Figure 7.4: Sketch of a Bayesian optimisation procedure. The first figure (a) corresponds to the GNN prediction before the Bayesian optimisation. The black line corresponds to the objective function f that we want to maximise. We assume we have a surrogate model that can output a mean prediction and 95% CIs (blue area in the figure). The acquisition function used here is the +95% CIs of the prediction. The mean prediction of the surrogate model is not visualised here for simplicity. The acquisition function indicates the point $\hat{p_1}$ as the most probable to maximise f. The black star denotes the maximum of the acquisition function for the current iteration. The red star denotes the maximum of f. In the next figure (b) the real value $f(\hat{p_1})$ has been calculated and the surrogate model has been updated, which can be seen from the new uncertainty estimation. The uncertainty at point $\hat{p_1}$ is zero, since its real value has been calculated. There are other points that have +95% CIs higher than this value, so the algorithm continues. The acquisition function indicates the second point $\hat{p_2}$ and the same procedure is repeated in the third plot (c). In the last plot (d), we can see that the point that maximises the acquisition function, $\hat{p_3}$, corresponds to the point that maximises f. After calculating $f(\hat{p_3})$, we can see that the maximum of the acquisition function is less than $f(\hat{p_3})$ ($=$ **max**$[f(\hat{p_1}),f(\hat{p_2}),f(\hat{p_3})]$), and thus the algorithm terminates.

7.6 Numerical examples

In this section we show numerical results of the centroid GNN applied in different types of problems. Firstly, we use a synthetic dataset and numerically prove that the centroid GNN can be used to provide a very good estimation of the maximum micro stress field. We study the network's performance with respect to the size of the training dataset. Additionally, we train a Bayesian centroid GNN and evaluate its ability to provide meaningful uncertainty information. Moreover, we explore the idea of transfer learning by using a new synthetic dataset where the range of the radii of the spherical pores and the porosity of the samples is different compared to the first synthetic dataset. Lastly, we attempt to predict the maximum micro stress on a real dataset that contains both almost spherical and strongly non spherical defects. The centroid GNN is not appropriate for this task, nonetheless we show that the uncertainty extracted by the Bayesian GNN can be used in a Bayesian Optimisation procedure to identify which defects are more likely to be associated with the maximum stress.

7.6.1 Numerical examples with deterministic GNN

Training dataset

For the purpose of training a deterministic centroid GNN, we will work with a synthetic heterogeneous material from which we extract and examine cubical specimens. The size of each specimen is 2 units along each spatial dimension. The specimen has a distribution of at most 50 spherical pores with different radii. The radii of the spherical pores are sampled from a Gaussian distribution, $\mathcal{N}(0.25, 0.05)$. The spherical pores are generated without intersecting, by iterating over all the spheres from the largest to the smallest and rejecting the intersecting ones. For the macroscale simulations, input of the centroid BGNN, the spheres are ignored. Two realisations of the cubical specimen are shown in [Fig 7.5].

We apply homogeneous Dirichlet boundary conditions away from the material specimen, exactly as described in [6.4.1]

For data pre-processing we use a simple linear rescaling by dividing the input stress components with the maximum stress component in the training dataset, as we did in the full GNN case.

The dataset we use consists of 11,000 FE simulations (10,000 for training and 1,000 for testing) completed in 11.9 hours on 5 cores on an Intel® Xeon® Gold 6148 CPU @

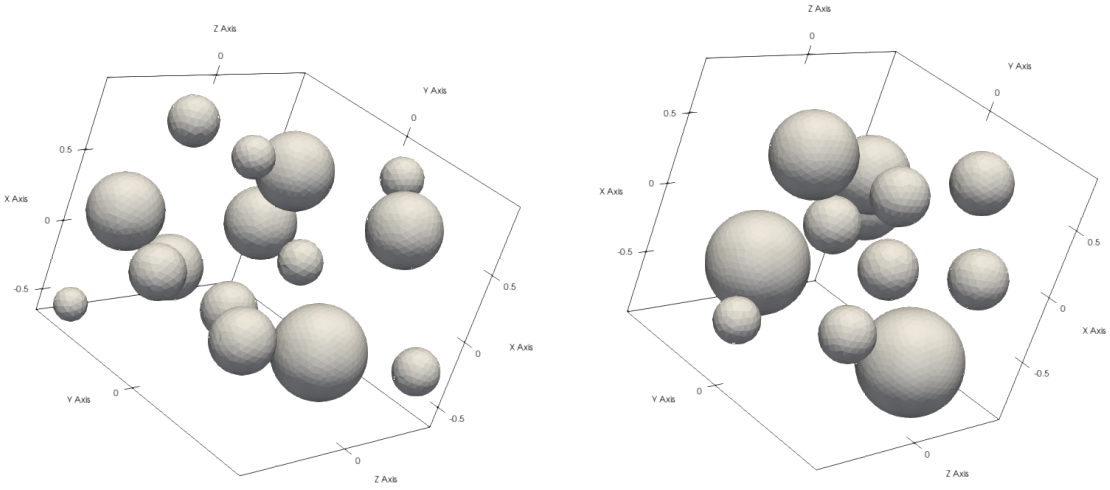


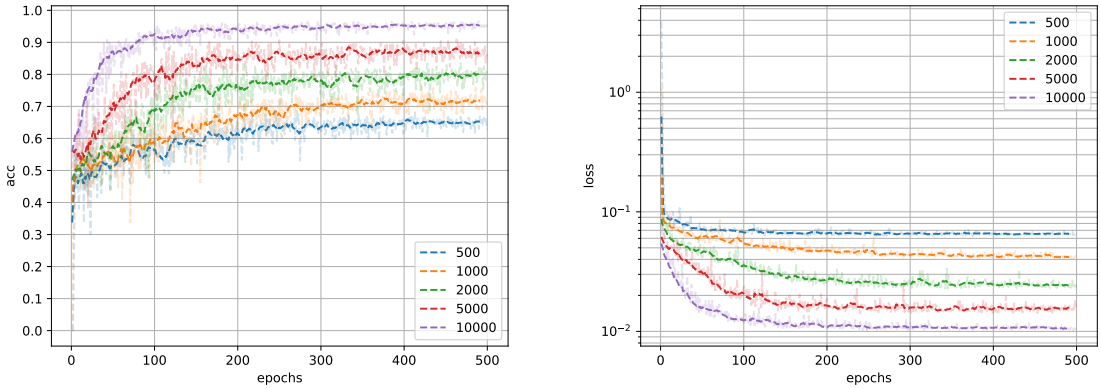
Figure 7.5: Two realisations of a cubical specimen from a heterogeneous material with a distribution of random spherical non-intersecting pores of different radii.

2.40GHz CPU.

For the training of the network we used a batch size of 32 and the Adam optimiser with an initial learning rate of $1 \cdot 10^{-4}$ decreasing by 5% every 50 epochs.

Centroid GNN prediction

We train 5 different models using 500, 1,000, 2,000, 5,000, 10,000 FE simulations and we use 1,000 FE simulations for the testing dataset. In [Fig 7.6] we observe the accuracy and the loss curves with respect to the size of the training dataset. The accuracy for the network trained with 10,000 data is 96% and its training requires 13 hours on an NVIDIA T1200 *laptop* GPU with 4GB of RAM. For comparison we remind the reader that for the full GNN, training using 180 data from the cubical Heterogeneous Material [6.4.1] requires 6 hours on an NVIDIA V100 *server* GPU with 16GB of RAM and training using 1000 data from the variable dimension dogbone dataset [6.4.4] requires 60 hours on the same GPU.



(a) Accuracy (b) Loss

Figure 7.6: In (a) we can observe the accuracy of the centroid GNNs using the 10% error threshold and in (b) the loss.

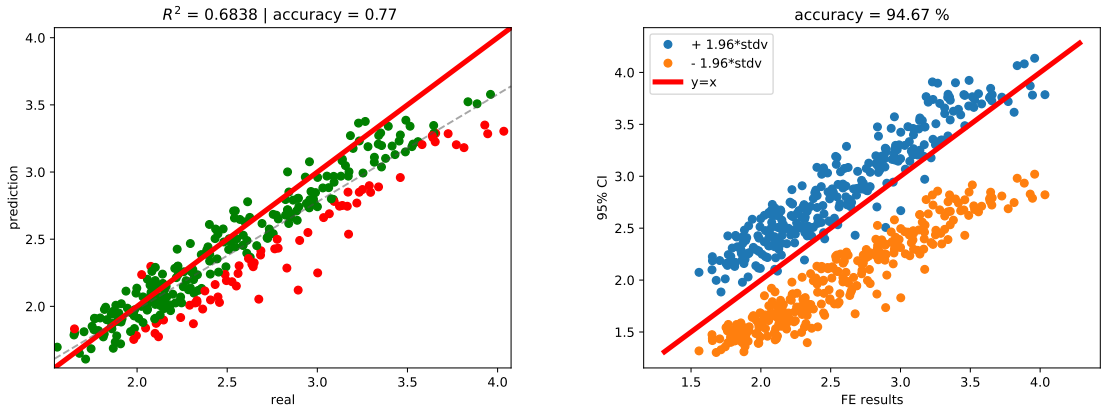
7.6.2 Numerical examples with probabilistic GNN

Training dataset

For the purpose of training a probabilistic centroid GNN we use the same dataset introduced in [7.6.1].

Centroid Bayesian GNN prediction

We train a Bayesian centroid GNN using 2,700 FE simulations as training set and 300 FE simulations as test set. The results corresponding to the mean prediction of the network can be found in [Fig 7.7a]. The accuracy is 77% for the 10% relative error threshold. Moreover, to evaluate the quality of the uncertainty of the network we need to examine the CIs of the prediction. In the following figure [Fig 7.7b] we observe the 95% CIs extracted from the network. We can see that for 95% of the data the real maximum is inside the upper and lower 95% CIs and thus we conclude that the network successfully determined the uncertainty of its prediction.



(a) mean prediction

(b) 95% CIs

Figure 7.7: In the 2 figures we see results from a Bayesian centroid GNN trained on 2,700 data and evaluated on 300 data from the dataset introduced in [7.6.1]. In (a) we can see the mean GNN prediction for the maximum micro Von Mises stress on the y axis and the one calculated by FE simulations on the x axis. The green dots correspond to data with less than 10% relative error and the red ones to data with more than 10%. The red line corresponds to the y = x line. In (b) we can see the 95% CIs of the prediction. The blue dots correspond to the +95% CIs and the orange ones to the -95% CIs. The red line corresponds to the y = x line.

7.6.3 Transfer Learning

Limited availability of training data is a common problem ML practitioners face. Even if enough data are available to train a NN on a specific family of inputs its performance significantly deteriorates when making predictions outside of this family of data. In this section we examine the idea of transfer learning to improve the network prediction on datasets for which we only have a limited number of data available for training.

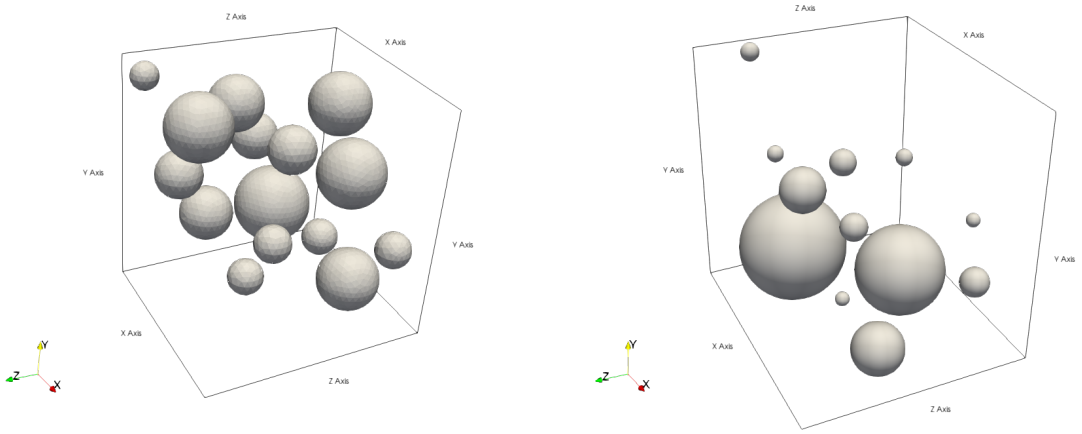
In the context of transfer learning, we define 2 datasets, namely the "initial" dataset and the "target" dataset. With the term "target" dataset we refer to the dataset of interest, the dataset that we want to make predictions on. Unfortunately, we have access to only a few samples from this dataset and we cannot train a NN solely based on these samples. On the other hand, with the term "initial" dataset we refer to another dataset that is similar to the "target" dataset and for which we have enough data to train a NN. The purpose of transfer learning is to use a NN trained on the "initial" dataset and calibrate/fine tune it with a few samples from the "target" dataset to make sensible predictions on the "target" dataset.

