Inference and Training Efficiency in Pruned Multilayer Perceptron Networks

Amirhossein Douzandeh Zenoozi[1], Laura Erhan[1], Antonio Liotta[1*], Lucia Cavallaro[2]

1 Department of Engineering, Free University of Bozen-Bolzano, Bolzano, Italy
2 Data Science Department, Institute for Computing and Information Sciences, Radboud University, Nijmegen, The Netherlands

* antonio.liotta@unibz.it

Abstract

This study explores how pruning strategies can improve the efficiency of deep neural networks (DNNs), which are widely used for tasks like image processing, medical diagnosis, *etc.* Although DNNs are powerful, they often contain weaker connections that can lead to increased energy consumption both during training and inference. To address this, we compare two pruning approaches: global pruning, which applies to all layers of the network, and layer-wise pruning, which focuses on the hidden layers. These approaches are tested across two MLP models, small-scale and medium-scale, and are then extended to a VGG-16 model as a representative example of Convolutional Neural Networks (CNNs). We evaluate the impact of pruning on five datasets (MNIST, FashionMNIST, EMNIST, CIFAR-10, and OctMNIST), and considering different sparsity levels (50% and 80%). Our results show that, in comparison to the benchmark dense networks (0% sparsity), layer-wise pruning offers the best trade-offs, by consistently reducing inference time and inference energy usage while maintaining accuracy. For example, training the small-scale model with the MNIST dataset and 50% sparsity led to a 33% reduction in inference energy usage, 33% in inference time, and only a negligible 0.49% decrease in accuracy. Furthermore, we investigate training energy consumption, CO2 emissions estimations, and peak memory usage, which again leads to choosing the layer-wise approach over global pruning. Overall, our findings suggest that layer-wise pruning is a practical approach for designing energy-efficient neural networks, particularly in achieving efficient trade-offs between performance and energy consumption.

Author summary

In this work, we explore how to make deep learning models more efficient by removing the weaker connections, through a method known as “pruning”. These models are widely used in everyday applications, from medical tools to smart devices. However, the energy consumption of dense “unpruned” models, while necessary for their configuration, is often suboptimal. The large number of connections in a dense model requires more energy to process inputs and compute the best output class.

By reducing the number of connections, pruning strives for maintaining the model performance close to that of its dense counterpart, while using less energy. This makes pruning an effective way to improve energy efficiency, both during the model training and at model inference time. The reduced computational load results in a more energy-efficient model, which is especially beneficial for devices with limited power.

We tested different pruning techniques and compared their performance in three different deep learning architectures, considering five different domain-specific datasets.
One of our main findings is that using a layer-wise pruning approach leads to significant efficiency gains, at negligible accuracy losses.

Introduction

Deep neural networks (DNN) serve as a robust computational framework [1, 2] to address various tasks in multiple domains, including the Internet of Things (IoT) [3], image processing [4–6], autonomous object management [7, 8], disease diagnosis [9, 10]. Traditionally, working with neural networks involves designing the network architecture, selecting the number of layers, and training the model on a given task. This process focused mainly on creating the network and fine-tuning the parameters for optimal performance. However, recent research has shown that many of the networks designed and trained this way often contain numerous weaker, redundant connections [11]. These redundant connections do not significantly contribute to the model's ability to make accurate predictions; yet they increase the model's complexity and resource usage. As a result, different types of pruning techniques have been proposed to reduce redundancy and make the networks more efficient, both in terms of memory usage and computational cost. These include pre-training pruning, where sparsity is applied before model training starts, and in-training pruning, where the network's connections are adjusted during training. To address this issue, researchers are now focusing on reducing this redundancy by sparsifying connections in the neural network architecture [12–14]. In particular, Sparse Evolutionary Training (SET), a particular type of in-training technique, has proven to be effective at retaining the most important connections, which helps make the model more efficient and perform better.

Prior work shows that the initial layers often capture the most salient input features, while the subsequent ones refine decision boundaries through more intricate connections [15–17]. Specifically, a higher number of connections in the final layer, which connect the hidden layer neurons to the output classes, may enhance classification accuracy by increasing the likelihood of correctly identifying the target class.

