ToSiM-IoT: Towards a Sustainable Optimisation of Machine Learning Tasks in Internet of Things

Ashish Kaushal, Osama Almurshed, Asmail Muftah, Nitin Auluck, and Omer Rana

***Abstract*—With the rise of digital infrastructure and Internet of Things (IoT), a substantial amount of data is continuously generated that needs to be processed efficiently. While modern artificial intelligence (AI) approaches have shown good capabilities in handling large volumes of data, their excessive demands for memory and processing power result in very high utilisation of resources. In this work, we propose ToSiM-IoT, an optimisation framework that introduces a layer selection approach to identify an ideal mix of active and inactive layers, using a genetic algorithm for model training. We also propose a pruning mechanism that identifies performance-critical features using heatmap visualisation, during model inference, and eliminate the remaining features. Two machine learning (ML) models – InceptionV3 and VGG16 have been evaluated on an agricultural weed detection scenario, using the DeepWeeds image classification dataset. Experimental results demonstrate that our framework can achieve a significant reduction in model size and training time, while maintaining high accuracy, for both models. This demonstrates that our approach can be efficiently deployed on intelligent IoT systems where computational capabilities are limited.**

***Index Terms*—channel pruning, genetic algorithm, heatmap visualisation, internet of things, layer selection, machine learning, optimisation.**

I. Introduction

In recent years, the advancements in Internet of Things (IoT) has offered significant opportunities to achieve Sustainable Development Goals (SDGs) by providing enhanced connectivity and real-time data analysis across various sectors. By integrating sensors, actuators, and end-user devices, IoT networks facilitate the collection of vast amounts of data and enable more informed decision-making with optimised utilisation of resources [1]. Integrating Machine Learning (ML) and Artificial Intelligence (AI) models in IoT can provide a direction for significant advancement in the domain of computational technology. These models have also seen exponential growth across numerous real-world tasks such as image classification [2], object detection [3], and video analysis [4] in the past few years. In order to achieve higher accuracy and performance, researchers have focused on designing architectures that are both deeper and broader, like VGG, Inception, ResNet, YOLO etc.

Deployment of complex, deep models on resource-constrained nodes, such as mobile robots, field side units, unmanned aerial vehicles (UAVs), and end-user IoT devices, presents significant challenges [5]. Performing convolution operations on these models demands substantial computational power and energy. Additionally, the extensive number of network parameters results in high storage requirements, causing further challenges in resource-limited environments [6]. If we consider InceptionV3 and VGG16 models as an example; they have more than 138 million and 23 million parameters respectively. Moreover, to process a single 224x224 image, these models require around 6 billion and 30 billion floating-point operations (FLOPs). In order to implement such large-scale deep learning models on resource-constrained infrastructures, it is important to address the issue of their expensive computation cost and high memory demands.

Numerous methods have been developed to handle the rising demand of memory and resources in deep learning models. Model enhancement techniques like pruning, quantization, regularisation, and knowledge distillation are crucial for reducing the size and improving the speed of these models. Pruning [7] involves removing unnecessary weights from a neural network, effectively decreasing its complexity and size. Quantization [7] converts parameter weight values from floating type to integer type, which decreases memory usage and can speed up inference. Knowledge distillation [8] transfers the knowledge from a large, complex model to a smaller, faster one without significant loss in accuracy. Additionally, designing compact neural architectures and optimising them for specific hardware are a few other strategies used to further accelerate the model performance.

Optimising ML pipelines and system life cycles often presents significant challenges and is a very complex task. Training larger neural networks with sparse activations can enhance model scalability and performance. However, this approach may lead to increased carbon emissions and energy utilisation due to higher demand of system resources [9]. Offloading model training and inference task to data centres powered by carbon-neutral energy sources offers a potential reduction in emissions but might not be practical for all application use cases. This is because the development of carbon-neutral infrastructure is often constrained by geographical and material availability limitations [10]. Moreover, with the rising trend of on-device learning to enhance data privacy, more computational tasks are being offloaded from centralised servers to the low-level edge and IoT devices [11], [12]. Therefore, there is a critical need for a sustainable training and inference infrastructure that reduces resource utilisation – both in terms of memory and computation – without compromising performance of the models.

The primary contributions of this paper are as follows –

- We introduce a new method to significantly reduce

Ashish Kaushal and Nitin Auluck are with the Department of Computer Science and Engineering, Indian Institute of Technology Ropar, India (email: {ashish.19csz0003, nitin}@iitrpr.ac.in). Asmail Muftah is with Azzaytuna University, Libya (email: asmail.muftah@azu.edu.ly). Osama Almurshed and Omer Rana are with the School of Computer Science and Informatics, Cardiff University, United Kingdom (email: {almurshedo, ranaof}@cardiff.ac.uk).

the computational demands associated with the back-propagation process in neural network training. This technique minimises the computational load by reducing the number of parameters that need to be updated, and also cuts down the time required for both the current training session and any future adjustments to the model.
- We propose an approach to refine the architecture of ML models by carefully adjusting their parameter count. The method uses an automated process to systematically identify and prioritise the most significant parameters based on the specific dataset used for training. This not only reduces the complexity of the model, but also significantly enhances the efficiency of ML operations during both the training and inference phases.
- We design a mathematical formulation that performs theoretical analysis to identify the objective functions that critically influence the execution performance of ML operations. The model quantifies the relation between various parameters and their impact on computational efficiency, thereby allowing for the precise tuning of the model to optimise performance.
- We evaluate the proposed framework on an agricultural weed detection use case scenario. It utilises the DeepWeeds Image classification dataset with two ML models – VGG16 and InceptionV3.

Our proposed ToSiM-IoT framework utilises GA and Grad-CAM based optimal solutions, primarily due to their adaptive and efficient execution nature. The GA-based layer selection method significantly reduces the computational overhead by selectively activating a subset of layers, rather than engaging the full network. This approach ensures that the training process is lighter and faster, which is critical in environments where quick real-time execution is needed. Moreover, by eliminating less critical features, the pruned models are not only faster, but also consume less energy and space, making them ideal for deployment on resource-constrained IoT systems. A brief overview of the ToSiM-IoT framework is shown in Fig. 1. Both our proposed techniques are highly adaptable algorithms that can utilise the neural network architecture and its parameters to varying data conditions without manual intervention. This adaptability makes them suitable for autonomous and dynamic environments where input conditions change frequently. Despite the reduction in computational resources, our methods maintain high accuracy levels, as they retain the necessary elements that significantly contribute towards high performance and efficiency.

II. Understanding the Problem

To understand and analyse the sustainable deployment of AI, it is crucial to explore the challenges described by researchers in this domain. The following subsections highlight the recent work done in the field and discuss their impact and potential on sustainability.

A. Challenges in AI, Sustainability, and IoT

Incorporating AI into the IoT provides high efficiency and intelligence to systems, enabling devices to process data autonomously. This integration equips devices with the ability to make decisions, optimise operations, and provide insights without direct or indirect human intervention. It includes providing everyday devices with smart functionalities for revolutionising processes and highlighting the transformative potential of AI within IoT frameworks. However, utilisation of this potential requires overcoming significant challenges, particularly in adapting AI technologies to the diverse environments where IoT devices operate. Mhaisen et al. [13] provides a survey of recent techniques and strategies designed for handling AI tasks in IoT applications. Another work [14] focuses on security techniques based on ML describing how AI can be used for enhancing security in IoT systems. ML-based techniques for authentication, malware detection, offloading, and access control are mainly focused in this work. Bu et al. [15] presents an agriculture system for IoT that utilises AI and cloud computing for making smart decisions such as determining the amount of water needed for irrigation in the fields.