Datasets

The "initial" dataset used in this section is the one introduced in section [7.6.1]. The "target" dataset follows the same principles used for the "initial" dataset. The difference between them lays on the range of radii of the spherical defects and on the porosity of the samples. Specifically the "target" dataset has a broader range of radii and a lower porosity. In [Fig 7.8] we see a comparison between samples from the 2 datasets. We produce 300 samples from the "target" dataset; 80 are used for training and 220 for testing

Transfer Learning numerical experiment

First of all, we use the NN from the [7.6.1] section trained with 10,000 samples from the "initial" dataset to make a prediction on the "target" dataset. We observe that the accuracy dropped from 96% to 68%. Moreover, we attempt to train from scratch a NN using only the 80 training data available from the "target" dataset. As expected, the accuracy of such a network is even lower, specifically 40%. The optimum strategy is to use both the NN trained with the "initial" dataset and the 80 samples from the "target" dataset. Specifically, in each epoch we use a total of 160 training data, 80



(a) Patch from the "initial" dataset (b) Patch from the "target" dataset

Figure 7.8: Comparison between a patch from the initial dataset (a) and a patch from the target dataset (b).

of them correspond to the samples from the "target" dataset and the other 80 are randomly sampled from the "initial" dataset. The accuracy of this network is 73%, which is a 5% increase compared to not training at all and a 33% increase compared to training from scratch with only the samples from the "target" dataset. The above results are summarised in [Fig 7.9].

7.6.4 Bayesian Optimisation

In this section we use the centroid GNN to identify the maximum Von Mises stress in a dataset containing real defects. The centroid GNN is not appropriate for making sensible stress predictions in this dataset since the defects cannot be easily described with a few geometrical parameters. Instead, in this section we want to evaluate to what degree the uncertainty of the centroid GNN can be used to indicate which defects are associated with the maximum Von Mises stress. This uncertainty will be used in a Bayesian Optimisation framework where fine scale FE simulations will be performed in the neighbourhood of the indicated defects to calculate the real Von Mises stress.

Dataset

Eight datasets containing real defects are available to us, referred to as "real datasets" from now on. Each dataset contains between 100 and 200 defects. All the "real

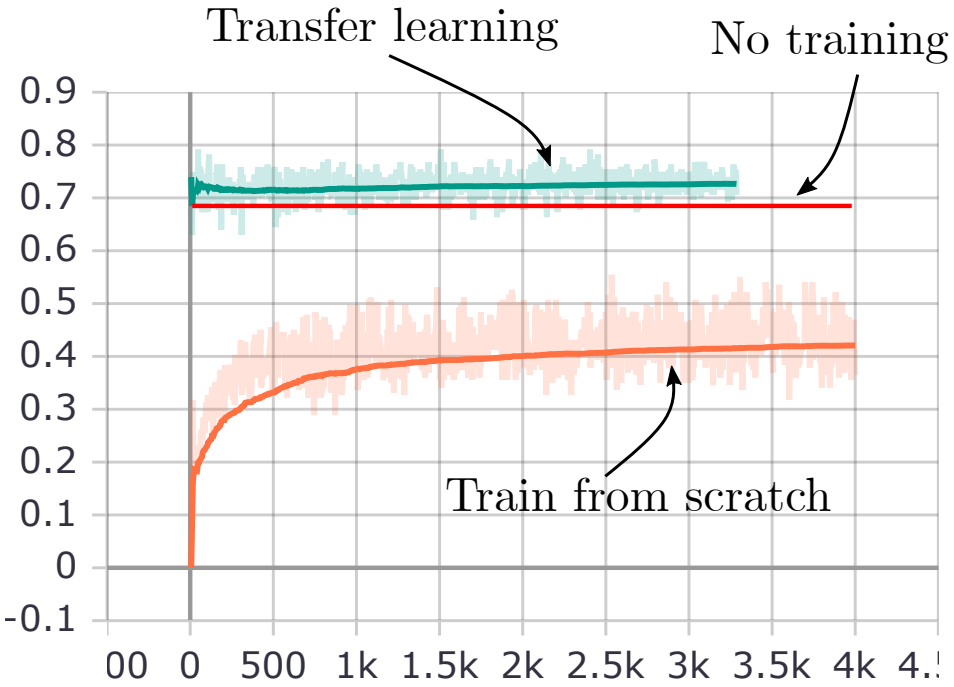


Figure 7.9: Transfer Learning using 2 synthetic datasets.

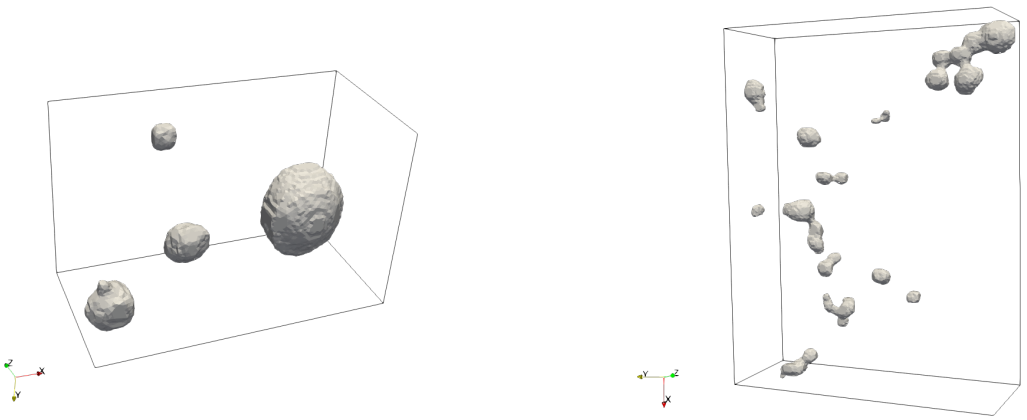
datasets" contain both almost spherical and strongly non spherical defects, [Fig 7.10].

Patches

Since the purpose of the Bayesian Optimisation framework that we develop is to find which defect is associated with the highest stress, we extract from each dataset one patch per defect. Specifically, to create the patches we iteratively consider each defect as central defect, and extract this defect and its neighbourhood. The neighbourhood of the central defect is defined as all the defects where the distance between their centroid and the centroid of the central defect is less than 3 times the maximum of their radii. This procedure is summarised in [Algorithm 2].

Sphericity

The defects in the "real datasets" cannot be described using primitive shapes, and thus a small number of geometrical parameters that can be easily defined and calculated. Instead, in this section we use the sphericity of the defects, which indicates how close a defect is to a sphere. In this work we use the following formula to calculate sphericity



(a) Almost spherical defects

(b) Strongly non spherical defects

Figure 7.10: In figure (a) we can see a subvolume containing almost spherical defects, while in figure (b) we see a subvolume containing strongly non spherical defects. The subvolumes are extracted from the real datasets.

$$sphericity = 6\sqrt{\pi}\frac{V}{S^{3/2}} \tag{7.1}$$

where V is the volume and S the surface area of the defect. Sphericity can take values from 0, for a strongly non spherical defect, to 1, for a perfect sphere.

Numerical examples with Bayesian Optimisation

The Bayesian Optimisation requires a probabilistic surrogate model and thus we train an initial BGNN for this purpose. Specifically, we follow exactly the same process we followed in section [7.6.2], with the same dataset, but we also use the sphericity as input node feature for every centroid. This dataset only contains spheres, so the sphericity is always going to be 1. To fine-tune this model, and allow it to try to learn how sphericity affects the stress field, we further train it using one of the 8 "real datasets" for 100 epochs. In this new dataset, that contains both spherical and non spherical defects, the sphericity will indicate to the network which defects are close to spheres, and which are not. In the former case, the network prediction is expected to be sensible since the network has been trained on spheres. In the latter case, the network is expected to output high uncertainty (in terms of 95% CIs) since non spherical defects were not present in the training dataset. This hypothesis is backed up by the numerical examples we have conducted. We summarise these findings in [Fig 7.11] where we see

Algorithm 2 Get central defect neighbourhood

```
function GET_NEIGHBOURHOOD(central_defect, defects)

    central_centroid = get_centroid(central_defect)
    neighbourhood = []

    for defect in defects do

        centroid = get_centroid(defect)          ▷ Get centroid for current defect

        centroid_distance = Distance(central_centroid, centroid)
        max_radius = max(central_defect.radius, defect.radius)

        if centroid_distance ≤ 3 · max_radius then  ▷ decide if the defects interact
            neighbourhood.append(defect)  ▷ add current defect to neighbourhood

    return neighbourhood
```

that the uncertainty of the network is higher for patches that contain strongly non spherical defects.

After obtaining the probabilistic surrogate model, we extract all the 196 patches from the second "real dataset" and perform Bayesian Optimisation as described in section [7.5]. We use the maximum 95% CIs extracted from the network as the acquisition function. In every iteration of the Bayesian Optimisation algorithm we retrain the network for 50 epochs. The results can be found in [Fig 7.12]. We observe that in 5 iterations the maximum Von Mises stress in the dataset was successfully identified. Specifically, we can see that in the second iteration there was an improvement on the estimation of the maximum Von Mises stress, followed by 2 iterations where there was no improvement, and finally in the fifth iteration, the maximum Von Mises stress indicated by the Bayesian Optimisation algorithm was the real maximum. A more in-depth explanation can be found in [Fig 7.13].

After showing in detail how the Bayesian Optimisation framework works on one "real dataset" we now apply it on all the remaining "real datasets" and show the total results in [Fig 7.14]. We can observe that in all but one "real datasets" the algorithm was able to find the real maximum in the dataset. For the dataset where the real maximum was not successfully identified the relative error between the predicted maximum and the real maximum is less than 5%. We can see that for most cases it takes less than

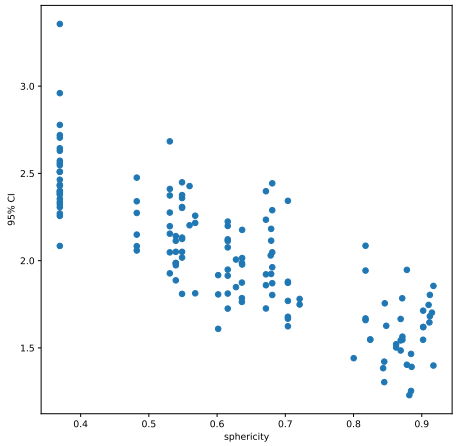


Figure 7.11: In this diagram each point corresponds to a patch from one of the "real datasets". Specifically, the x axis corresponds to the minimum sphericity in the patch and the y axis to the maximum 95% CIs in the patch.

10 iterations to terminate the algorithm, and the maximum reported iterations is 18, while all the datasets had more than 100 defects.

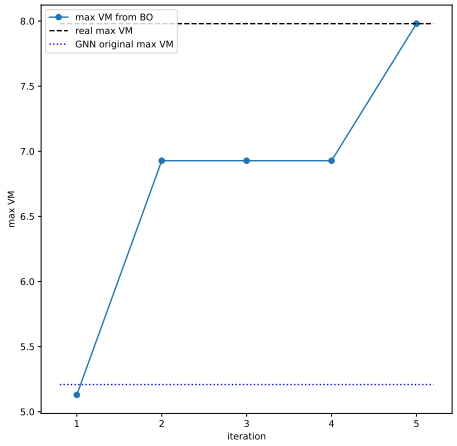


Figure 7.12: Results of the Bayesian Optimisation procedure on a "real dataset". In every iteration a new maximum Von Mises stress in the dataset is proposed, based on FEA analysis in the neighbourhood of indicated defects. The blue solid line corresponds to the maximum of the proposed Von Mises stresses up to the current iteration. The blue horizontal discontinuous line corresponds to the initial estimation for the maximum Von Mises stress from the network. The black horizontal discontinuous line corresponds to the real maximum Von Mises stress in the dataset.