In multilayer perceptrons (MLPs) [18], the distribution of critical features across layers plays a pivotal role in determining network efficiency. In fact, the choice of MLPs as the primary focus of this study stems from their architectural simplicity, which allows us to apply the same sparsity techniques uniformly across all layers without exceptions. This design flexibility makes it easier to analyze model behavior in detail and to adjust architectural parameters, such as the number of layers or neurons per layer. In contrast, Convolutional Neural Networks (CNNs) require distinct sparsity techniques for the feature extraction and classification parts, given their different roles and structures. Thus, to strengthen our observations and ensure the generality of our findings, we also expand our experiments to include CNN architectures, specifically using the VGG-16 model.

Building upon our previous work [19], where we examined MLP architectures by measuring model performance across four sparsity techniques (*e.g.,* Pre-Training, In-Training, Post-Training, and Sparse Evolutionary Training or SET, in short) in industrial settings, this study narrows its focus to investigate how Pre-Training and SET (*i.e.*, the most promising ones from our previous study) affect the training performances when specific layers of the network are targeted. In other words, we aim to investigate the impact of the above mentioned pruning techniques to assess whether focusing on the hidden layers (*i.e.*, herein referred to as *layer-wise* pruning approach) can yield more beneficial effects on overall training performance compared to the conventional approach, in which the entire network, including the input and output layers, is sparsified (*i.e.*, herein referred to as *global* pruning approach). In terms of beneficial effects, we are mostly interested on investigating the role of energy consumption and the evaluation of trade-offs between test accuracy, inference time and inference energy efficiency with the goal of achieving more sustainable models.

Our analysis unveils that layer-wise pruning can provide good trade-offs with negligible accuracy loss (*i.e.*, 0.49% in best-case scenarios and nearly 5% in the worst-case ones) along with significant savings in inference time and inference energy, for example, up to a 44% reduction in inference energy and a 33% reduction in inference time in the best-case scenario. Furthermore, in most considered scenarios, layer-wise pruning outperforms global pruning.

For the sake of completeness, we also conducted our analyses on CNN models, specifically using the *VGG-16* architecture, which is widely used in image classification tasks and offers a balanced complexity for exploring pruning techniques [20–24]. From such analysis emerged that the accuracy fluctuation for all pruning scenarios amounts to less than 1%, which is in line with our earlier observations [25].

This current research adds an important piece to the pruning puzzle by providing a deeper understanding of how pruning influences the performance and energy efficiency of MLP models, while also extending the study to CNN architectures, specifically using the *VGG-16* model. Since CNNs are commonly used in image classification tasks and are a standard deep learning architecture, including them in our analysis helps ensuring that our findings are applicable to different network types, providing a broader perspective on the effectiveness of pruning techniques.

The paper is structured as follows. Section 1 reviews pruning techniques, their impact on performance and energy efficiency, and their use in IoT and industrial applications. Section 2 presents all the relevant methodology, including details on sparsity techniques (*i.e.*, Pre-training pruning and SET) and approaches (*i.e.*, layer-wise *vs* global pruning), and the datasets and models included in our experimental design. Next, Section 3 shows the results, focusing on MLP model performance, training energy, and peak memory usage. For the sake of completeness, the exploration of CNN architectures, and the evaluation of a domain-specific dataset are also presented. Finally, Section 4 concludes the paper and outlines directions for future work.

1 Related Work

Recent improvements [14, 26–28] in neural network pruning have highlighted its potential to enhance computational efficiency while maintaining model accuracy. Previous work explored diverse pruning strategies, offering unique approaches to sparsification. These strategies can be broadly categorized into structured and unstructured pruning. Structured pruning involves removing entire units such as neurons, channels, or layers. In contrast, unstructured pruning removes individual weights, which are identified and discarded based on algorithms that measure their contribution to the model's overall performances. By focusing on unstructured pruning, we can algorithmically identify and remove the weakest connections, leading to a more precise and efficient pruning process. As a result, the related works we discuss in this paper primarily focus on this category of pruning.

An exemplary approach to unstructured pruning was proposed by Mocanu *et al.* [13], who introduced Sparse Evolutionary Training (SET). This is a global pruning technique (operating on the whole network), which removes the weaker connections through an evolutionary process. The authors present a method that starts with a sparse neural network initialized through the *Erdős-Rényi* network model [29], where the connections between neurons are randomly assigned with fixed probability. This initial sparse network is then progressively evolved during training, with pruning occurring at each epoch based on an evolutionary strategy. The key idea is to adaptively remove the weakest connections while preserving the model's performance (and its network topological features), using a genetic algorithm to iteratively select and optimize for the most crucial connections.