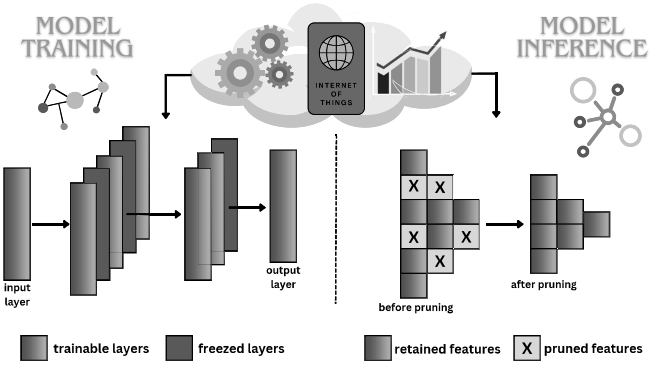

Fig. 1: A brief overview of ToSiM-IoT framework.

As IoT networks expand, so do the computational, memory, and energy demands of AI models used with them. Larger AI models require very high computational power and memory, leading to increased energy consumption during both training and inference phases. The work by Canziani et al. [16] and Li et al. [17] analyses the trade-offs between model size, performance, and energy efficiency; illustrating how larger models, while potentially more accurate, can significantly have high energy consumption and prominent impact on the environment. The size of an AI model also affects its deployability in real-world applications, especially in resource-constrained environments. Large models may not be feasible for deployment on mobile devices or in edge computing scenarios, where energy efficiency is a critical performance factor. This limitation challenges the scalability of AI solutions and their ability to be deployed sustainably across a diverse range of IoT platforms [16]. Many efforts to create model architectures such as EfficientNet by Tan and Le [18], [19], demonstrate the potential to reduce the impact of large AI models. These architectures aim to maintain or improve performance while reducing the computational demands and energy consumption, addressing the sustainability concerns associated with model size. Hu et al. [20] explore a channel pruning method which can be used for compressing large AI

models. Another work [21] evaluates the efficiency of model compression within the context of energy-efficient inference. Chen et al. [22] utilise the low-rank approximation to eliminate the redundancy within the filter and accelerate the deep neural network pruning process. The researchers [23] proposed a mechanism which identifies the sensitive input values that are highly correlated to the accuracy and applies the low-fidelity quantization on non-critical regions to boost the execution performance. Wu et al. [24] propose a method based on dynamic gradient compression and knowledge distillation to execute the federated learning tasks efficiently.

Along with optimisation, another crucial challenge is to extend sustainable AI development beyond optimising model efficiency and training time to encompass environmental and societal impacts. This requires embedding sustainability in AI's lifecycle, adopting new optimised techniques and cultivating a culture that prioritises sustainability, including transparent reporting of environmental impacts and adopting energy-efficient practices. Addressing the sustainability issues related to AI model training and size is vital for its performance advancement, ensuring its societal benefits while minimising environmental damage. The challenge, as illuminated in [25], revolves around designing machine learning models that emphasise model architecture and strategic size reduction highlighting the importance of ensuring efficiency during the early design stages of AI development, aiming for high-performance AI technologies that provide optimal operation within limited resource conditions.

B. Existing Solutions and their Potential

In deep learning, optimising models is important, particularly when considering the computational cost of training, which is heavily influenced by floating-point operations. Strategies such as quantization, pruning, and layer freezing can serve as important approaches to mitigate these costs and enhance overall performance.

A strategic approach to reducing FLOPs involves converting data from 32-bit floating-point precision (float-32) to 8-bit integer precision (integer-8), followed by performing critical operations in integer-8 and subsequently restoring the output to float-32. This method effectively optimises computational efficiency and memory usage while maintaining the accuracy crucial for deep learning tasks. The effectiveness of this quantization approach has been thoroughly explored with machine learning frameworks such as TensorFlow and PyTorch – offering specialised tools to facilitate this process. Therefore, emphasising its importance in improving model performance through FLOP management process. Based on the concept of computational efficiency, pruning is another strategy for refining model's structure. By targeting and removing less impactful parameters rather than adjusting model precision, pruning enhances efficiency by focusing on the redundancy of features. This technique maintains the model's integrity by emphasising structural optimisation instead of numerical adjustments of weights and biases. The pruning process reduces the computational complexity, operational costs, and ensures a balance between preserving model performance and achieving efficiency gains. Layer freezing, in addition to quantization and pruning, allows a strategic selection to either train or freeze neural network layers, impacting the learning process without altering any internal parameters. This technique, often achieved through hyperparameter tuning, reduces the number of calculations and FLOPs, thereby reducing computational load and energy consumption of the models. Layer freezing improves training efficiency by allowing the training of specific layers and freezing rest of the layers, thereby presenting another approach for optimising AI models.

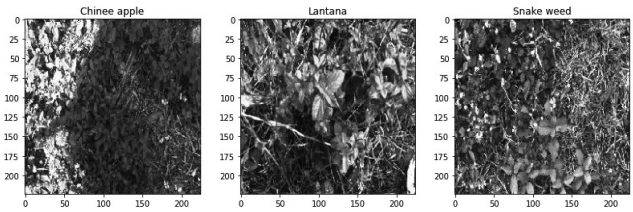

Fig. 2: Three sample weed images from DeepWeeds dataset.

C. Application Use-Case and its Generalisability

We have considered a weed identification agricultural scenario as a use case in this work. Effective weed management in precision agriculture is a critical task that can help farmers in maximising crop yield, reducing cost, and minimising the use of herbicides in agricultural fields. Two ML models – InceptionV3 and VGG16 have been used for training and inference procedures with the DeepWeeds dataset in our framework. A few sample weed images from DeepWeeds dataset are shown below in Fig. 2. The VGG16 architecture developed by researchers from the Visual Graphics Group (VGG) at Oxford is primarily known for its simplicity and depth. It features a series of convolutional layers, mainly 16 (13 convolutional and 3 fully connected layers), that use 3x3 filters with a stride of 1 pixel. This small filter size allows it to capture the fine details from the image through deeper architectures, without a rapid increase in computational complexity. The layers are followed by ReLU activation units and max pooling layers to reduce spatial dimensions, while retaining the most critical feature information. However, the InceptionV3 model – a part of the GoogLeNet family, introduces a more complex architecture designed to provide high accuracy with computational efficiency. The main aspect of InceptionV3 is its inception modules, which allow the model to choose from filters of various sizes (1x1, 3x3, 5x5) at each layer of the network. This multi-scale processing enables the model to capture information at various spatial hierarchies. Inception layers are also designed with dimensionality reduction techniques that involve 1x1 convolutions to reduce the computational burden before expensive 3x3 and 5x5 convolutions. This configuration makes InceptionV3 less prone to overfitting, while enhancing its ability to generalize across different datasets and tasks. The model also incorporates advanced features such as label smoothing, factorized 7x7 convolutions, and batch normalisation, providing high efficiency and speed during training.