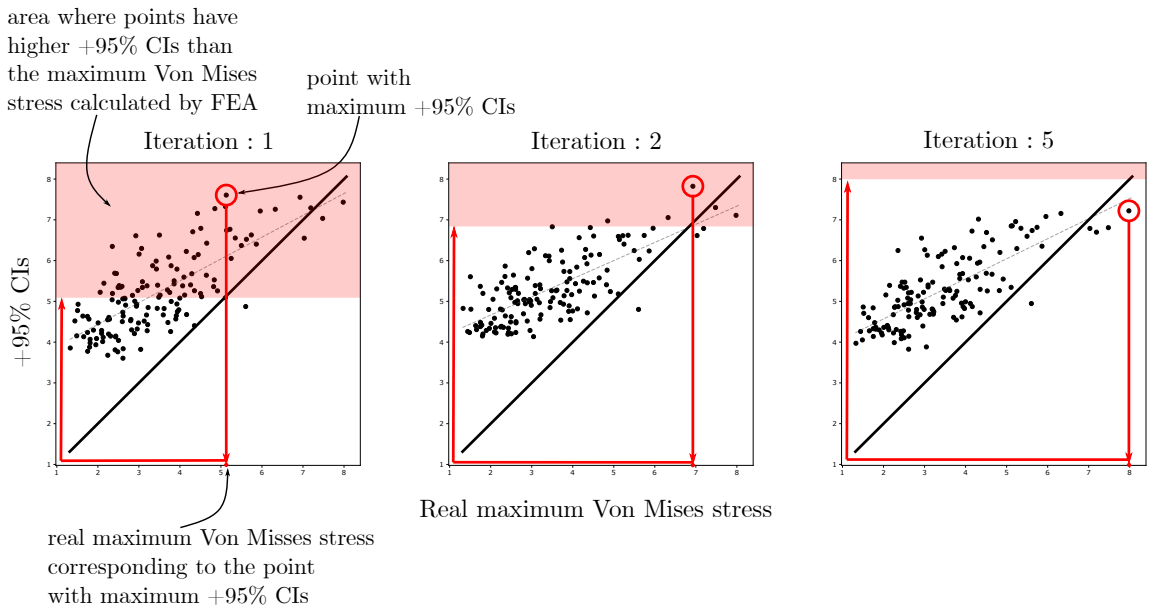


Figure 7.13: In this figure we observe 3 plots corresponding to the first, second and fifth iteration of the Bayesian Optimisation algorithm. In every plot, each point corresponds to one defect in the "real dataset". Specifically, the x axis corresponds to the maximum Von Mises stress as calculated by FEA on the neighbourhood of the defects. The y axis corresponds to the upper 95% CIs as calculated by applying the Bayesian centroid GNN on the neighbourhood of the defects. The black line is the "y=x" line. In the first plot, first iteration, based on the maximum 95% CIs, a defect is proposed as the one that can potentially have the maximum Von Mises stress in the dataset. After performing FEA on the neighbourhood of this defect and calculating the maximum Von Mises stress, we can see that there are plenty of other defects with upper 95% CIs higher than this maximum Von Mises stress (points in the red area), so the algorithm continues. The defect is added to the training dataset, and the network is retrained for 50 epochs. In the second figure, second iteration, we observe the new 95% CIs as predicted by the retrained GNN. The same procedure is followed, and FEA is performed in the neighbourhood of the newly indicated defect. The new maximum Von Mises stress is higher than the previous one and now considerably less defects have 95% CIs higher than the current maximum Von Mises stress. For the next two iterations, not included in this figure, there is no improvement in the maximum Von Mises stress. This means that the maximum Von Mises stresses calculated by FEA in the neighbourhood of the indicated defects were lower than the maximum Von Mises determined in the second iteration. Lastly, in the third figure, fifth iteration, we see that the maximum Von Mises stress calculated by FEA in the neighbourhood of the indicated defect is higher than the 95% CIs of all the other defects. At this point the algorithm terminates. This is the best estimation we can have for the maximum Von Mises stress in the dataset based on the uncertainty provided by the network and it turns out that indeed this is the real maximum in the dataset.

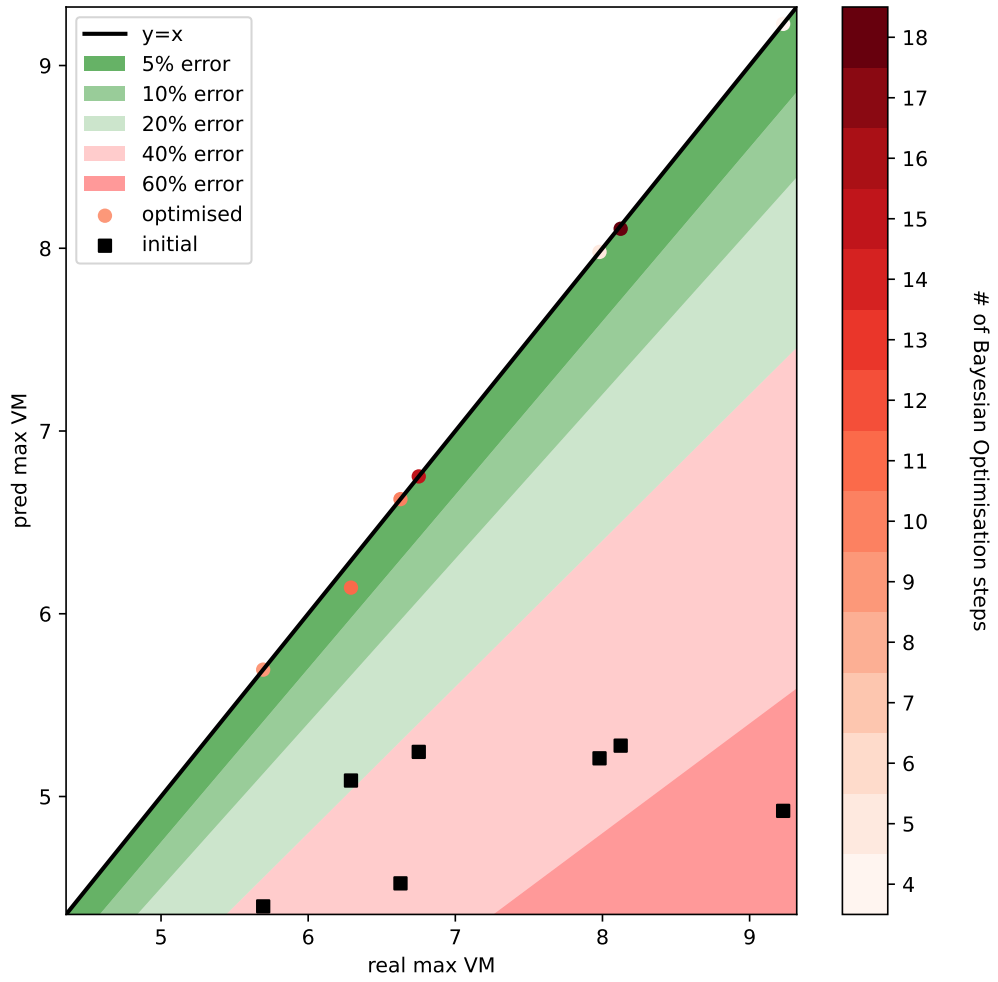


Figure 7.14: Results of the Bayesian Optimisation procedure on all the available "real datasets". The black line corresponds to the "y=x" line. Each point corresponds to one "real dataset". Specifically the x axis corresponds to the maximum Von Mises stress as calculated by FEA in the dataset. The y axis corresponds to the predicted maximum Von Mises stress. The square points correspond to the GNN prediction before the Bayesian Optimisation procedure while the circular points correspond to the output of the Bayesian Optimisation algorithm. Areas with different colours correspond to different relative errors between the predicted and real maximum Von Mises stress in the dataset. The colour of the circular points indicates how many steps of the Bayesian Optimisation algorithm were performed.

7.7 Conclusion and perspectives of this chapter

The goal of this chapter is to modify the 3D Graph Neural Network (GNN) introduced in [Chapter 6], to be applied instead of the entire mesh of the multiscale structure to the centroids of the microscale features and to ultimately create a lightweight 3D GNN surrogate model to circumvent the need for 3D direct numerical analysis of stress simulations in porous materials and structures.

We show that this GNN can be trained in a few hours on a laptop GPU, in contrast to the GNN introduced in [Chapter 6] that requires days of training on a server GPU. Additionally, we show that the accuracy reported from this model is comparable to the accuracy of the full GNN, and it can reach a value of 96% given enough data.

In order to tackle the problem of limited training data we utilised a transfer learning framework to transfer knowledge gained from one synthetic dataset, "initial dataset", to a new synthetic dataset, "target dataset". We show that the transfer learning approach outperforms both training from scratch a GNN with limited samples from the "target dataset" and directly applying the GNN trained with samples from the "initial dataset" to the "target dataset".

As done in the full GNN, we showed that we can extract uncertainty information from the centroid GNN. In 95% of the test data the real value was inside the 95% CIs, implying that the Bayesian GNN successfully identified the error in its predictions. Additionally, the upper 95% CIs of the Bayesian centroid GNN were used as an acquisition function in a Bayesian Optimisation framework. Specifically, we managed by performing a small number of FE simulations, 4 to 18 depending on the dataset, to identify the maximum Von Mises stress in datasets with, 100 to 200, real defects, of random shape.

Chapter 8

Conclusions

8.1 Problem statement

Multiscale structures are common in a wide range of engineering applications. Nonetheless, simulating them using classical Finite Element Analysis (FEA) is challenging. This is mainly attributed to the very dense mesh that needs to be used around the fine scale features to properly model their behaviour, leading to increased computational cost.

To tackle this problem, multiscale techniques have been developed. These are usually divided in homogenisation and concurrent scale analysis techniques. In both cases, the problem is split into a macroscale problem, which does not take into account the existence of fine scale features, and a microscale problem, which does. Homogenisation, that is computationally less intensive, assumes that the macroscale gradient varies slowly across the structure. This is commonly violated in the real applications due to the existence of sharp macroscale features. In these cases, concurrent multiscale modelling needs to take place that is computationally challenging and practically more intrusive.

8.2 Proposed approach and contribution to state of the art

In this thesis we develop FE surrogate models for stress predictions in multiscale structures exhibiting spatially random microscopic features. We propose an end-to-end Neural Network (NN) based framework, that in contrast to traditional surrogate model tech-

niques like Gaussian Processes is able to operate on high dimensional data like images and graphs. The proposed NN framework takes as input an inexpensively calculated FE solution that ignores the existence of microscale features, and outputs corrections to the macroscale stress field, which take into account the existence of the microscale features.

In contrast to the state-of-the-art NN based FE surrogate models, our networks are able to accurately predict the effect of fine scale features on the global stress field in cases where multiple fine scale features are interacting with each other and the boundaries of the structure.

Another gap in the literature that this thesis attempts to cover is the adoption of Bayesian NNs (BNNs) over deterministic ones. Most of the works found in the relevant literature tend to be deterministic, and thus their predictions cannot be trusted. In this thesis we use the Bayes By Backprop method to convert the aforementioned deterministic NNs to BNNs that are able to quantify the uncertainty of their predictions.

Lastly, we explore different techniques that can be utilised to reduce the training data requirements, something that a lot of researchers have reported as a blocker in the adoption of ML techniques in a variety of engineering fields. To this end we adopt standard techniques from the computer vision domain like data augmentation and transfer learning and show how they can be used in the context of regression NNs to efficiently train models with limited number of data. Additionally, we utilise the uncertainty of the BNNs to examine two more techniques that can help with the adoption of NNs in more engineering fields. The first technique is the Ensemble Kalman method, which is being used to enforce online physics-based corrections to the output of the network. We show that this leads to improved results in cases where data is limited. To the best of our knowledge this has not been explored in the relevant literature. The last technique we utilise is selective learning. This technique is popular in fields like medical image segmentation and is being adopted both by industry and academia. In this thesis, we show how we can use selective learning in the context of regression NNs, to reduce the need for labelled training data, which tend to be very expensive to acquire.