Similarly, Galchonkov *et al.* [14] investigate the combined effect of two techniques: pruning connections in the neural network to reduce its complexity and pre-processing input data (*e.g.* rotations, shifts, and distortions). These techniques aim to make the training of neural networks more efficient while maintaining high accuracy, particularly in the task of handwritten digit recognition using the MNIST dataset.

In comparison, Kalyanam *et al.* [30] propose an unstructured pruning method for multilayer perceptron (MLP) models with the hyperbolic tangent (*i.e.* tanh) activation, introducing a novel neuron removal strategy. This study targets the neurons with activations clustered near -1, 0, or 1, leveraging the zero-centered properties of the *tanh* function. This method is rather different from other works, which focus on link weights or connections between the neurons [13, 28, 31, 32].

Lee *et al.* [33] present the Single-shot Network Pruning (SNIP) technique, which introduces a method for pruning neural networks once at initialization by measuring connection sensitivity. This approach eliminates the need for the traditional iterative pruning-retraining cycles often required by other pruning methods. Instead of pruning during or after training, SNIP evaluates the importance of each connection based on how much it impacts the loss function using a mini-batch of data. Connections with low sensitivity are pruned in a single shot, making it a highly efficient method for reducing model complexity, without sacrificing accuracy.

Additionally, Fan *et al.* [34] introduces a layer-wise pruning approach that uses mutual information (MI) to prune neurons. In this method, sparsity is applied individually to each layer, allowing for a more targeted pruning process. The technique focuses on retaining neurons that have the highest mutual information with the neurons preserved in the next layer. This approach is similar to Kalyanam *et al.*'s work in that both methods prioritize pruning neurons rather than connections between neurons.

Another important aspect of neural network optimization is sustainability. This includes considering how pruning methods affect not only model size and accuracy but also energy consumption and resource efficiency, both during training and inference.

In recent years, researchers have begun to explore how to make neural networks more energy-efficient and sustainable, ensuring that models are not only smaller but also more efficient in terms of their computational and environmental impact [25, 35–37]. For instance, Kamolov [36] investigates the three different strategies including magnitude-based pruning, structured pruning, and random pruning in two Convolutional Neural Network (CNN) models namely, ResNet18, VGG-16. He considers four different sparsity levels, namely 10%, 30%, 50% and 70%, on both architectures, using the CIFAR-10 dataset. In this work, all models are trained for 10 epochs after applying the sparsity techniques to restore the performance of the models. This research demonstrates that magnitude-based pruning techniques are more effective than both structured and random pruning, in preserving model accuracy while reducing computational complexity. It is also worth pinpointing that the SET technique (see Sect. 2.1.2), which is one of the primary pruning methods used in our study, combines the *Erdős–Rényi* random algorithm with magnitude-based pruning to optimize model performance and direct the network to continuously use the strongest connections.

Li *et al.* [37] introduce a hardware-aware optimization method called IHSOpti, aimed at improving the performance of deep neural networks (DNNs) by leveraging hardware features like pipelining and parallelism. They propose the Polar_HSPG algorithm, which enhances DNN pruning and introduces a residual strategy to reduce redundancies at the layer level. Their evaluation focuses on metrics such as pruning ratio, Top-1 accuracy, and inference time. The research explores how pruning techniques can reduce computational overhead, at a negligible accuracy loss, particularly for models like VGG, ResNet, and RegNet.

Widmann *et al.* [38] investigate the impact of neural network pruning on energy efficiency for IoT devices. The authors apply unstructured pruning based on magnitude and structured pruning using the average percentage of zero (APoZ) method on a *LeNet-5* CNN model. These methods were tested on a *Raspberry Pi Pico* (RP2040 chip) with the FashionMNIST dataset. The authors analyze how pruning at different compression rates (from 2 to 64) affects energy consumption, inference speed, and accuracy. The compression rates are obtained by dividing the original number of parameters to the new one [39]. Their research shows that structured pruning outperforms unstructured pruning in most cases, whilst achieving also notable improvements in energy consumption and inference speed.

Liu *et al.* [40] introduce network slimming, a method for optimizing CNNs by applying sparsity regularization to the scaling factors in Batch Normalization layers. This approach automatically identifies and prunes unimportant channels, resulting in more compact models with reduced size, memory usage, and computational operations, while maintaining or even improving accuracy. This learning scheme aims to reduce the network size without the need for specialized software or hardware accelerators. There are still more areas to explore, such as sparsity techniques (*e.g.*, Pre-Training, SET) and pruning approaches (*e.g.*, global or layer-wise), which offer valuable opportunities for further research and development.