Considering the applicability of our ToSiM-IoT framework, while demonstrated in an agricultural scenario, it is fundamentally designed to be adaptable to a range of applications beyond this specific use case. The use of GAs extends across any ML scenario where model complexity and computational efficiency are of concern. For instance, in healthcare, GAs can be utilised to design deep learning models that consider vast amounts of medical data, while optimising for accuracy and speed – both highly critical in diagnostic applications. Similarly, in autonomous vehicles, GAs can be crucial in refining CNNs that process and interpret real-time sensory data, ensuring that models are both quick and precise in their decision-making under variable environmental conditions. Grad-CAM's capability to visually highlight which features in the data most significantly impact the model's predictions offers extensive benefits for model refinement and optimisation in diverse use cases. In the financial sector, this could be applied to identify the most influential economic indicators affecting predictive models for stock prices or market trends. In customer service and sentiment analysis, Grad-CAM could pinpoint specific aspects of customer interactions that most affect sentiment scores, guiding better customer engagement strategies. By integrating these methods into different domains, users can enhance model performance, and also achieve higher operational efficiency, supporting compliance with growing regulatory demands for transparency in AI applications, such as in Europe under GDPR, or in various industries under sector-specific regulations.

III. Mathematical Formulation

In this section, we introduce the quantitative objective functions for our ML optimisation framework. The formulation is designed to identify the interrelations between various model parameters, and their impact on computational efficiency. We have formulated our selection approach as an optimisation problem to maximise the accuracy of a neural network model by strategically selecting layers for training. This selection process is controlled by binary decision variables for each layer x_i.

Let $x = (x_1, x_2, \ldots, x_N)$, where each layer x_i is defined as:

$$x_i = \begin{cases} 1 & \text{if layer } i \text{ is active} \\ 0 & \text{if layer } i \text{ is inactive} \end{cases} \tag{1}$$

for $i = 1, 2, \ldots, N$. The formulation consists of the following key parameters:

- N: Number of layers.
- C: Computational cost limit.
- c_i: Computational cost of layer i.
- v_i: Performance value of activating layer i.
- α, β: Weighting coefficients.

To identify an ideal combination of trainable layers, we use an adaptive heuristic search algorithm. The steps of the search are as follows.

First, we initialise by generating a population of M chromosomes:

$$P = \{x^{(1)}, x^{(2)}, \ldots, x^{(M)}\} \tag{2}$$

Then, we define a fitness function that can evaluate the desirability of each selected chromosome:

$$\text{Fitness}(x) = Z(x) \tag{3}$$

Based on the fitness, the next step is to select K chromosomes for mating, and enhancing them:

$$S = \text{select}(P, K, \text{Fitness}) \tag{4}$$

After the selection has been made, we identify some pairs $(x^{(a)}, x^{(b)})$ of chromosomes, and perform crossover to produce new offspring:

$$x^{(new)} = \text{crossover}(x^{(a)}, x^{(b)}) \tag{5}$$

Along with the crossover, we also perform mutation of the offspring's genes with a probability p_m:

$$x^{(mutated)} = \text{mutate}(x^{(new)}, p_m) \tag{6}$$

After performing the gene alteration, we integrate new offspring back into the population, by replacing them with weaker chromosomes:

$$P = \text{update}(P, x^{(mutated)}) \tag{7}$$

We repeat the selection, crossover, mutation, and replacement steps for G generations, or until a stopping criterion specified by the user is met.

The objective of our layer selection mechanism is also formulated as an optimisation function, represented as maximisation of $Z(x)$, such that:

$$Z(x) = \alpha \sum_{i=1}^{N} v_i x_i - \beta \left(\sum_{i=1}^{N} c_i x_i - C \right)^{+} \tag{8}$$

Here $(z)^{+} = \max(z, 0)$, which denotes the penalty for exceeding the computational limit of selected resource node. To achieve this, the following constraint should be satisfied:

$$\sum_{i=1}^{N} c_i x_i \leq C \qquad \text{and} \qquad x_i \in \{0, 1\} \quad \forall i$$

The aim of this formulation is to find an optimal set of layers (x_i) that maximises the neural network's accuracy within the given computational cost C. This constraint can be used to establish a balance between execution performance, resource utilisation; and highlights the complexity of training neural network models along with the importance of each layer selection decision.

For our feature pruning mechanism, we have designed another multi-objective optimisation function. The aim of the pruning process is to balance the predictive performance of the model with its computational efficiency. The pruning function is as follows:

$$P(x) = \omega \cdot \text{Acc}(x) - \lambda \cdot \text{Comp}(x) \tag{9}$$

Here:

- $x = (x_1, x_2, \ldots, x_c)$ is the binary vector indicating the presence (1) or absence (0) of each channel i in the CNN's target layer.

- Acc(x) is the accuracy of the CNN, given the set of active channels defined by x.
- Comp(x) quantifies the total computational cost associated with the active channels.
- ω and λ are weighting factors that prioritise accuracy versus computational cost.

We integrate Grad-CAM in our approach to identify the importance of each channel in the decision-making process. To achieve this, we define a function $I(i)$ that measures the significance of each channel i, based on its contribution to the model's output. This is determined by generating the heatmap and visualising its impact score. The importance function $I(c)$ is given as:

$$I(i) = \int H(x, y, i)\, dx\, dy \tag{10}$$

Here, $H(x, y, i)$ is the heatmap value at position (x, y) for channel i, and the integral sums the contributions across the spatial dimensions of the heatmap. The binary decision variables x_i for each channel i are determined as:

$$x_i = \begin{cases} 1, & \text{if } I(i) \geq T \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

Here, T is a dynamic threshold based on a value of the importance scores of all channels, allowing for adaptive pruning intensity based on model complexity and dataset specifics, such that, the total computational cost of the retained channels must not exceed a predefined limit L:

$$\sum_{i=1}^{c} w_i \cdot x_i \leq L \tag{12}$$

Here, w_i represents the computational cost of channel i. To ensure that the pruned model maintains a baseline level of accuracy, we also impose a minimum performance constraint:

$$\text{Acc}(x) \geq A_{\text{min}} \tag{13}$$

Here, A_{min} is the minimum acceptable accuracy for the pruned model, to prevent over-pruning of similar channels, and ensure a diverse set of features is retained:

$$\text{Diversity}(x) \geq D_{\text{min}} \tag{14}$$

Here, D_{min} is the minimum required diversity level. The solution to this problem involves using an algorithm that can iteratively adjust the vector x to find the configuration that optimises the objective function $P(x)$, while satisfying all the necessary constraints.

IV. Complexity Analysis

The Genetic Algorithm method we utilised for layer selection strategically activates a subset of layers, thereby reducing the total number of parameters that need to be updated during back propagation. This selective training scales down the computational complexity from $O(n.m)$, where n is the total number of layers, and m is the number of parameters per layer typically considered in full-network training, to $O(k.m)$, where $k < n$ and k represent the number of selected layers. It is fair to consider that genetic operations also contribute to complexity, however, $O(k.m) >> O(mutation) + O(crossover) + O(selection)$, and the computational cost is predominantly determined by the neural network training itself. For space complexity, since each individual's fitness is computed once and stored, this does not affect the time complexity, but contributes a bit to the overall memory requirements of the system. Pruning the neural network requires direct operations, as it iterates over all parameters of the neural network to apply the pruning criteria. When the feature pruning method is integrated with Grad-CAM, it focuses on pruning parameters based on their relevance, which can be observed by their heatmap values. By pruning non-critical features, the number of computations required during the forward and backward passes is significantly reduced. The computational complexity for inference, $O(p)$ for p parameters in the model, is effectively reduced to $O(q)$, where $q < p$ reflects the pruned model's parameters. This pruning not only accelerates inference, but also reduces the model's memory footprint, making it suitable for real-time applications on devices with limited computational capabilities. However, it is important to note that Grad-CAM pruning requires additional memory overhead to store heatmaps and pruning masks. In terms of space complexity, storing these additional components requires $O(k.n)$ space, where $k.n$ represents the size of the feature maps at each layer.