8.3 Results

8.3.1 Neural Networks for fast stress predictions in multiscale structures

Regarding the objective of creating NN based multiscale models for fast stress predictions in porous materials and structures, three such models were created operating on 2D image data (Chapter [5]), 3D graph data representing the surface of the porous structure of interest (Chapter [6]) and centroids of microscale features (Chapter [7]). All of the models were extensively tested and the conducted numerical experiments demonstrated good agreement between the model prediction and the FE results, for data in the training data distribution. Typically, the accuracy of the networks, depending on the examined case, ranges from 70% to 96%. In this thesis we define accuracy in a way that serves the purpose of accurately predicting maximum equivalent stresses (Tresca or Von Mises stress). Accuracy of 70% means that in 70% of the data the maximum equivalent stress is predicted with a relative error less than the threshold of 10%, unless stated otherwise. In the next three paragraphs we recapitulate our main findings regarding each of these networks.

The CNN model (Chapter [5]) was tested on linear and nonlinear 2D elasticity problems. For the linear elasticity case, the examined structures are square plates with a number of ellipses as macroscale features and random distributions of disks as microscale features. The accuracy varies from 80% to 96% using 450 FE simulations as training data, depending on the complexity of the examined dataset. For the nonlinear elasticity case, a rectangular with a single macroscale feature of random size and position, and random distributions of disks as microscale features is being examined. The reported accuracy is 73% for a CNN trained using 200 FE simulations as training dataset. Our experiments show that the CNN generalises well in new realisations of known macroscale structures and microscale distributions, but the performance quickly deteriorates when faced with microscale features not present in the training dataset, (e.g., when ellipses are present in the testing dataset whereas only disks are present in the training dataset).

The full GNN model (Chapter [6]) was tested in three 3D linear elasticity problems. The first dataset consists of 200 FE simulations of a cubical heterogeneous material with two elliptical pores as macroscale features, with random position, size and orientation and a distribution of 50 to 100 spherical pores with the same radius as microscale features that are not allowed to intersect with each other or the boundaries of the struc-

ture. The reported accuracy is 96%. The second and third datasets are devoted to a specimen compatible with uniaxial tensile testing that we refer to as dogbone. For the second dataset the dimension of the dogbone remains constant in all FE simulations, this also applies to the macroscale feature which is a cylinder positioned in the middle of the structure. The microscale features are represented by a random distribution of 50 to 100 spheres that are allowed to intersect with each other and the boundaries of the geometry, something that greatly increases the number of possible configurations and thus the difficulty to explore this space. The experiments have shown that relatively few simulations were required to train the GNN to an insightful level of accuracy. Typically, training the GNN with 50 direct numerical simulations yields an accuracy of 80% for the 20% relative error threshold, whilst training it with 800 FE simulations gives an accuracy of 80% for the 15% relative error threshold. However, the experiments also show that in applications where a lower error threshold is required, then the training dataset needs to be considerably larger. In the third dataset, the dimension of the dogbone and the number and size of the macroscale features change in every FE simulation. This increases even further the difficulty to create a representative dataset while keeping its size reasonable (curse of dimensionality). The reported accuracy using a dataset of 1,000 FE simulations is 70% for the 15% error threshold. So, at the current stage of our understanding of the capabilities of deep learning algorithms, we recommend to either deploy this type of deep learning methodologies to predict stresses in random distributions of microscale pores over fixed macroscale geometries, or use a very small number of structural parameters.

Lastly, for the centroid GNN (Chapter [7]) we examine a dataset consisting of cubical specimens with no macroscale features and with up to 50 spherical pores with different radii as microscale features. We show that the accuracy of this model is comparable to the accuracy of the full GNN, and it can reach a value of 96% given enough data. The advantage of this network is that it requires significantly less computational resources and can in practice be trained on a laptop GPU in contrast to the full GNN that would definitely require a server GPU for the turnaround times to remain practical. On the other hand, it requires significantly more data.

8.3.2 Reliable Neural Network stress predictions

In the context of reliable predictions, all the 3 models are Bayesian. These models are able to output credible intervals (CIs) that can be used to warn the user in cases where

the network is not certain about the prediction. Numerical experiments confirmed that the CIs where broader in cases where the error of the prediction was higher.

In [Chapter 5] we highlight the importance of the empirical Bayes, where we show that a Bayesian NN (BNN) that was trained by updating both the prior and posterior distribution of weights better explains the error in the prediction compared to a BNN where only the posterior is updated. Furthermore, in [Chapter 5] and [Chapter 6] we can see that the BNNs are able to quantify the error in their predictions in the case where they face microscale features outside of the training distribution. Lastly, in [Chapter 7] we present an example where a GNN is trained with perfectly spherical defects and then it performs inference on a dataset with real defects. We show that as the sphericity decreases (defects look less like spheres) the uncertainty of the prediction increases and thus the user is informed that the prediction of the network is less reliable.

8.3.3 Data requirements and generalisation ability

A number of different techniques were used to address the issue of limited training data and to increase the generalisation ability of the network. In [Chapter 5], we used mechanically consistent rotations as data augmentation technique to generate more training data without needing to perform additional FE simulations. We show that when we apply this data augmentation scheme to one quarter of the dataset, the trained CNN is able to outperform the CNN trained with the entire dataset. Furthermore, to reduce the large computational cost associated with the creation of labelled data (i.e. multiscale FEA simulations) in [Chapter 5] we used selective learning to choose and label only the data that contains new information for the network, leading to an up to 50% decrease in labelled data requirements. Additionally, numerical experiments showed that the online stress correction technique developed in [Chapter 6] led to a 10% increase in the accuracy of the model when applied to an under-trained network. Lastly, in [Chapter 7] a transfer learning approach was used to transfer knowledge gained from one synthetic dataset to a new synthetic dataset resulting in 30% increase in accuracy compared to training a network from scratch.

8.4 Limitations

Extrapolation: In all the 3 different types of models that we trained, we confirmed, the well-known fact, that the extrapolation ability of NNs is rather poor. Techniques were developed in the context of this thesis to warn the user about the uncertainty of the prediction. Nonetheless, one major takeaway from this work is that since the domain of applicability of NNs is limited, ML practitioners interested in training NNs for engineering applications should restrict the application domain of the NN to relatively narrow families of boundary value problems.

Locality: In the context of this work we assumed that the local macroscale fields that we use as input to the NNs are sufficient to predict the microscale stress, for all the structural problems over which training is performed. This is an assumption of locality that cannot in general be used in the context of non-diffusive problems (e.g. wave propagation, crack propagation).

8.5 Future work

Time dependent problems: Many problems arising in computational mechanics are history dependent, for instance elastoplasticity or viscoelasticity. There are examples in literature of both CNNs and GNNs being used to make predictions on time dependent problems. Nonetheless, this was outside of the scope of the current thesis. This is a future direction we are interested in exploring.

Volume based simulations: In the context of the current thesis we tackled 3D problems by utilising GNNs to perform geometric learning on a graph representing the surface of the examined structures. This is possible because the porous structures we examined could be fully described by their surface. This is not always the case, for instance structures with heterogeneous material properties. In this case, GNNs could still be used, operating on a graph representing the volume mesh of interest. Computational gain compared to voxel-based CNNs is still expected, since GNNs can operate on a variable resolution unstructured mesh in contrast to CNNs that require a constant resolution structured mesh. In the context of this thesis we did not examine such structures, but it is an interesting direction to explore.

Physics Informed Neural Networks: In Physics Informed Neural Networks (PINNs) physical constraints are enforced during the training stage. In this work we do not follow this paradigm since we want to have a general framework that is agnostic to the physical laws. Nonetheless, in cases where the physical laws are known the penalisation of the NN divergence from these laws can lead to a decrease in the size of the training dataset, by constricting the space of possible solutions.

Bibliography

Al-Dirini, R. M. A., Martelli, S., and Taylor, M. (2020). Computational efficient method for assessing the influence of surgical variability on primary stability of a contemporary femoral stem in a cohort of subjects. *Biomechanics and Modeling in Mechanobiology*, 19(4):1283–1295. 2

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. arXiv preprint arXiv:1607.06450. 113

Barrault, M., Maday, Y., Nguyen, N. C., and Patera, A. T. (2004). An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations. *Comptes Rendus Mathematique*, 339(9):667–672. 54

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks. arXiv preprint arXiv:1806.01261. 3, 30, 47, 48, 112, 113

Bessa, M., Bostanabad, R., Liu, Z., Hu, A., Apley, D. W., Brinson, C., Chen, W., and Liu, W. (2017). A framework for data-driven analysis of materials under uncertainty: Countering the curse of dimensionality. *Computer Methods in Applied Mechanics and Engineering*, 320:633–667. 55

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., USA. 69

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. In *Proceedings of the 32nd International Conference on In-*

ternational Conference on Machine Learning - Volume 37, ICML'15, page 1613–1622. JMLR.org. 5, 34, 35

Bruna, J., Zaremba, W., Szlam, A., and Lecun, Y. (2014). Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*. 28

Cardoso, M. J., Li, W., Brown, R., Ma, N., Kerfoot, E., Wang, Y., Murrey, B., Myronenko, A., Zhao, C., Yang, D., Nath, V., He, Y., Xu, Z., Hatamizadeh, A., Myronenko, A., Zhu, W., Liu, Y., Zheng, M., Tang, Y., Yang, I., Zephyr, M., Hashemian, B., Alle, S., Darestani, M. Z., Budd, C., Modat, M., Vercauteren, T., Wang, G., Li, Y., Hu, Y., Fu, Y., Gorman, B., Johnson, H., Genereaux, B., Erdal, B. S., Gupta, V., Diaz-Pinto, A., Dourson, A., Maier-Hein, L., Jaeger, P. F., Baumgartner, M., Kalpathy-Cramer, J., Flores, M., Kirby, J., Cooper, L. A. D., Roth, H. R., Xu, D., Bericat, D., Floca, R., Zhou, S. K., Shuaib, H., Farahani, K., Maier-Hein, K. H., Aylward, S., Dogra, P., Ourselin, S., and Feng, A. (2022). Monai: An open-source framework for deep learning in healthcare. arXiv preprint arXiv:2211.02701. 6

Cheng, X., Li, X., Yang, J., and Tai, Y. (2018). Sesr: Single image super resolution with recursive squeeze and excitation networks. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 147–152. 17, 67

Croom, B. P., Berkson, M., Mueller, R. K., Presley, M., and Storck, S. (2022). Deep learning prediction of stress fields in additively manufactured metals with intricate defect networks. *Mechanics of Materials*, 165:104191. 6

Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In *Advances in neural information processing systems*, pages 3844–3852. 28

Deshpande, S., Bordas, S. P. A., and Lengiewicz, J. (2022a). Magnet: A graph u-net architecture for mesh-based simulations. arXiv preprint arXiv:2211.00713. 3, 4, 5, 48

Deshpande, S., Lengiewicz, J., and Bordas, S. P. (2022b). Probabilistic deep learning for real-time large deformation simulations. *Computer Methods in Applied Mechanics and Engineering*, 398:115307. 2, 4, 5

Deshpande, S., Sosa, R. I., Bordas, S. P. A., and Lengiewicz, J. (2023). Convolution, aggregation and attention based deep neural networks for accelerating simulations in mechanics. *Frontiers in Materials*, 10. 5, 48

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1050–1059. JMLR.org. 34

Gal, Y., Islam, R., and Ghahramani, Z. (2017). Deep bayesian active learning with image data. arXiv preprint arXiv:1703.02910. 6, 92

Ghanem, R. G. and Spanos, P. D. (1991). *Stochastic Finite Elements: A Spectral Approach*. Springer New York, NY. 41, 54

Goetz, A., Durmaz, A., Müller, M., Thomas, A., Britz, D., Kerfriden, P., and Eberl, C. (2022). Addressing materials' microstructure diversity using transfer learning. *npj Computational Materials*, 8:27. 6