Furthermore, Tmamna *et al.* [41] review various pruning strategies aimed at improving the energy efficiency of deep neural networks, especially CNNs. Their work highlights network pruning as an effective method for reducing the size and computational complexity of models while minimizing their carbon footprint. By selectively eliminating redundant parameters, pruning techniques significantly accelerate model inference and reduce energy consumption. This survey categorizes pruning approaches into three main types: criteria-based, embedded, and automatic methods. The authors investigate the pruning strategies from different angles, including sparsity stages, model architecture, and datasets. They provide a comprehensive analysis of each strategy's impact. It serves as a valuable foundation for further detailed analysis and exploration.

Existing research, including the works of Mocanu *et al.* [13], Galchonkov *et al.* [14], Kalyanam *et al.* [30], Lee *et al.* [33], and Fan *et al.* [34], has advanced the understanding of how pruning techniques and approaches affect model performance and energy efficiency across various architectures. These studies have introduced innovative pruning techniques that focus on reducing model size while maintaining accuracy, exploring both global and layer-wise pruning approaches.

Our work extends previous research by investigating two pruning techniques, including pre-training (Section 2.1.1) and SET in-training (Section 2.1.2). We examine their performance in both global and layer-wise pruning scenarios, specifically within MLP architectures. By doing so, we aim to understand how these techniques impact energy efficiency and model performance, providing insights into the trade-offs between efficiency and accuracy in MLPs. Additionally, to better generalize our findings, we extend our analysis to CNN models, which are more complex, and are commonly used in image classification tasks.

Moreover, our research addresses the sustainability of pruning techniques, an often overlooked area in earlier studies. Specifically, we explore how both global and layer-wise pruning strategies influence energy consumption during both training and inference, carbon dioxide emissions, and overall resource efficiency. This provides a deeper understanding of the actual benefic effects of pruning on model sustainability.

2 Methodology

In this section, the methodology adopted for the study is presented. Firstly, Sect. 2.1 provides definitions of sparsity, the techniques applied, and the mathematical formulations that describe these sparsity methods. The implementation of each technique in the neural network models and their role in enhancing performance are also outlined. Next, Sec. 2.2 introduces two alternative pruning approaches (or strategies), including global and layer-wise pruning. Sect. 2.3 presents the datasets utilized in the research. Following this, Sect. 2.4 describes the architectures of the models employed in the study. Finally, Sect. 2.5 details the overall experimental design, including the process of collecting experimental data and the configurations applied in our experiments.

2.1 Sparsity techniques

Pruning in artificial neural networks refers to the process of removing portions of the network reduce model complexity and size and, in turn, improve its efficiency. This process is typically categorized into structured and unstructured pruning. In structured pruning, entire groups of weights, neurons, or layers are removed, resulting in a regular sparsity pattern. Techniques such as weight-magnitude pruning and filter pruning remove entire units.

Unstructured pruning, instead, removes individual weights based on their contribution to the model, often determined by magnitude or gradient. For example, magnitude-based pruning discards weights holding the smaller values, considering that these will contribute less to the model's performance. This leads to an irregular sparsity pattern, which can reduce memory usage. Magnitude-based pruning, like other unstructured techniques, reduces the number of active connections or weights within the network. This concept is inspired by biological neural systems, where only a small portion of neurons and connections are active at any time [42, 43]. The primary purpose of pruning is to develop more efficient and smaller networks that use fewer computational resources, consume less energy, whilst still achieving strong performance on specific tasks [44, 45].

For comparison purposes, we benchmark sparse ANN against their dense (unpruned) counterparts, (*i.e.,* ANN with 0% sparsity level), where all weights are retained and no sparsity techniques are applied.

In this study, we employ two pruning techniques: Pre-Training (Sect. 2.1.1), and Sparse Evolutionary Training (SET) (Sect. 2.1.2). These two sparsity techniques encompass possible approaches for pruning the neural network, based on the training stage at which the sparsity techniques are applied.

Algorithm 1 presents the pseudocode of our implementation for the pruning techniques used in this research, namely Pre-Training and SET.

2.1.1 Pre-Training Pruning

A common pruning method applies the L1 technique [46] to remove weak connections immediately after the neural network is initialized but before training begins. By starting with a pruned network, where a fraction of connections are removed based on the desired sparsity level, the number of parameters to be optimized is significantly reduced. This not only simplifies the optimization process but also reduces computational costs and speeds up training, as the model requires fewer resources to achieve effective learning [47]. In the following paragraphs, we provide a detailed explanation of the L1 norm, including its mathematical formulation and an illustrative example.