V. Proposed Framework

This section provides a detailed description of our proposed ToSiM-IoT framework. The first subsection describes the genetic algorithm based selection mechanism for layers and the second subsection highlights the heatmap visualisation technique for channel pruning.

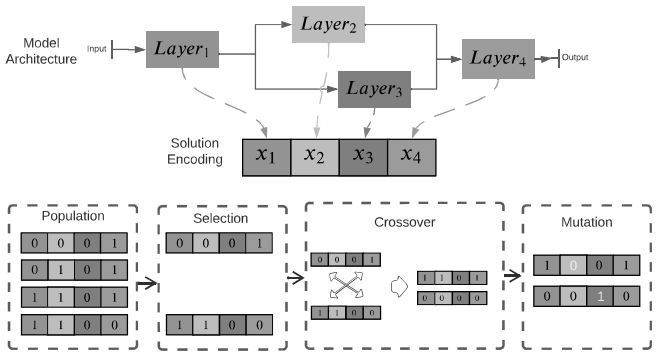

Fig. 3: Solution encoding in our layer selection mechanism.

A. Optimising Layer Selection with Genetic Algorithm

The solution encoding consists of an array that defines each layer's selection during the training process. A genetic algorithm is used to manipulate this encoding by modifying the array to discover an optimal mix of active and inactive layers. Selecting the optimal layers for training while keeping the remaining layers frozen (to preserve their weights) constitutes a complex computational challenge. It is crucial for improving the model's functionality, especially considering the complexity and scale of neural networks which make manual

selection non-feasible. Therefore, an algorithm is needed to efficiently navigate the potential configurations and identify the most effective ones for deployment. Genetic algorithms are particularly suited for this task due to their ability to handle complex optimisation problems. Inspired by the genetic process of combining attributes, these algorithms utilise methods such as mutation, crossover, and selection to refine solutions progressively. For the layer selection mechanism, the GA will enable a systematic exploration of layer combinations, identifying those layers that can significantly reduce the computational demand and enhance the performance.

Algorithm 1 ToSiM-IoT Layer Selection

Input: G, p_m, p_c, K, C, c_i, v_i, α, β, N, M
Output: Return the best chromosomes based on fitness

Initialise: Generate an initial population P of M chromosomes. Each chromosome $x^{(j)}$ is a binary vector of length N.
function FITNESS(x)
    return $\alpha \cdot \sum_{i=1}^{N} v_i x_i - \beta \cdot \max\left(\sum_{i=1}^{N} c_i x_i - C, 0\right)$
end function
for $gen = 1$ to G **do**
    Select K chromosomes from P based on fitness to form S.
    for each pair $(x^{(a)}, x^{(b)})$ in S **do**
        $x^{(new)} \leftarrow$ Crossover$(x^{(a)}, x^{(b)})$ with probability $p_c = 0.9$
        $x^{(mutated)} \leftarrow$ Mutate$(x^{(new)}, p_m)$ with mutation probability $p_m = 0.2$
        Replace weaker chromosomes in P with $x^{(mutated)}$
    end for
    Evaluate $FITNESS()$ of each chromosome
    if termination condition is met **then**
        break
    end if
end for

Fig. 3 illustrates our solution encoding mechanism and its manipulation by the genetic algorithm. The selection algorithm chooses solutions in each iteration, followed by the crossover and mutation operations. Mutation involves flipping bits ('0' to '1' or vice versa) by targeting specific indices based on a random distribution. These operations are repeated multiple times to generate new probable solutions. Individuals with higher fitness are preferably selected through *Binary Tournament Selection* mechanism, ensuring those more suited to the problem are more likely to reproduce. A 90% chance of *Single Point Crossover* promotes genetic diversity by mixing genes from pairs of parents, while a 20% *Bit Flip Mutation* rate introduces new genetic variations to explore the solution space effectively. Algorithm 1 runs until a specific number of fitness evaluations are obtained, maintaining constant population size. This setup provides a balance between exploring new possibilities and exploiting already-known good solutions, aiming for an efficient search of optimal solutions with a limited number of computational steps.

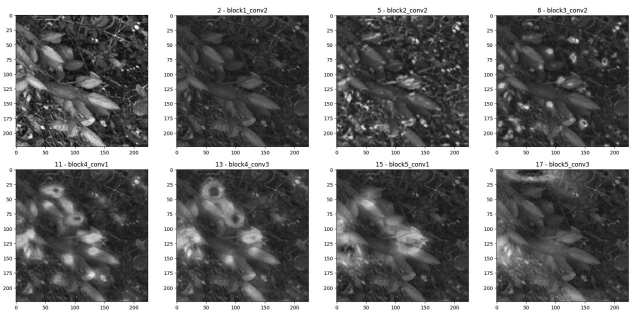

Fig. 4: Visualisation of the data and its corresponding class activation heatmap for identified labels.

B. Efficient Feature Mapping for Channel Pruning

Modern deep neural networks consist of various types of convolutional layers, and the execution runtime during an inference phase is dominated by the evaluation of these layers. In order to speed up the inference process, our strategy involves pruning complete feature maps by utilising the analytical capability of Grad-CAM and performing heatmap visualisation to identify the impactful features. Grad-CAM – stands for Gradient-weighted Class Activation Mapping [26], is a technique that can enhance the interpretability of CNNs by analysing gradients in the final layer and quantifying the significance of each neuron in the decision-making process. This technique visualises the importance of features using heatmaps, where colour intensity represents values between 0 and 1 to show the scale of significance in the data. Fig. 4 below shows a step-by-step visualisation of the active areas in an image that was highlighted during the feature identification process. We have considered a threshold variable for establishing the separation between the importance value of features and assisting the pruning process to eliminate less critical connections, thus optimising model performance and efficiency.

We have designed the Algorithm 2 that shows the step-by-step approach of our pruning workflow. The process begins with the identification and extraction of the target CNN layer from the neural network model. Following this, the weights of the designated layer are extracted before further analysis and manipulation are performed. Next, the channel importance is calculated by summing up heatmap values per channel, indicating their contribution to the model's output and assistance in pruning decisions. The selection of retained channels is based on a pre-specified importance threshold value. Channels with cumulative heatmap values surpassing this threshold are selected for retention, while those with values below it are considered for removal. This process ensures the retention of only those channels that significantly contribute to the network's effectiveness. The pruning phase involves the targeted deletion of weights and biases associated with channels that are deemed non-essential (below the threshold). This crucial step aims to decrease the model's computational demands while preserving its true predictive characteristics. The values of L and A_{min} considered in this work are 300mins and 65%. Following the pruning process, a new CNN layer is

created. This layer mirrors the original layer in terms of its architectural parameters (e.g., kernel size, strides, padding, activation function) but differs in the number of filters, which are adjusted to match the count of channels retained. The pruned weights and biases are then allocated to the newly instantiated CNN layer, ensuring that the layer is properly equipped to perform its function within the network. The final step replaces the original convolutional layer with its pruned version, ensuring the model's integrity and connectivity.

VI. Experimental Design and Setup

The section describes the experimental tools and detailed configurations that have been utilised during evaluation of our ToSiM-IoT framework.