Gong, S., Bahri, M., Bronstein, M. M., and Zafeiriou, S. (2020). Geometrically principled connections in graph neural networks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11412–11421, Los Alamitos, CA, USA. IEEE Computer Society. 25

Goury, O., Amsallem, D., Bordas, S., Liu, W., and Kerfriden, P. (2016). Automatised selection of load paths to construct reduced-order models in computational damage micromechanics: from dissipation-driven random selection to bayesian optimization. *Computational Mechanics*, 58. 54, 55

Graves, A. (2011). Practical variational inference for neural networks. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 2348–2356, Red Hook, NY, USA. Curran Associates Inc. 33, 34

Guo, K. and Buehler, M. J. (2020). A semi-supervised approach to architected materials design using graph neural networks. *Extreme Mechanics Letters*, 41:101029. 5, 45

Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S., and Cohen-Or, D. (2019). Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38:1 – 12. 24, 26

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034. 120

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)*, pages 770–778, Los Alamitos, CA, USA. IEEE Computer Society. 17, 114

Hennigh, O., Narasimhan, S., Nabian, M. A., Subramaniam, A., Tangsali, K., Fang, Z., Rietmann, M., Byeon, W., and Choudhry, S. (2021). Nvidia simnet™: An ai-accelerated multi-physics simulation framework. In Paszynski, M., Kranzlmüller, D., Krzhizhanovskaya, V. V., Dongarra, J. J., and Sloot, P. M., editors, *Computational Science – ICCS 2021*, pages 447–461, Cham. Springer International Publishing. 7, 50

Hesthaven, J., Zhang, S., and Zhu, X. (2015). Reduced basis multiscale finite element methods for elliptic problems. *SIAM Journal on Multiscale Modeling and Simulation*, 13:316–337. 1, 54

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580. 34, 113

Hinton, G. E. and van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, COLT '93, page 5–13, New York, NY, USA. Association for Computing Machinery. 33, 34

Hochreiter, S., Bengio, Y., and Frasconi, P. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In Kolen, J. and Kremer, S., editors, *Field Guide to Dynamical Recurrent Networks*. IEEE Press. 17

Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-excitation networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7132–7141. 17, 67

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167. 43

Islam, R. (2016). Active learning for high dimensional inputs using bayesian convolutional neural networks. PhD. dissertation, Dept. Eng., Univ. Cambridge, Cambridge, U.K. 6, 94

Jaegle, A., Borgeaud, S., Alayrac, J.-B., Doersch, C., Ionescu, C., Ding, D., Koppula, S., Zoran, D., Brock, A., Shelhamer, E., Hénaff, O., Botvinick, M. M., Zisserman, A., Vinyals, O., and Carreira, J. (2022). Perceiver io: A general architecture for structured inputs & outputs. arXiv preprint arXiv.2107.14795. 48

Jiang, H., Nie, Z., Yeo, R., Farimani, A. B., and Kara, L. B. (2021). StressGAN: A Generative Deep Learning Model for Two-Dimensional Stress Distribution Prediction. *Journal of Applied Mechanics*, 88(5). 051005. 2, 4, 5, 45

Kerfriden, P., Allix, O., and Gosselet, P. (2009). A three-scale domain decomposition method for the 3d analysis of debonding in laminates. *Computational Mechanics*, 44:343–362. 1, 54

Kim, J., Lee, J. K., and Lee, K. M. (2016). Deeply-recursive convolutional network for image super-resolution. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1637–1645. 17, 114

Kingma, D. P., Salimans, T., and Welling, M. (2015). Variational dropout and the local reparameterization trick. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc. 5, 47

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114. 5, 34, 35

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907. 46

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc. 17

Krokos, V., Bui Xuan, V., Bordas, S., Young, P., and Kerfriden, P. (2021). A bayesian multiscale cnn framework to predict local stress fields in structures with microscale features. *Computational Mechanics*, pages 1–34. 5, 108

Lei, H., Akhtar, N., and Mian, A. (2021). Picasso: A cuda-based library for deep learning over 3d meshes. arXiv preprint arXiv:2103.15076. 24

LeNail, A. (2019). Nn-svg: Publication-ready neural network architecture schematics. *Journal of Open Source Software*, 4(33):747. 13

Li, H., Kafka, O., Gao, J., Yu, C., Nie, Y., Zhang, L., Tajdari, M., Tang, S., Li, G., Tang, S., Cheng, G., and Liu, W. (2019). Clustering discretization methods for generation of material performance databases in machine learning and design optimization. *Computational Mechanics*, 64. 56

Li, X., Wu, J., Lin, Z., Liu, H., and Zha, H. (2018). Recurrent squeeze-and-excitation context aggregation net for single image deraining. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 17, 67

Liang, L., Minliang, L., Caitlin, M., and Wei, S. (2018). A deep learning approach to estimate stress distribution: a fast and accurate surrogate of finite-element analysis. *Journal of the Royal Society, Interface*, 15:138. 2, 42

Lim, B., Son, S., Kim, H., Nah, S., and Lee, K. M. (2017). Enhanced deep residual networks for single image super-resolution. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1132–1140. 17, 114

Lino, M., Cantwell, C., Bharath, A. A., and Fotiadis, S. (2021). Simulating continuum mechanics with multi-scale graph neural networks. arXiv preprint arXiv:2106.04900. 47

Liu, Z., Bessa, M., and Liu, W. K. (2016). Self-consistent clustering analysis: An efficient multi-scale scheme for inelastic heterogeneous materials. *Computer Methods in Applied Mechanics and Engineering*, 306:319–341. 55

Lookman, T., Balachandran, P. V., Xue, D., and Yuan, R. (2019). Active learning in materials science with emphasis on adaptive sampling using uncertainties for targeted design. *npj Computational Materials*, 5(11):1–17. 6

Masci, J., Boscaini, D., Bronstein, M. M., and Vandergheynst, P. (2015). Geodesic convolutional neural networks on riemannian manifolds. In *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, pages 832–840, Los Alamitos, CA, USA. IEEE Computer Society. 25

Meister, F., Passerini, T., Mihalef, V., Tuysuzoglu, A., Maier, A., and Mansi, T. (2018). Towards fast biomechanical modeling of soft tissue using neural networks. arXiv:1812.06186. 42

Mendizabal, A., Márquez-Neila, P., and Cotin, S. (2020a). Simulation of hyperelastic materials in real-time using deep learning. *Medical Image Analysis*, 59:101569. 2, 4, 5, 44

Mendizabal, A., Márquez-Neila, P., and Cotin, S. (2020b). Simulation of hyperelastic materials in real-time using deep learning. *Medical Image Analysis*, 59:101569. 3

Mylonas, C., Tsialiamanis, G., Worden, K., and Chatzi, E. N. (2022). Bayesian graph neural networks for strain-based crack localization. In Madarshahian, R. and Hemez, F., editors, *Data Science in Engineering, Volume 9*, pages 253–261, Cham. Springer International Publishing. 4, 5, 47, 114, 191

Nie, Z., Jiang, H., and Kara, L. B. (2019). Stress field prediction in cantilevered structures using convolutional neural networks. *Journal of Computing and Information Science in Engineering*, 20(1). 2, 4, 5, 43, 45, 66

Niu, S., Bellala, V., Qureshi, D. A., and Srivastava, V. (2023). A machine learning method to characterize the crack length and position in high-density polyethylene using ultrasound. arXiv preprint arXiv:2304.11497. 5

Oden, J., Prudhomme, S., Romkes, A., and Bauman, P. (2006). Multiscale modeling of physical phenomena: Adaptive control of models. *SIAM Journal on Scientific Computing*, 28(6):2359–2389. 1, 54

Opper, M. and Archambeau, C. (2009). The Variational Gaussian Approximation Revisited. *Neural Computation*, 21(3):786–792. 5

Ozdemir, F., Peng, Z., Fuernstahl, P., Tanner, C., and Goksel, O. (2021). Active learning for segmentation based on bayesian sample queries. *Knowledge-Based Systems*, 214:106531. 6

Paladim, D., Almeida, J., Bordas, S., and Kerfriden, P. (2016). Guaranteed error bounds in homogenisation: an optimum stochastic approach to preserve the numerical separation of scales. *International Journal for Numerical Methods in Engineering*, 110. 1, 54

Penido, R. E.-K., da Paixão, R. C. F., Costa, L. C. B., Peixoto, R. A. F., Cury, A. A., and Mendes, J. C. (2022). Predicting the compressive strength of steelmaking slag concrete with machine learning – considerations on developing a mix design tool. *Construction and Building Materials*, 341:127896. 6

Perera, R., Guzzetti, D., and Agrawal, V. (2022). Graph neural networks for simulating crack coalescence and propagation in brittle materials. *Computer Methods in Applied Mechanics and Engineering*, 395:115021. 4, 5, 48

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. W. (2021). Learning mesh-based simulation with graph networks. International Conference on Learning Representations; arXiv preprint arXiv:2010.03409. 3, 4, 5, 47, 114, 191

Pilkey, W. and Pilkey, D. (2008). Peterson's stress concentration factors, third edition. *Peterson's Stress Concentration Factors, Third Edition*, pages 1–522. 68

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017a). Pointnet: Deep learning on point sets for 3d classification and segmentation. Conference on Computer Vision and Pattern Recognition (CVPR) 2017; arXiv preprint arXiv: 1612.00593. 25

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017b). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5105–5114, Red Hook, NY, USA. Curran Associates Inc. 25

Raghavan, P. and Ghosh, S. (2004). Concurrent multi-scale analysis of elastic composites by a multi-level computational model. *Computer Methods in Applied Mechanics and Engineering*, 193(6):497–538. 1, 54

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707. 7, 49

Raissi, M., Yazdani, A., and Karniadakis, G. E. (2020). Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030. 7, 49

Rajender, A. and Samanta, A. K. (2023). Compressive strength prediction of metakaolin based high-performance concrete with machine learning. *Materials Today: Proceedings*. 5

Rao, C. and Liu, Y. (2020). Three-dimensional convolutional neural network (3d-cnn) for heterogeneous material homogenization. *Computational Materials Science*, 184:109850. 3

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1278–1286, Bejing, China. PMLR. 5

Rocha, I., Kerfriden, P., and van der Meer, F. (2021). On-the-fly construction of surrogate constitutive models for concurrent multiscale mechanical analysis through probabilistic machine learning. *Journal of Computational Physics: X*, 9:100083. 103

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In Navab, N., Hornegger, J., Wells, W. M., and Frangi, A. F., editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham. Springer International Publishing. 44

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252. 10

Ryckelynck, D. (2009). Hyper-reduction of mechanical models involving internal variables. *International Journal for Numerical Methods in Engineering*, 77:75 – 89. 54

Saha, S., Gan, Z., Cheng, L., Gao, J., Kafka, O., Xie, X., Li, H., Tajdari, M., Kim, H., and Liu, W. (2020). Hierarchical deep learning neural network (hidenn): an artificial intelligence (ai) framework for computational science and engineering. *Computer Methods in Applied Mechanics and Engineering*, 373. 56

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. (2020). Learning to simulate complex physics with graph networks. In III, H. D.

and Singh, A., editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8459–8468. PMLR. 3, 47, 114, 191

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. (2019). How does batch normalization help optimization? arXiv preprint arXiv:1805.11604. 43

Schult, J., Engelmann, F., Kontogianni, T., and Leibe, B. (2020). Dualconvmesh-net: Joint geodesic and euclidean convolutions on 3d meshes. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8609–8619. 24, 109

Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489. 10

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815. 10

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Machine Learning Res.*, 15:1929–1958. 34

Stuckner, J., Harder, B., and Smith, T. M. (2022). Microstructure segmentation with deep learning encoders pre-trained on a large microscopy dataset. *npj Computational Materials*, 8(11):1–12. 6

Sun, Y., Hanhan, I., Sangid, M. D., and Lin, G. (2020). Predicting mechanical properties from microstructure images in fiber-reinforced polymers using convolutional neural networks. arXiv preprint arXiv:2010.03675. 2, 4, 5, 43

Sussillo, D. and Abbott, L. F. (2015). Random walk initialization for training very deep feedforward networks. arXiv:1412.6558. 17

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, Los Alamitos, CA, USA. IEEE Computer Society. 17