Algorithm 1 Algorithm for applying the considered Sparsification Techniques with Training Loop

```
Require: Model $M$, dataset $D$, sparsity level $\alpha$, learning rate $\eta$, number of epochs $E$,
    replacement rate $\zeta$
Ensure: Trained and sparsified model $M$
 1: Step 1: Initialize Model Weights
 2: $M$.initialize_weights()
 3: Step 2: Create Initial Model
 4: if pruning_technique == "before" then
 5:     $M \leftarrow$ prune_weights($M, \alpha$)                  ▷ Prune weights before training
 6: else if pruning_technique == "set" then
 7:     $M \leftarrow$ initial_erdos_model($n, \alpha$)    ▷ Create sparse connections with Erdős-Rényi
 8: end if
 9: Step 3: Training Loop
10: for epoch = 1 to $E$ do
11:     for each batch $(X, y)$ in $D$ do
12:         Forward Pass:
13:         $output \leftarrow M$.forward($X$)
14:         $loss \leftarrow$ compute_loss($output, y$)
15:         Backward Pass:
16:         $gradients \leftarrow$ compute_gradients($loss, M$)
17:         $M$.update_weights($gradients, \eta$)
18:     end for
19:     if pruning_technique == "set" then
20:         $M \leftarrow$ replace_weak_connections($M, \zeta$)  ▷ Replace $\zeta$% of weakest connections
21:     end if
22: end for
23: Return $M$
```

The L1 norm [48] is a mathematical technique widely used in machine learning and statistics to enhance feature selection, reduce model complexity, and improve prediction accuracy. It minimizes the sum of the absolute values of selected variables (*e.g.*, the weights of connections in our case), leading to sparser networks where most variables become zero (*i.e.*, the removed connections in our experiment). This method is commonly applied in pruning techniques, signal processing, image recognition, and bioinformatics.

The L1 norm, also known as the *least absolute deviation*, measures the distance between points in a mathematical space. For a vector $x = [x_1, x_2, x_3, ..., x_n]$, the L1 norm is defined as:

$$\|x\|_1 = |x_1| + |x_2| + ... + |x_n| \tag{1}$$

The L1 norm calculates the sum of the absolute values of each element in the vector x. Unlike other norms, such as the L2 norm [49], which squares each element, the L1 norm does not amplify the magnitude of larger values. This characteristic makes it particularly effective for generating sparse solutions, where many elements in the vector become zero.

In machine learning, the unstructured L1 method is frequently used to solve optimization problems by minimizing a *loss value*, which measures the difference between predicted and actual values, and incorporating a regularization term based on the L1 norm. This technique is widely used in Lasso Regression, where the objective is to identify feature coefficients that best predict the output while simultaneously reducing model complexity.

2.1.2 In-training Pruning based on SET - Sparse Evolutionary Training

The *Erdős-Rényi* model [29] is a foundational mathematical framework for generating random graphs, where nodes are connected by edges based on probabilities. This model is widely used to study the structure of real-world networks, such as social or biological systems, by simulating their connectivity patterns. Starting with n nodes, each pair of nodes is connected by an edge with a probability p. Due to its simplicity and adaptability, the *Erdős-Rényi* model is a valuable tool for initializing sparse architectures in neural networks.

The SET method, proposed by Mocanu *et al.* [13], leverages the *Erdős-Rényi* model to initialize a sparse neural network. Unlike other sparsification techniques, such as Random or L1 methods, SET uses this probabilistic model to define the initial connections between neurons. By setting the probability p, SET controls the network's initial connectivity, ensuring a sparse yet effective starting structure.

SET optimizes the network during the training, epoch by epoch. It dynamically updates the network by replacing the 20% of smallest positive and largest negative weights (*i.e.*, weaker connections) with previously pruned connections, this is the proportion that Mocanu *et al.* [13] used in their own implementation of this technique. This dynamic adjustment ensures that the network continuously strengthens the most important connections while keeping the level of sparsity constant. By the end of training, the network evolves into a robust structure with optimized connections, achieving an effective balance between computational efficiency and performance.

Fig 1. The architecture of our two multilayer perceptron (MLP) models including Small-Scale and Medium-Scale, and the pruning strategies we applied over the models. The scissor icons between layers indicate the locations where pruning is applied. The numbers in the layers indicate the number of neurons in the layer.