A. Tools and Hardware Configuration

In our setup, we have used TensorFlow to build and modify state-of-the-art VGG16 and InceptionV3 models. We also used JMetalPy[1] for genetic algorithm operations in our layer selection mechanism. For pruning, we leveraged TensorFlow[2] and Keras built-in capabilities to implement our Grad-CAM solution. The heatmap generation in the approach uses TensorFlow's Keras API[3], where a function builds a model to observe a specific convolutional layer's output and model's predictions for an image array. This process generates heatmap values highlighting key image regions and shows the API's ability to analyse and interpret model behaviour. We performed the ML operations on a virtual instance of dual-core Intel(R) Xeon(R) CPU at 2.00GHz, 12 GB of RAM, and an NVIDIA Tesla T4 GPU with 16 GB of memory. The system has 79GB of total storage, with 27GB used and 52GB available. Datasets were pre-loaded from Google Drive as virtual instances to minimise data transfer overhead and optimise the execution.

Our agricultural use case considers collecting data from IoT devices at the network's edge and processing it on a Field-Side Unit (FSU) using the DeepWeeds [27] image classification plant-based dataset. The DeepWeeds dataset is the first large, public dataset for Australian range-land weeds, having 17,509 images of eight common species of weeds from eight regions spread across northern Australia. This dataset is designed to support the development of classification methods, enabling the use of robotic weed control in challenging environments and highlighting the role of machine learning in improving weed management and agricultural tasks.

B. Estimation of Power Consumption

To estimate the power consumption of Nvidia T4 GPU, based on its running time, we have used the following mathematical model:

Let P_{load} be the power consumption in watts when resource node (GPU) is running and let T_{load} be the total time for which our node is executing the tasks.

[1]https://github.com/jMetal/jMetalPy
[2]https://www.tensorflow.org
[3]https://www.tensorflow.org/guide/keras

Algorithm 2 ToSiM-IoT Channel Pruning

Input: model, L, T, A_{min}
Output: prunedModel

function CALC_IMPORTANCE($model$, $layerName$)
  $layer \leftarrow$ GETLAYER($model, layerName$)
  $weights, biases \leftarrow$ GETWEIGHTS($layer$)
  $importance \leftarrow$ empty map
  for $channel$ in $layer$ **do**
    $importance[channel] \leftarrow H(x, y, channel)\, dx\, dy$
  end for
  return $importance$
end function
$importance \leftarrow$ CALCULATEIMPORTANCE($model, layerName$)
$retainChannels \leftarrow$ empty list
for $channel, imp$ in $importance$ **do**
  if $imp \geq T$ **then**
    APPEND($retainChannels, channel$)
  end if
end for
function OPTIMIZELAYER($model$, $retainChannels$)
  $newWeights, newBiases \leftarrow$ arrays of zeros with sizes based on $retainChannels$
  $compCost \leftarrow 0$
  $modelAcc \leftarrow 0$
  for $channel$ in $retainChannels$ **do**
    $newWeights[channel], newBiases[channel] \leftarrow weights[channel], biases[channel]$
    $compCost \leftarrow compCost + w_{channel}$
    $modelAcc \leftarrow modelAcc + v_{channel}$
  end for
  if $compCost \leq L$ and $modelAcc \geq A_{\text{min}}$ **then**
    $layer \leftarrow$ CREATELAYER($len, kSize, stride, pad, act$)
    SETWEIGHTS($layer, newWeights, newBiases$)
    REPLACELAYER(model, layerName, layer)
  end if
end function
OPTIMIZELAYER(model, retainChannels)
return prunedModel

Given that $P_{load} = 74$ watts. The total power consumption (P_{total}) in watt-hours (Wh) can be calculated as follows:

$$P_{total} = (P_{load} \times T_{load}) \tag{15}$$

Similarly, in kilowatt-hours (kWh), the power can be calculated as:

$$P_{total} = (P_{load} \times T_{load})\,/1000 \tag{16}$$

This model allows us to estimate the T4 GPU's power consumption over the duration for which GPU was consuming power and executing the ML tasks utilised for evaluation in our experimentation.

C. Experimental Configuration

This section describes the configuration details for executing the layer selection and feature pruning mechanism in this work.

1) Layer Selection Setup: This configuration focuses on setting up the layer selection mechanism of our ToSiM-IoT framework. The steps are as follows:

a) Data Splitting: The dataset is divided into three segments: 60% for training, 20% for validation, and 20% for testing. This division is used to ensure effective model training, fine-tuning, and performance assessment of the framework.

b) Data Preparation: The process begins with resizing images to standardised input dimensions, followed by data augmentation techniques such as flipping and rotating. Additionally, these images are normalised, by converting pixel values from integers to floats and scaling them to the range of $[0, 1]$, and then batched during the training process.

c) Model Customisation: This phase involves transforming pre-trained InceptionV3 and VGG16 models, which were initially trained with the ImageNet dataset. The models are customised with DeepWeeds dataset by appending new layers and freezing the original layers to preserve learned/retained features. This customisation aims to leverage the pre-existing knowledge of the models while making them relevant for the new tasks.

d) Initialise Training: The models are trained with defined hyperparameters, including 10 epochs and batch sizes of 20. Additionally, a genetic algorithm is utilised to fine-tune the model and customise it.

e) Genetic Algorithm Based Selection: A similar model adjusted based on various parameters undergoes brief training to learn the changes and improve model adaptability. The generated models are then evaluated with a focus on accuracy and the obtained results are used to assist in genetic algorithm driven refinements, continuously improving model outcomes. The genetic algorithm population considered in our experimentation is 10.

f) Evaluation and Analysis: After training, the models are evaluated on the test dataset to determine their generalisation capabilities. This evaluation helps in assessing the effectiveness of the training and the customisation process. The performance metrics are plotted over the epochs to analyse the model's learning behaviour and fit with respect to the data.

2) Feature Maps Pruning Setup: After model training, this part of our configuration is designed to use a heatmap visualisation technique for pruning the features. The steps are as follows:

a) Data Preparation: The process begins by dividing the complete dataset into two separate parts (referred to as Dataset A and Dataset B) to simplify the training and evaluation process.

b) Initial Training: In this step, Dataset A is utilised as a base for the model's initial training process. This phase establishes the benchmark for the model's performance and provides insights into their learning capabilities.

c) Pruning Phase: Following the initial training, the model undergoes a pruning process where we selectively remove some of the model components, such as weights, biases, or neurons, that have minimal impact on the output. The aim is to simplify the model's structure and enhance its operational efficiency. We have used the approach outlined in section V-B to selectively remove the model components and reduce its size.

d) Fine-Tuning with Dataset B: After pruning, Dataset B is used to fine-tune the selected model. This phase aims to perform minor adjustments to our model's parameters to refine its performance, focusing on recovering or improving aspects, and generalisations that potentially gets diminished or removed during our earlier pruning phase. We have used both Dataset A and B as training sets within this experimental approach. The main difference in their usage is that – Dataset A is allocated for initial model training whereas Dataset B is designated for post-pruning fine-tuning.

e) Evaluation and Analysis: The final step involves a comprehensive evaluation of the model after its fine-tuning. This analysis focuses on various performance metrics to verify the rate of the improvements we were able to achieve during the pruning and fine-tuning phases.