Tsymbalov, E., Panov, M., and Shapeev, A. (2018). Dropout-based active learning for regression. In van der Aalst, W. M. P., Batagelj, V., Glavaš, G., Ignatov, D. I., Khachay, M., Kuznetsov, S. O., Koltsova, O., Lomazova, I. A., Loukachevitch, N., Napoli, A., Panchenko, A., Pardalos, P. M., Pelillo, M., and Savchenko, A. V., editors, *Analysis of Images, Social Networks and Texts*, pages 247–258, Cham. Springer International Publishing. 6

Tunyasuvunakool, K., Adler, J., Wu, Z., Green, T., Zielinski, M., Žídek, A., Bridgland, A., Cowie, A., Meyer, C., Laydon, A., Velankar, S., Kleywegt, G. J., Bateman, A., Evans, R., Pritzel, A., Figurnov, M., Ronneberger, O., Bates, R., Kohl, S. A. A., Potapenko, A., Ballard, A. J., Romera-Paredes, B., Nikolov, S., Jain, R., Clancy, E., Reiman, D., Petersen, S., Senior, A. W., Kavukcuoglu, K., Birney, E., Kohli, P., Jumper, J., and Hassabis, D. (2021). Highly accurate protein structure prediction for the human proteome. *Nature*, 596(7873):590–596. 10

Vlassis, N. N., Ma, R., and Sun, W. (2020). Geometric deep learning for computational mechanics part i: anisotropic hyperelasticity. *Computer Methods in Applied Mechanics and Engineering*, 371:113299. 46, 106

Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., and Fergus, R. (2013). Regularization of neural networks using dropconnect. In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1058–1066, Atlanta, Georgia, USA. PMLR. 34

Wang, Y., Oyen, D., Guo, W. G., Mehta, A., Scott, C. B., Panda, N., Fernández-Godino, M. G., Srinivasan, G., and Yue, X. (2021). Stressnet - deep learning to predict stress with fracture propagation in brittle materials. *npj Materials Degradation*, 5. 2, 5, 45, 103

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. (2019). Dynamic graph cnn for learning on point clouds. *ACM Trans. Graph.*, 38(5). 25, 28

Xiao, M., Breitkopf, P., Coelho, R., Knopf-Lenoir, C., Sidorkiewicz, M., and Villon, P. (2009). Model reduction by cpod and kriging. *IntJStruc Multidisc Optim*, 41:555–574. 41, 54

Yan, J., Mu, L., Wang, L., Ranjan, R., and Zomaya, A. (2020). Temporal convolutional networks for the advance prediction of enso. *Scientific Reports*, 10:8055. 103

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. In Richard C. Wilson, E. R. H. and Smith, W. A. P., editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 87.1–87.12. BMVA Press. 17, 114

Ziaeipoor, H., Taylor, M., and Martelli, S. (2020). Population-Based Bone Strain During Physical Activity: A Novel Method Demonstrated for the Human Femur. *Annals of Biomedical Engineering*, 48(6):1694–1701. 2

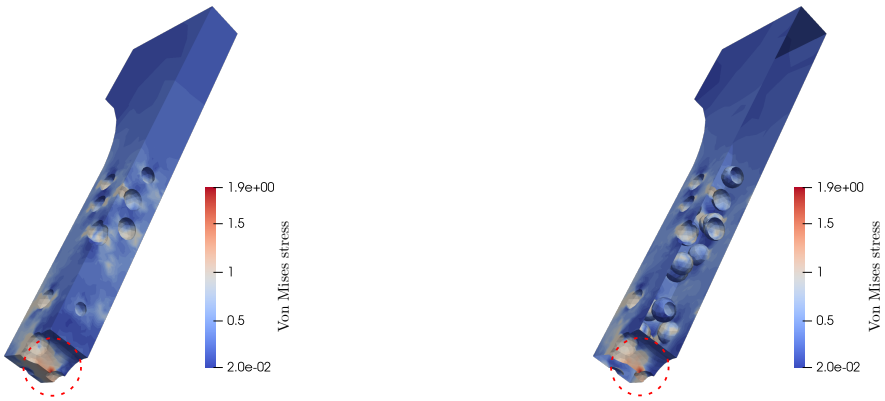
Zohdi, T. and Wriggers, P. (2005). *An Introduction to Computational Micromechanics*, volume 20. 1, 51

Évariste Sanchez-Palencia (1987). *General introduction to asymptotic methods*, volume 272. 1, 51

Appendix A

Volume and surface mesh

As discussed in section [6.2.1] of this thesis, the input and output of the model is the surface and not the volume stress. In [Fig. A.1] we see an example of a surface stress extracted from the volume mesh. We can observe that the maximum value of the Von Mises stress is the same in both cases and the location where the maximum Von Mises stress occurs is the same as well.



(a) Volume Stress (b) Surface Stress

Figure A.1: Two structures corresponding to one quarter of a random realisation of the dogbone geometry. In the diagram on the left (a) we see the Von Mises stress distribution on the volume mesh. In the diagram on the right (b) we see the Von Mises stress distribution on the surface mesh. We can see, in the red circles, that the maximum values in both structures are in the same location.

Appendix B

GNN parameters and architectural choices

We identified some key parameters for the training of the GNNs examined in [Chapter 6] and performed several tests to identify their optimum values. The parameters that we examined is the number of filters in the MLPs, the number of GN Blocks, the number of maximum neighbours per node, the existence of a skip connection between the input and the output of the network and finally the type of encoder. For these tests we used 600 patches from the dataset introduced in [section 6.4.1], 500 patches were used as the training set and 100 patches for the test set.

B.1 Skip connection

We suggest that using a skip connection to add the input macroscale stress tensor to the output of the network will improve GNN's performance. The reason behind this is that the GNN will learn how the microscale stress field deviates from the macroscale stress field instead of learning the microscale stress field from zero which we believe that it is easier and it should also improve the generalization ability of the network. In order to validate our assumption we train 2 identical GNNs with the exact same parameters and training set but one of them has a skip connection and the other does not. Both networks have MLPs with 128 filters, 10 GN blocks and each node has at most 10 neighbours. The results can be found in [Fig B.1] where we can observe that the GNN trained with the skip connection has smoother convergence compared to the GNN trained without the skip connection. Also, the skip connection improved accuracy

from 65% to 80%. We conclude that indeed the skip connection helped not only to have more stable training but also to improve the accuracy and thus we decide to include it in our architecture.

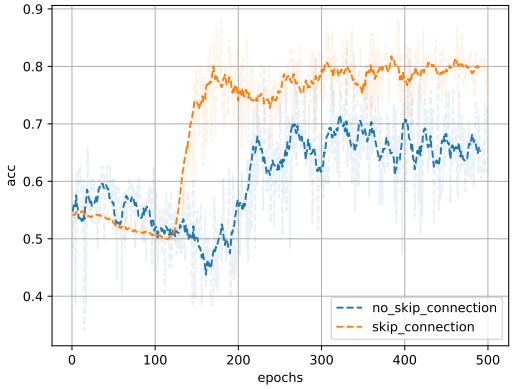


Figure B.1: In the diagram we see accuracy curves for the test dataset defined using the maximum Von Mises stress. The yellow line corresponds to a GNN trained with a skip connection to add the input macroscale stress to the output of the GNN and the blue line corresponds to a GNN trained without this skip connection. We observe that the skip connection results in smoother training and higher accuracy.

B.2 Number of filters

An important parameter that heavily influences not only the accuracy of the GNN but also the training time and memory requirements is the number of filters in the MLPs. In order to identify the minimum number of filters that result in optimum accuracy we trained 3 GNNs with 64, 128 and 256 filters in the MLPs and we kept all the other parameters the same. Specifically, all networks have the skip connection described in [B.1], 10 GN blocks and each node has at most 10 neighbours. The results can be found in [Fig B.2] where we observe that 128 and 256 filters result in the same accuracy which is higher than the accuracy for the 64 filters, 80% compared to 72%. Additionally, the training time for the GNN with the 256 filters is 12.7 hours and the maximum required memory is 15.4 GB while for the 128 filter GNN the training was completed in 5.8 hours and the maximum required memory was 7.7 GB. By choosing to use 128 filters in the MLPs of the GNN we achieve the same accuracy as in the 256 filters GNN but with a 54% decrease in the training time and a 50% decrease in memory requirements.

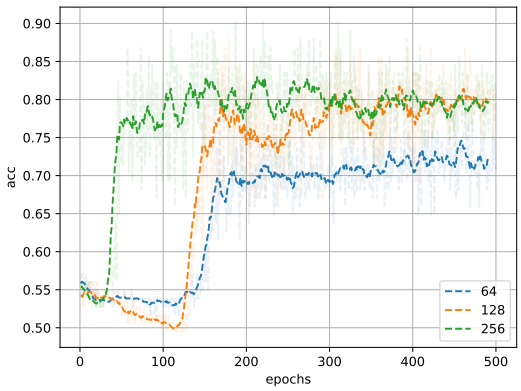


Figure B.2: In the diagram we see accuracy curves for the test dataset defined using the maximum Von Mises stress. Coloured lines correspond to networks trained with different number of filters, namely 64, 128 and 256. We observe that 128 and 256 filters result in the same accuracy 80% where 64 filters result in a lower accuracy of 72%.

B.3 Independent decoder

As already discussed in section [6.2.5], in the first layer of the GNN we choose to encode the edge and node features independently into the latent dimension as suggested by [Sanchez-Gonzalez et al., 2020; Pfaff et al., 2021; Mylonas et al., 2022]. We want to test if this choice results in an improved GNN and thus we perform an experiment between 2 GNNs with the same parameters where one uses a GN block to encode the inputs into the latent dimension and the other encodes them independently. Both of the GNNs have 10 GN blocks, 128 filters in the MLPs, the skip connection described in [B.1] and each node has at most 10 neighbours. The results can be found in [Fig B.3] where we observe that both GNNs have similar accuracy curves and they both have a final accuracy of 80%. Nevertheless, we can observe that in the independent encoder case the accuracy starts increasing sooner, at epoch 50, where in the GN block case it starts increasing later, at epoch 120. Lastly, the independent encoder version involves slightly less calculations and results in a 5% decrease in training time. Consequently, we choose to use the independent encoder GNN.

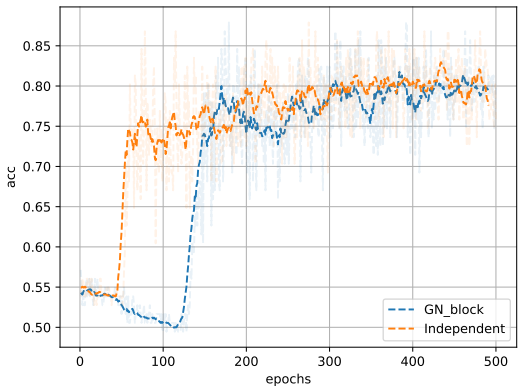


Figure B.3: In the diagram we see accuracy curves for the test dataset defined using the maximum Von Mises stress. The yellow line corresponds to a GNN trained with an independent encoder and the blue line corresponds to a GNN trained with a GN block as an encoder. We observe that both GNNs have similar accuracy curves with the same final accuracy but the GNN with the independent encoder starts increasing its accuracy sooner, epoch 50, compared to the other one, epoch 120.

B.4 Number of GN blocks

Another important parameter that affects both the memory requirement and the training time is the number of residual GN blocks. Because we are using residual connections, we do not expect a decrease in accuracy as we add more GN blocks but we are expecting that there should be a saturation point where the GNN does not benefit anymore from the GN blocks but it performs unnecessary calculations resulting in increased computational cost. To validate this assumption and determine the most proper number of GN blocks we train 3 GNNs with the same parameters but different number of GN blocks. All the GNNs have 128 filters in the MLPs, the skip connection described in [B.1], independent encoder and each node has at most 10 neighbours. The results can be found in [Fig B.4] where we observe that the GNN with only 3 GN blocks presents a decrease of 10% in the test accuracy compared to the other two. Also we can see that the GNNs with 5 and 10 GN blocks have similar test accuracy curves and they both reach a test accuracy of 80%. We decide to opt for the GNN with the 5 GN blocks that presents optimum accuracy without additional computational cost. This results in a training time of 3.1 hours and a maximum memory of 4.3 GB which is a decrease of 45% in training time and 44% in memory requirements compared to the GNN with the 10 GN blocks.

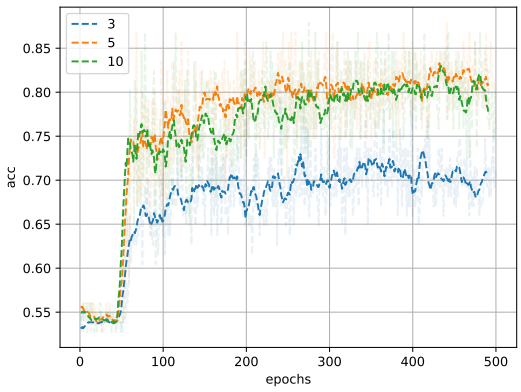


Figure B.4: In the diagram we see accuracy curves for the test dataset defined using the maximum Von Mises stress. Coloured lines correspond to networks trained with different number of GN blocks, namely 3, 5 and 10. We observe that the GNN with 3 GN blocks has a decreased accuracy compared to the rest. Additionally, we can see that the GNNs trained with 5 and 10 GN blocks have almost exactly the same behaviour.

B.5 Number of neighbours

We want to study the effect of the maximum number of neighbours on the GNN training. A small number of maximum neighbours will result in a graph where disconnected areas of the FE mesh do not share an edge and thus cannot exchange information, for instance microscale and macroscale features. We train 3 GNNs with the same parameters apart from the maximum number of neighbours that have values 5, 10 and 15. All the GNNs have 5 GN blocks, 128 filters in the MLPs, the skip connection described in [B.1] and independent encoder. The results can be found in [Fig B.5] where we observe that 10 and 15 neighbours result in the same accuracy which is higher than the accuracy for the 5 neighbours, 81% compared to 75%. Additionally, the training time for the GNN with the 15 neighbours is 27% higher and the memory requirements 31% higher compared to the GNN with 10 neighbours. We conclude that in our case we do not have a reason to use more than 10 neighbours although in denser meshes this number could be different.

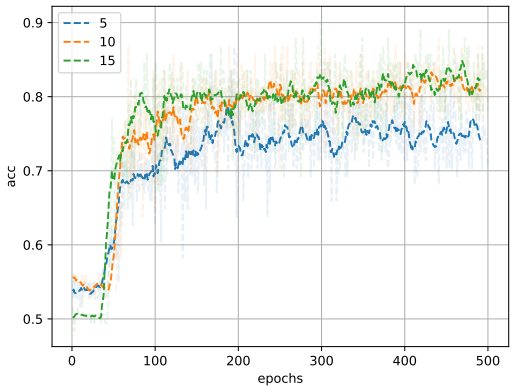
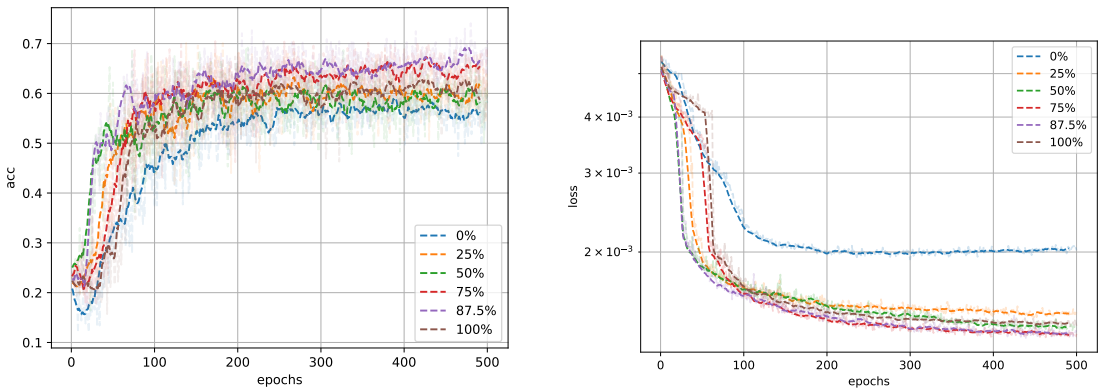


Figure B.5: In the diagram we see accuracy curves for the test dataset defined using the maximum Von Mises stress. Coloured lines correspond to networks trained with different number of maximum neighbours per node, namely 5, 10 and 15. We observe that 10 and 15 neighbours result in the same accuracy 81% where 5 neighbours result in a lower accuracy 75%.

B.6 Geodesic and Euclidean convolutions

We study the effect of the joint convolutions by training 6 GNNs with different number of Geodesic and Euclidean filters. Here we use data from the dogbone dataset introduced in [section 6.4.3]. We keep the total number of filters constant to 128 but we change the ratio between Geodesic and total (Euclidean + Geodesic) filters. We investigate the case where the ratio is 0% (only Euclidean), 25%, 50% and 75%, 87.5% and 100% (only Geodesic). All the GNNs have 5 GN blocks, residual connection, independent encoder, 128 filters and a maximum of 20 neighbours per node. The results can be found in [Fig B.6]. We can observe that both for the accuracy and the loss the worst case is when the ratio is 0, so no Geodesic convolutions are present. Both the accuracy and the loss improve (accuracy increases and loss decreases) as we increase the ratio until the value 75%. After that no further improvement in performance is observed, both the accuracy and loss curves for ratio values 75% and 87.5% are the same. We conclude that a 75% ratio is the most beneficial for this case.



(a) Accuracy (b) Loss

Figure B.6: In the diagram on the left (a) we see accuracy curves defined using the maximum Von Mises stress and a 15% threshold for the relative error. Specifically, we see the accuracy as function of the training epochs. In the diagram on the right (b) we see the loss function as a function of the epochs. For both diagrams, coloured lines correspond to GNNs trained with different Geodesic to total (Euclidean + Geodesic) filter ratio, namely 0% (only Euclidean), 25%, 50%, 75%, 87.5% and 100% (only Geodesic).

B.7 Full stress field VS maximum stress

We investigate the option to directly predict the maximum Von Mises stress on the surface of the dogbone structure. To this end, we train 2 identical GNNs on patches extracted from 100 FE simulations and we evaluate them in a different set of patches extracted from 100 FE simulations. One of the GNNs directly predicts the maximum Von Mises stress and the other the full stress tensor. Here we use data from the dogbone dataset introduced in [section 6.4.3]. Both GNNs have 5 GN blocks, independent encoder, 128 filters (32 Euclidean and 96 Geodesic) and a maximum of 20 neighbours per node. The results can be found in [Fig B.7]. We observe that the 2 GNNs have similar accuracy, but the GNN predicting the full stress has slightly higher values. We conclude that predicting the full stress is beneficial since it not only results in better prediction of the maximum Von Mises stress, but it also provides us with the full stress tensor that can be used for the online bias corrections [section 6.4.2].

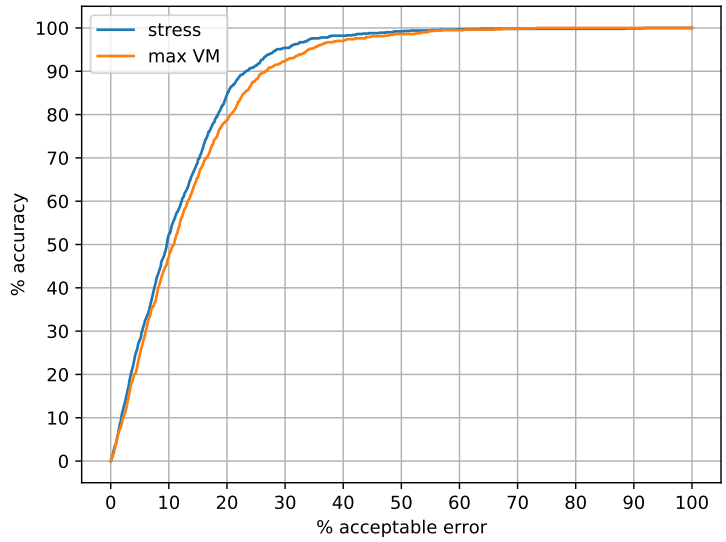
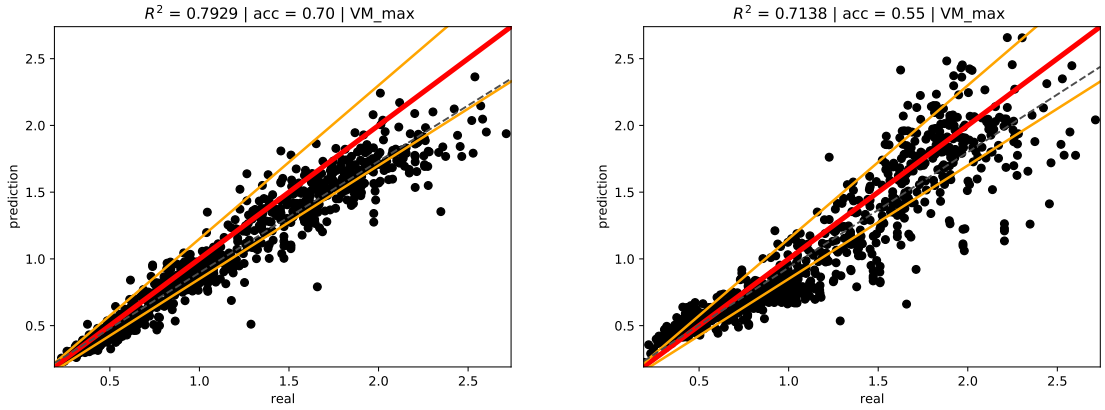


Figure B.7: Accuracy curves as a function of the threshold value used for the relative error. The accuracy curves are defined using the maximum Von Mises stress. The blue line corresponds to a GNN that predicts the full microscale stress distribution first and then from this the maximum Von Mises stress in the ROI of the patch. The orange line corresponds to a GNN that directly predicts the maximum Von Mises in the ROI of the patch.

B.8 Patches VS full structure

Lastly, we want to evaluate to what degree our choice to train the GNN using patches of the geometry affects the quality of the results. This choice stems from the fact that defects only locally affect the macroscale stress field and distant areas do not offer significant information. On one hand, training on patches has the benefit of making the GNN unaware of the specific structure and it encourages it to focus on predicting how the microscale features affect the global stress field, thus leading to improved generalisation. On the other hand, our choice for the size of ROI and patch has to be careful so that the network has all the necessary information to make predictions in the ROI. Choosing to train the GNN using the entire structure alleviates us from this problem. We perform an experiment to compare the two strategies. Here we use data from the dogbone dataset introduced in [section 6.4.3]. We train two GNNs with the same parameters, namely 5 GN blocks, residual connection, independent encoder, 128 filters (32 Euclidean and 96 Geodesic) and a maximum of 20 neighbours per node. The first GNN is trained using patches from 400 FE simulations (10 patches from each simulation) and the second GNN is trained directly on the 400 FE simulation results. We evaluate both GNNs on a separate set of patches extracted from 100 FE simulations. The results can be found in [Fig B.8]. We can observe that for the GNN that was trained on patches, the maximum Von Mises in the ROI of the patches is less scattered around the $y = x$ line, true value. The coefficient of determination for the patch case is 0.79 compared to 0.71 for the full case and the accuracy significantly drops from 70% for the patch case to 55% for full dogbone case. Thus we conclude that our choice to train the GNN with patches indeed leads to improved generalisation. We also compare the Von Mises stress distribution on the surface of a dogbone structure as predicted by the GNN trained with patches, with the one predicted by the GNN trained with the entire structure. In [Fig B.9] we see that the GNN that was trained using patches more accurately predicts the high stress area compared to the GNN that was trained with the full structure.



(a) patches (b) full

Figure B.8: In both diagrams the x-axis corresponds to the maximum Von Mises stress in the ROI of the patches as calculated by FE simulations and the y-axis to the maximum Von Mises stress in the ROI of the patches as predicted by the GNN. The yellow lines correspond to the 15% relative error threshold. The accuracy is defined as the percentage of points with relative error less than 15%. The image on the left (a) corresponds to a GNN trained with patches extracted by 400 FE simulations and the image on the right (b) to a GNN trained directly on the same 400 FE simulations.