VII. Results and Evaluation

This section highlights the experimental results and our findings after evaluating the proposed ToSiM-IoT framework. Fig. 5 compares the training times (in seconds) for one epoch of the InceptionV3 and VGG16 models across different percentages of trainable layers, ranging from 10% to 100% of the total layers. The training time increases for both models as the percentage of trainable layers increases, which is expected since more parameters require more computational effort to update the model. Moreover, VGG16 has consistently higher training time across all percentages of trainable layers than InceptionV3. This can be attributed to the architectural complexities and the overall number of parameters in VGG16, known for its simplicity in structure but high computational cost due to the depth and size of its fully connected layers.

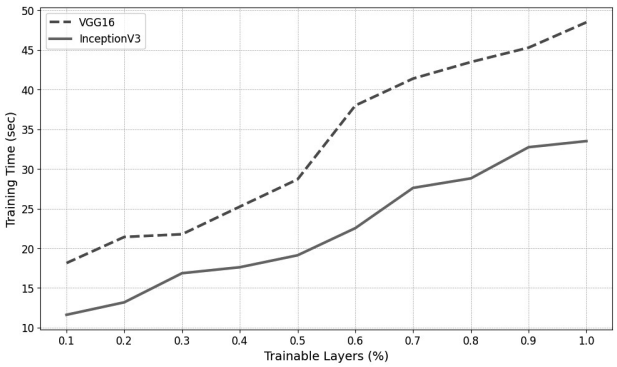

Fig. 5: Total training time with different percentage of trainable layers

The Table I below summarises the power consumption, measured in watts, of two deep learning models (per epoch) during the training sessions with different percentages of their layers trainable. Specifically, it shows measurements for three scenarios: when 10%, 50%, and 100% of the layers are trainable. For both models, the power consumption increases as the percentage of frozen layers decreases. This progressive increase in power can be attributed to the computational

Trainable (%)	*InceptionV3*	*VGG16*
0.1	11.61	18.13
0.5	19.11	28.68
1.0	33.50	48.48

TABLE I: Power consumption (watts) of models during training (one epoch) with different percentages of trainable layers.

demand associated with updating a greater number of parameters. With more layers participating in the training process, the models require more computational resources, leading to higher energy usage. The comparison between InceptionV3 and VGG16 also reveals the inherent differences in their architectural efficiency. InceptionV3 consistently requires less power than VGG16 across all scenarios, indicating it is a more energy-efficient model for training tasks. This can also be due to its more optimised architecture, which might involve fewer parameters or more efficient operations compared to the deeper and more parameter-intensive architecture of VGG16.

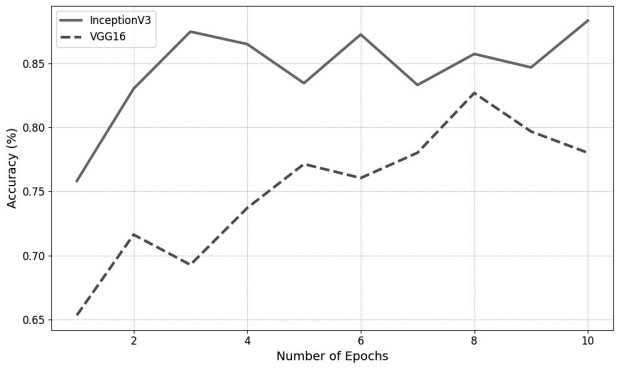

Fig. 6: Accuracy benchmarks on DeepWeeds dataset for model training.

Fig. 6 outlines the validation accuracies of two convolutional neural network models we have considered for optimisation in our framework. We performed the model training using the layer selection mechanism described above in previous sections. The accuracies are measured across a span of 10 epochs, providing insight into the learning efficiency and performance of each model over time. InceptionV3 demonstrates a starting accuracy of 75.81% at epoch 1, which exhibits a generally upward trend, peaking at 88.32%. The overall trend indicates improvement in model performance with continued training and shows the strong ability of InceptionV3 to generalise from training data to unseen validation data. On the other hand, VGG16 starts with a bit lower accuracy of 65.33% at epoch 1 and follows an upward trend by peaking at 82.67%. The observed differences in accuracy between InceptionV3 and VGG16 can be attributed to factors inherent in their architectures and the complexity of the task. InceptionV3 consists of an Inception module, which is designed to efficiently manage computational resources while capturing complex features at various scales. This design contributes to its higher accuracy and ability to better generalise during the validation phase. VGG16 model is characterised by its simplicity and depth with repetitive convolutional blocks and demonstrates a considerable capacity for feature extraction. However, its architecture sometimes have higher susceptibility to overfitting and require more data or regularisation to achieve optimal generalisation, which explains its lower performance compared to InceptionV3 in this scenario.

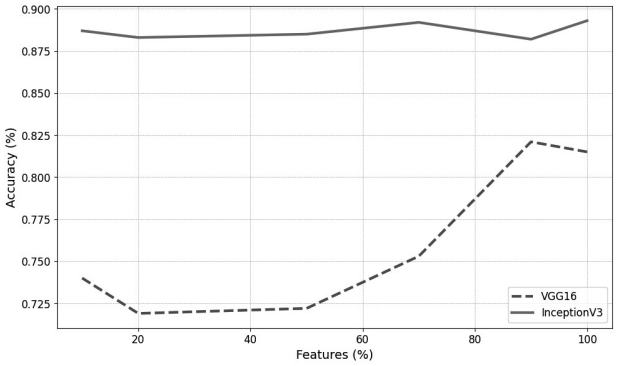

Fig. 7: Change in model accuracy with different percentages of features retained.

The heatmap-based pruning approach when evaluated on two available models (Fig. 7) shows that there is a general trend of increase in accuracy when higher percentage of features are retained. This pattern is observed because retaining more features likely preserves more relevant information necessary for accurate predictions. Models with deeper or more complex architectures, like the VGG, show a more significant impact from feature pruning, requiring a higher percentage of features to be retained for optimal accuracy. InceptionV3's design, incorporating modules with parallel convolutions of different kernel sizes, allows it to capture information at various scales effectively and thus ensuring high accuracies across all percentages.

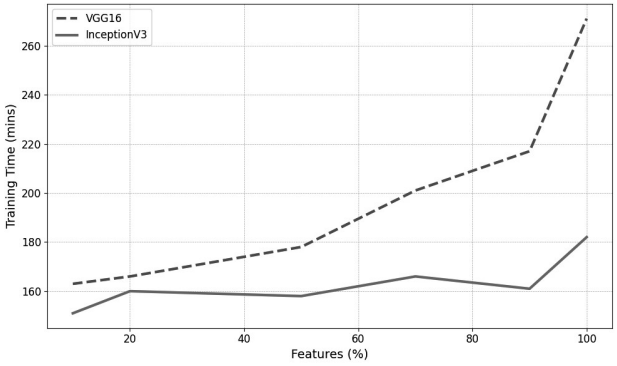

Fig. 8: Effect on training time with changing percentages of retained features.

The pruning process aims to reduce the computational complexity of the models by selectively removing less important channels based on their contributions using heatmaps. This selective reduction can significantly affect the training time of the models, as observed in Fig. 8. The VGG16 model shows an increasing trend in training time as the percentage of features retained increases. Over the range of feature percentages, the training time increases from 163 minutes to 271 minutes gradually. The significant change in training time indicates

that the pruning process effectively removes redundant or less important features without compromising the model's predictive power significantly. For the InceptionV3 model, the training time initially decreases when the percentage of retained features goes from 10% to 50%; and then increases slightly when feature retension percentage is increased to 100%. This suggests that InceptionV3 may achieve optimal performance with a moderate level of feature retention, where the balance between model complexity and computational load is ideal. Beyond this point, the slight increase in training time as more features are retained could be due to the added computational burden outweighing the benefits of additional features.