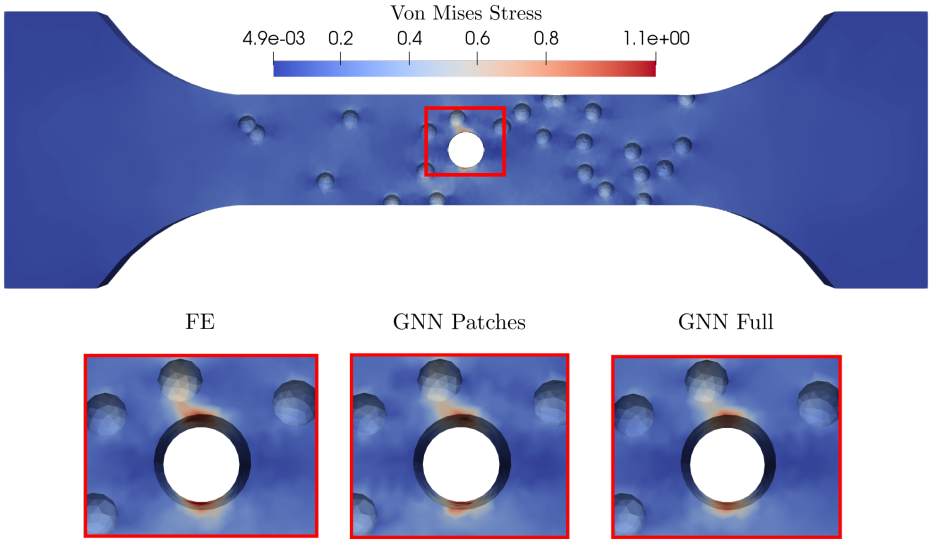


Figure B.9: Von Mises stress on the surface of a dogbone structure (top). On the bottom of the image we have zoomed in the high stress area. From left to right we see the FE result, the prediction of the GNN trained with patches and the GNN prediction of the GNN trained with full structures.

Appendix C

Ensemble Kalman method: Observation matrix free implementation

We can simplify the calculations involved in the Kalman update procedure (section 6.3) by avoiding to explicitly define the observation matrix. Instead we can define a function that will provide the noise free value of the data. This function is called observation function and is of the from

$$h(\mathbf{x}) = \mathbf{H}\mathbf{x} \tag{C.1}$$

The posterior can be written as

$$\mathbf{X}^\star = \mathbf{X} + \frac{1}{N-1}\mathbf{A}(\mathbf{H}\mathbf{A})^T\mathbf{P}^{-1}(\mathbf{D} - \mathbf{H}\mathbf{X}) \tag{C.2a}$$

$$\mathbf{H}\mathbf{X} = h(\mathbf{X}) \tag{C.2b}$$

$$\mathbf{A} = \mathbf{X} - \frac{1}{N}\mathbf{X} \tag{C.2c}$$

$$\mathbf{H}\mathbf{A} = \mathbf{H}\mathbf{X} - \frac{1}{N}\mathbf{H}\mathbf{X} \tag{C.2d}$$

$$\mathbf{P} = \frac{1}{N-1}\mathbf{H}\mathbf{A}(\mathbf{H}\mathbf{A})^T + \mathbf{\Sigma}_\epsilon \tag{C.2e}$$

40110 words 196563 characters (not including spaces)

`File: main.tex`

```
Encoding: utf8
Sum count: 40110
Words in text: 32729
Words in headers: 453
Words outside text (captions, etc.): 6340
Number of headers: 140
Number of floats/tables/figures: 112
Number of math inlines: 527
Number of math displayed: 61
Subcounts:
  text+headers+captions (#headers/#floats/#inlines/#displayed)
  0+1+0 (1/0/0/0) Chapter: Introduction
  1002+3+0 (1/0/0/0) Section: Motivation and strategies
  49+6+0 (1/0/0/0) Section: Aims and contributions of this thesis
  326+3+0 (1/0/0/0) Subsection: Multiscale Neural Networks
  215+3+0 (1/0/0/0) Subsection: Bayesian Neural Networks
  604+5+0 (1/0/0/0) Subsection: Data requirements and generalisation ability
  367+2+0 (1/0/0/0) Section: Thesis structure
  18+1+0 (1/0/0/0) Section: Publications
  66+2+0 (1/0/0/0) Subsection: International journals
  179+5+0 (1/0/3/0) Subsection: Conference papers, presentations and workshops
  52+2+0 (1/0/0/0) Chapter: Machine Learning} \label{Machine Learning Chapter
  188+1+0 (1/0/0/0) Section: Introduction
  302+2+16 (1/1/12/3) Section: Linear regression} \label{linear regression
  64+4+0 (1/0/0/0) Section: Fully connected Neural Networks
  542+4+67 (1/2/28/3) Subsection: Single hidden layer NNs
  88+2+0 (1/0/11/1) Subsection: Multi-layer NNs
  56+3+0 (1/0/0/0) Section: Convolutional Neural Networks
  242+1+0 (1/0/0/0) Subsection: Introduction
  552+2+168 (1/2/18/1) Subsection: Convolution operation} \label{capybara section
  178+1+7 (1/1/4/0) Subsection: Pooling} \label{pooling_section
  118+1+35 (1/3/0/0) Subsection: Upsampling
  23+2+0 (1/0/4/1) Subsection: Simple CNN
  89+4+24 (1/2/0/0) Subsection: Example of a CNN
  294+5+23 (1/0/6/0) Subsection: Comparison with fully connected layers
```

53+3+0 (1/0/0/0) Section: Graph Neural Networks} \label{GNN section
63+1+31 (1/1/0/0) Subsection: Introduction
492+2+0 (1/0/0/0) Subsection: Literature review} \label{literature_review_GNN
807+2+47 (1/2/34/2) Subsection: Graph convolutions
68+3+0 (1/0/0/0) Section: Bayesian Neural Networks} \label{BNN intro
419+1+95 (1/2/6/1) Subsection: Introduction} \label{BNN intro intro
140+2+40 (1/1/5/2) Subsection: Bayesian modelling
323+2+0 (1/0/5/1) Subsection: Variational Inference
155+3+0 (1/0/8/0) Subsection: Bayesian linear layer} \label{Bayesian Layer
114+5+28 (1/1/0/0) Subsection: Example of a Bayesian NN
61+5+0 (1/0/0/0) Subsection: Extension to CNNs and GNNs
896+1+0 (1/0/25/7) Section: Optimiser} \label{optimisers
45+4+0 (1/0/0/0) Section: Conclusions of the chapter
0+4+0 (1/0/0/0) Chapter: Finite Element surrogate models} \label{PDEs and ML Chapter
94+1+0 (1/0/0/0) Section: Introduction
129+3+0 (1/0/4/1) Section: Classical surrogate models
61+4+0 (1/0/0/0) Section: PDEs and Machine Learning
223+5+0 (1/0/0/0) Subsection: NNs with fully connected layers
709+1+34 (1/4/8/0) Subsection: CNNs} \label{CNN literature
1079+1+123 (1/1/18/1) Subsection: GNNs} \label{GNN literature
348+1+45 (1/1/2/0) Subsection: PINNs
140+2+0 (1/0/0/0) Section: Multiscale methods
248+1+70 (1/2/11/8) Subsection: Homogenisation
149+3+31 (1/1/0/0) Subsection: Concurrent multiscale modelling
293+4+0 (1/0/1/0) Section: ML assisted multiscale methods
96+4+0 (1/0/0/0) Section: Conclusions of the chapter
0+9+0 (1/0/0/0) Chapter: Multiscale problem formulation, structures of interest and
60+1+0 (1/0/0/0) Section: Introduction
175+1+0 (1/0/21/10) Section: Elasticity
42+2+0 (1/0/4/0) Section: Equivalent stress
60+2+73 (1/1/0/0) Section: Porous medium
205+2+26 (1/1/9/0) Section: Multiscale problem} \label{Multiscale Problem
180+1+2 (1/0/21/0) Section: Accuracy
85+4+0 (1/0/0/0) Section: Conclusions of the chapter
0+13+0 (1/0/0/0) Chapter: Convolutional Neural Networks for the prediction of equiva

352+1+0 (1/0/0/0) Section: Introduction
32+3+0 (1/0/0/0) Section: Convolutional Neural Network
271+1+72 (1/2/15/0) Subsection: Input-Output
51+2+0 (1/0/0/0) Subsection: Loss function
272+1+42 (1/3/18/0) Subsection: Architecture
114+2+0 (1/0/0/0) Section: Numerical examples
2821+22+964 (6/16/59/6) Subsection: Linear elasticity} \label{linear elastic
620+10+283 (3/7/10/0) Subsection: Nonlinear elasticity} \label{Non Linear Section
982+2+239 (1/4/0/0) Subsection: Selective learning} \label{Selective Learning
377+4+213 (1/3/6/0) Subsection: Out of distribution study
309+3+326 (1/3/0/0) Section: Comparison to homogenisation
402+3+0 (1/0/0/0) Section: Assumptions and limitations
284+6+0 (1/0/0/0) Section: Conclusion and perspectives of this chapter
0+12+0 (1/0/0/0) Chapter: Graph Neural Networks for the prediction of stress in 3D p
274+1+0 (1/0/0/0) Section: Introduction
51+6+0 (1/0/0/0) Section: Geometric learning for multiscale stress analysis
352+3+64 (1/1/3/0) Subsection: Assumptions and justifications} \label{Assumptions an
460+2+96 (1/1/18/0) Subsection: Graph construction} \label{neighbourhood
233+1+199 (1/2/12/0) Subsection: Input-Output
55+2+0 (1/0/0/0) Subsection: Loss function
601+5+44 (3/2/21/0) Subsection: GNN model
720+15+0 (1/0/22/11) Section: Physics-based corrections of the NN predictions: enfor
0+2+0 (1/0/0/0) Section: Numerical examples
934+24+306 (6/6/17/1) Subsection: Numerical example with cubical heterogeneous mater
691+11+443 (3/6/5/0) Subsection: Online stress correction} \label{Online_stress_corr
1110+21+392 (5/8/10/0) Subsection: Numerical example with dogbone data} \label{singl
506+12+351 (3/5/0/0) Subsection: Variable dimension dogbone
760+6+0 (1/0/0/0) Section: Conclusion and perspectives of this chapter
0+14+0 (1/0/0/0) Chapter: Centroid Graph Neural Network for the prediction of equiva
422+1+0 (1/0/0/0) Section: Introduction
327+1+224 (1/3/0/0) Section: Input-Output
94+2+0 (1/0/0/0) Section: Loss function
65+1+0 (1/0/0/0) Section: Architecture
179+2+227 (1/1/28/0) Section: Bayesian Optimisation} \label{BO strategy
178+2+0 (1/0/0/0) Section: Numerical examples

```
  347+10+44 (3/2/2/0) Subsection: Numerical examples with deterministic GNN} \label{De
  130+11+129 (3/1/0/0) Subsection: Numerical examples with probabilistic GNN} \label{B
  458+7+33 (3/2/0/0) Subsection: Transfer Learning
  804+10+664 (5/5/13/1) Subsection: Bayesian Optimisation
  297+6+0 (1/0/0/0) Section: Conclusion and perspectives of this chapter
  0+1+0 (1/0/0/0) Chapter: Conclusions
  150+2+0 (1/0/0/0) Section: Problem statement
  418+9+0 (1/0/0/0) Section: Proposed approach and contribution to state of the art
  0+1+0 (1/0/0/0) Section: Results
  796+9+0 (1/0/0/0) Subsection: Neural Networks for fast stress predictions in multisc
  199+5+0 (1/0/0/0) Subsection: Reliable Neural Network stress predictions
  191+5+0 (1/0/0/0) Subsection: Data requirements and generalisation ability
  150+1+0 (1/0/0/0) Section: Limitations
  264+2+0 (1/0/0/0) Section: Future work

File: output.bbl
Encoding: utf8
Sum count: 0
Words in text: 0
Words in headers: 0
Words outside text (captions, etc.): 0
Number of headers: 0
Number of floats/tables/figures: 0
Number of math inlines: 0
Number of math displayed: 0
```