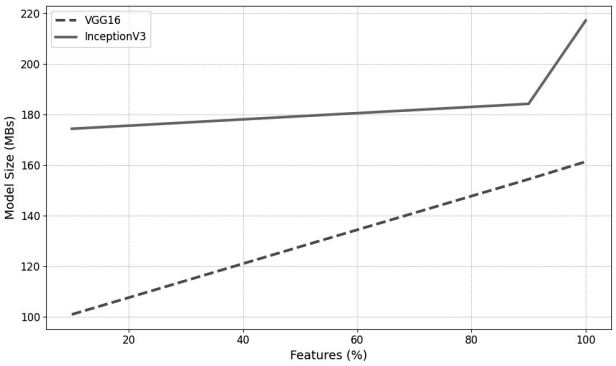

Fig. 9: Change in model size with different percentages of features retained.

Fig. 9 shows that the size of a model is directly related to its complexity and the amount of information it can process and store. The increase in model size with a high percentage of features indicates a near-linear relationship between the two factors. More features mean more channels in the convolutional layers, which in turn increases the number of weights and biases that need to be stored, thereby increasing the model size. The InceptionV3 model shows a more complex pattern. The size increases from 174.41 MBs at 10% feature retention to 217.23 MBs at 100% feature retention. It is observed that the increase in size is not as linear as with VGG16, especially in the higher percentages of feature retention. The larger initial size of InceptionV3 compared to VGG16 at lower feature retention levels suggests that even with fewer features, InceptionV3's complex architecture requires more parameters to be stored. This complexity contributes to the overall increase in model size as more number of features are retained.

Features (%)	*InceptionV3*	*VGG16*
0.1	164.03	219.53
0.5	194.86	238.03
1.0	224.46	334.23

TABLE II: Power consumption (watts) of models after pruning with different percentages of retained features.

The given Table II illustrates the effect of pruning process on power consumption of both the models at specific levels (10%, 50%, and 100%) of retained features. For both models, the power consumption increases as the percentage of retained features increases. The data indicates that VGG16 consumes more power than InceptionV3 across the same levels of feature retention, suggesting that VGG16's architecture may be inherently more power-intensive or less efficient at these high levels of feature reduction. Additionally, the increasing trend in power consumption with higher feature retention percentages highlights the relationship between model complexity and power needs. It shows that even slight increases in retained features can significantly impact power consumption, likely due to the increased computational workload associated with processing a higher number of features.

Model (→)	*InceptionV3*			*VGG16*		
Class (↓)	*P*	*R*	*F1*	*P*	*R*	*F1*
Chinese apple	88.0	90.5	89.2	85.1	80.2	82.6
Lantana	95.0	94.5	94.7	81.6	85.4	83.5
Parkinsonia	98.9	92.4	95.6	90.2	88.7	89.4
Parthenium	92.0	92.5	92.2	77.5	82.1	79.8
Prickly acacia	92.0	93.6	92.8	85.6	84.3	84.9
Rubber vine	90.0	92.8	91.4	88.7	80.4	84.3
Siam weed	93.0	92.7	92.8	83.8	88.3	86.0
Snake weed	93.8	95	94.4	84.9	86.1	85.4
Negatives	94.8	93.4	94.1	90.4	92.1	91.2

TABLE III: Class-wise performance metrics for InceptionV3 and VGG16 models. P: Precision; R: Recall; F1: F1-Score

Analysing the process of structured training and fine-tuning process, which involves initial training with Dataset A, followed by model pruning and subsequent fine-tuning with Dataset B, provides a comprehensive understanding of the approach and its implications on the model performance. This approach utilises the distinct characteristics of two datasets to enhance model performance progressively. The initial training on Dataset A establishes a foundation, allowing the model to learn general features and patterns. The pruning process then refines the model by eliminating less significant features, reducing complexity and computational demand. Finally, fine-tuning with Dataset B adjusts the model to perform well in specific, possibly more challenging, real-world scenarios. This method highlights the adaptability of the models to new information and their capacity for incremental learning. The maximum accuracy achieved after re-training by VGG16 and InceptionV3 models is 86.8% and 93.1% respectively. A detailed class-wise description of other performance metrics is shown in Table III. Comparing the accuracies with the ones recorded after the pruning process, both models exhibit significant improvements after fine-tuning. The VGG16 model observed an average of 6.3% increase over the initial results whereas InceptionV3 achieved improvements of 4.3%.

VIII. Summing up the Findings

The experimental results highlight the differences in performance dynamics of VGG16 and InceptionV3 models under varying configurations of trainable layers, with particular emphasis on training time, power consumption, and accuracy benchmarks. We observed that VGG16 has longer training times and higher power consumption across all degrees of trainable layers. On the other hand, the InceptionV3 has higher accuracy and model size over the different percentages of retained features. The InceptionV3 model performs about 32.63% better than the VGG16 model in terms of execution

time with different values of trainable layers. The average accuracy of InceptionV3 model is around 6%-7% higher than VGG16 across all degrees of retained features. As the percentage of retained features is reduced, the model size for InceptionV3 and VGG16 decreases by 20.27% and 37.5% as we go from 100% to 10%. Moreover, we also observed that the average power consumption of InceptionV3 models is 26.33% better than the VGG16 models. The VGG16 model showed a 66.26% decrease in training time when the number of features are reduced to 10%. In contrast, the InceptionV3 model exhibits a 27.27% decrease over the same measure. The results also show that the InceptionV3 model is approximately 18.90% more efficient in training time compared to the VGG16 model. A comparison of our ToSiM-IoT framework with some other optimisation techniques is shown below in Table IV. Comparing the two architectures highlights that the model structure, number of features, and trainable layers are some of the critical parameters that directly affect the performance of AI models and can be a deciding factor for the efficient implementation of ML optimisation approaches on limited resource nodes. Moreover, based on the time, accuracy, and space requirements of limited resource nodes; a decision can be made using our framework to select the optimal values of parameters that will generate the best possible results for the end-users and applications for which the ML models are designed.

Techniques	*InceptionV3*	*VGG16*
L1-Norm Based Pruning	85.3	78.6
L2-Norm Based Pruning	88.4	82.2
Quantization	87.5	81.6
Random Layer Freezing	82.7	79.7
Group Lasso Regularisation	87.8	83.4
ToSiM-IoT	93.1	86.8

TABLE IV: Comparison of Model Accuracies Across Different Optimisation Techniques.

IX. Exploring Aspects for Further Optimisation

We observed that our ToSiM-IoT framework achieves desirable results in reducing the training time and model size, however, in order to contribute towards more sustainable practices for IoT applications, it does present certain limitations that can provide options for future exploration and enhancement.

Our approach for layer selection is currently managed as a single-objective optimisation problem, where the number of layers are predetermined by the user. For improvement, we can consider the number of layers as an additional objective function along with model accuracy, using a multi-objective optimisation method. This will allow for a balanced trade-off between maximising accuracy and minimising the number of layers, addressing this problem as a connected optimisation challenge. Moreover, the pruning approach is a constraint satisfaction problem where the aim is to meet specific conditions defined by a predetermined threshold. For improvements, we can consider the pruning approach as an optimisation challenge, rather than simply satisfying the set constraints. This adjustment can decrease the variability and transform the pruning threshold value into an adjustable variable. Objective functions can then be used to evaluate the most effective areas for pruning, identifying the optimal threshold that aligns best with a particular model and dataset.

From our experiments, we also observed that each neural network has distinct characteristics that can significantly influence the optimisation framework. These characteristics include layer size, total number of parameters, model branching (which allows for the concurrent execution of two or more layers), and the parameter count within individual layers. We can explore the possibility of designing a method that examines these specific aspects of each network to support ongoing testing and improvements. By understanding these elements, we aim to restructure models to promote energy-efficient and sustainable operations, thereby improving the performance of ML-based tasks in IoT infrastructures.

X. Conclusion

The paper introduces ToSiM-IoT, a sustainable optimisation framework for enhancing the efficiency of Machine Learning (ML) tasks in the Internet of Things (IoT) based applications. Our framework leverages the ability of genetic algorithms and heatmap visualisation techniques for layer selection and pruning in ML models. Through systematic experimentation with VGG16 and InceptionV3 models using the DeepWeeds dataset, we demonstrated that our proposed methodology significantly reduces model size and training time, without compromising on accuracy. Results also show that the strategy of selectively freezing neural network layers and pruning less significant parameters addresses the critical challenge of limited-resource availability of IoT infrastructure. The framework can act as the potential procedure for the deployment of ML and AI capabilities in environments where computational, storage, and energy resources are in limited capacity. We plan to extend the optimisation techniques to other ML models and different real-world IoT applications to understand domain-specific challenges and adjust our approach accordingly. We also plan to develop methods that approximate power consumption and utilise them as heuristic measures in an optimisation problem. We aim to design ML models that are capable of predicting power consumption based on the hardware used and applied computational model. This decision can be utilised to enhance the energy efficiency structure and ensure improved resource allocation in IoT infrastructures.

References

[1] C.-J. Wu, R. Raghavendra, U. Gupta, B. Acun, N. Ardalani, K. Maeng, G. Chang, F. Aga, J. Huang, C. Bai, *et al.*, "Sustainable AI: Environmental implications, challenges and opportunities," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 795–813, 2022.

[2] A. Kaushal, O. Almurshed, A. Alabbas, N. Auluck, and O. Rana, "An edge-cloud infrastructure for weed detection in precision agriculture," in *IEEE Intl Conf on Pervasive Intelligence and Computing (PiCom)*, pp. 0269–0276, 2023.

[3] L. Hu and Q. Ni, "Iot-driven automated object detection algorithm for urban surveillance systems in smart cities," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 747–754, 2017.

[4] A. Alabbas, A. Kaushal, O. Almurshed, O. Rana, N. Auluck, and C. Perera, "Performance analysis of apache openwhisk across the edge-cloud continuum," in *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*, pp. 401–407, IEEE, 2023.

[5] R. Marculescu, D. Marculescu, and U. Ogras, "Edge ai: Systems design and ml for iot data analytics," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3565–3566, 2020.

[6] S. Boovaraghavan, A. Maravi, P. Mallela, and Y. Agarwal, "Mliot: An end-to-end machine learning system for the internet-of-things," in *Proceedings of the International Conference on Internet-of-Things Design and Implementation*, pp. 169–181, 2021.

[7] P. Prakash, J. Ding, R. Chen, X. Qin, M. Shu, Q. Cui, Y. Guo, and M. Pan, "Iot device friendly and communication-efficient federated learning via joint model pruning and quantization," *IEEE Internet of Things Journal*, vol. 9, no. 15, pp. 13638–13650, 2022.

[8] E. S. Jeon, A. Som, A. Shukla, K. Hasanaj, M. P. Buman, and P. Turaga, "Role of data augmentation strategies in knowledge distillation for wearable sensor data," *IEEE internet of things journal*, vol. 9, no. 14, pp. 12848–12860, 2021.

[9] J. Azar, A. Makhoul, M. Barhamgi, and R. Couturier, "An energy efficient iot data compression approach for edge machine learning," *Future Generation Computer Systems*, vol. 96, pp. 168–175, 2019.

[10] H. Ma, Z. Zhou, X. Zhang, and X. Chen, "Towards carbon-neutral edge computing: Greening edge ai by harnessing spot and future carbon markets," *IEEE Internet of Things Journal*, 2023.

[11] S. Zhu, K. Ota, and M. Dong, "Energy-efficient artificial intelligence of things with intelligent edge," *IEEE Internet of Things Journal*, vol. 9, no. 10, pp. 7525–7532, 2022.

[12] S. Zhu, K. Ota, and M. Dong, "Green ai for iiot: Energy efficient intelligent edge computing for industrial internet of things," *IEEE Transactions on Green Communications and Networking*, vol. 6, no. 1, pp. 79–88, 2021.

[13] E. Baccour, N. Mhaisen, A. A. Abdellatif, A. Erbad, A. Mohamed, M. Hamdi, and M. Guizani, "Pervasive ai for iot applications: A survey on resource-efficient distributed artificial intelligence," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 4, pp. 2366–2418, 2022.

[14] L. Xiao, X. Wan, X. Lu, Y. Zhang, and D. Wu, "Iot security techniques based on machine learning: How do iot devices use ai to enhance security?," *IEEE Signal Processing Magazine*, vol. 35, no. 5, pp. 41–49, 2018.

[15] F. Bu and X. Wang, "A smart agriculture iot system based on deep reinforcement learning," *Future Generation Computer Systems*, vol. 99, pp. 500–507, 2019.

[16] A. Canziani, A. Paszke, and E. Culurciello, "An analysis of deep neural network models for practical applications," *arXiv preprint arXiv:1605.07678*, 2016.

[17] D. Li, X. Chen, M. Becchi, and Z. Zong, "Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus," in *2016 IEEE international conferences on big data and cloud computing (BDCloud), social computing and networking (SocialCom), sustainable computing and communications (SustainCom)(BDCloud-SocialCom-SustainCom)*, pp. 477–484, IEEE, 2016.

[18] M. Tan and Q. Le, "Efficientnetv2: Smaller models and faster training," in *International conference on machine learning*, pp. 10096–10106, PMLR, 2021.

[19] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*, pp. 6105–6114, PMLR, 2019.

[20] Y. Hu, S. Sun, J. Li, X. Wang, and Q. Gu, "A novel channel pruning method for deep neural network compression," *arXiv preprint arXiv:1805.11394*, 2018.

[21] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.

[22] Z. Chen, Z. Chen, J. Lin, S. Liu, and W. Li, "Deep neural network acceleration based on low-rank approximated channel pruning," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 4, pp. 1232–1244, 2020.

[23] Z. Song, B. Fu, F. Wu, Z. Jiang, L. Jiang, N. Jing, and X. Liang, "Drq: dynamic region-based quantization for deep neural network acceleration," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1010–1021, IEEE, 2020.

[24] C. Wu, F. Wu, L. Lyu, Y. Huang, and X. Xie, "Communication-efficient federated learning via knowledge distillation," *Nature communications*, vol. 13, no. 1, p. 2032, 2022.

[25] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, "Estimation of energy consumption in machine learning," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 75–88, 2019.

[26] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," pp. 618–626, 2017.

[27] A. Olsen, D. A. Konovalov, B. Philippa, P. Ridd, J. C. Wood, J. Johns, W. Banks, B. Girgenti, O. Kenny, J. Whinney, *et al.*, "Deepweeds: A multiclass weed species image dataset for deep learning," *Scientific reports*, vol. 9, no. 1, p. 2058, 2019.