2.2 Pruning approaches

Herein, we mostly focus on comparing two pruning approaches for multilayer perceptron (MLP) models: *global* and *layer-wise* pruning. The pruning techniques (Sect. 2.1) are used to reduce the number of connections in a neural network, which can improve efficiency and lower resource usage during inference.

Global unstructured pruning involves removing weights from the entire network architecture based on their importance, regardless of which layer they belong to. In this approach, a fixed percentage of the weakest connections across the whole network are pruned, regardless of the layer they are in. While this method is simple, it might not always lead to optimal performance, as it does not account for the specific contributions of each layer to the overall network.

On the other hand, the layer-wise pruning approach we propose herein targets specific layers of the network. Our intuition is that the input layer in MLPs holds important features from the input data, and removing connections from this layer can reduce the model's performance. For the same reason, keeping all connections in the last layer (*i.e.*, from the last input layer to the output) is important for maintaining the necessary connections to accurately classify the inputs.

Thus, as shown in Fig 1 for the MLP architectures, we prune only the hidden layers leaving the input and output layers unchanged.

Similarly, in Convolutional Neural Network (CNN) models, we apply the same sparsity techniques to the classification part of the network, as well as to all connections between the last convolutional layers and the first classification layer. These connections represent the extracted features from the convolutional layers, and they serve as the input to the classification part of the network. However, we do not apply any sparsity techniques to the feature extraction part of the network (*i.e.*, the convolutional layers).

In our experiments, we compare the performance of two neural network models (*i.e.,* MLPs and CNNs) under both pruning approaches (*i.e.*, Global and Layer-wise), focusing on key metrics such as test accuracy, inference time, energy efficiency during both inference and training phases, and memory usage during training. By examining these factors, we aim to understand how global and layer-wise pruning affect models performance, and determine which approach is more effective and efficient in terms of both energy use and accuracy.

2.3 Datasets

In this section, we describe the datasets used in our study. The following subsections provide details on each dataset, namely MNIST in Sect. 2.3.1, FashionMNIST in Sect. 2.3.2, EMNIST in Sect. 2.3.3, CIFAR-10 in Sect. 2.3.4, and OctMNIST in Sect. 2.3.5. With the exception of OctMNIST, all these datasets are natively available through the PyTorch builtin datasets module (official documentation at `https://docs.pytorch.org/vision/stable/datasets.html`. OctMNIST is provided as a standalone Python library (accessible via `https://pypi.org/project/medmnist`). We utilized these specific implementations throughout our research experiments. For the experiments involving CNNs (Sect. 3.5), we only use the last three datasets, CIFAR-10, EMNIST, and OctMNIST, which are the most complex ones.

2.3.1 MNIST

The MNIST dataset is a widely used benchmark in machine learning, consisting of 28×28 pixel grayscale images of handwritten digits (0–9). It includes 60,000 images, making it a standard choice for classification tasks and sparsity techniques. MNIST's simplicity, small size, and well-defined structure allow for efficient experimentation and comparison of results.

2.3.2 FashionMNIST

Similarly to MNIST, FashionMNIST [50] is often used as a benchmark for machine learning; however, it is used for slightly more complex and realistic classification problems, since the images are not just digits, but clothing items. It consists of 60,000 grayscale images of 28×28 pixels, and includes 10 classes, each representing a different type of clothing. These include items such as t-shirts, trousers, and sneakers.

In our study, we employ FashionMNIST to evaluate how well sparsity techniques perform when applied to a slightly more complex dataset.

2.3.3 EMNIST

The EMNIST dataset [51] is an extension of the MNIST dataset (*i.e.*, it is more than 11 times larger than the MNIST dataset) and is designed for more complex handwriting recognition tasks. It consists of handwritten characters, including digits and letters, making it suitable for multi-class classification problems.

It is worth to mention that this dataset has several different splits including ByClass, ByMerge, Balanced, Letters, Digits and MNIST. One of the most commonly used ones is the *ByClass* split, which is composed by a total of 62 classes. These classes cover not only digits (*i.e.* 10 classes from 0-9) but also uppercase and lowercase English letters (*i.e.* 26 classes each from A-Z and from a-z, respectively).

Similarly to both MNIST and FashionMNIST, the dataset is composed by grayscale images of 28×28 pixels. The *ByClass* subset, contains 814,255 samples from 62 unbalanced classes. The increased number of classes in EMNIST, which includes both handwritten digits and letters, makes it a suitable benchmark for evaluating the performance of sparsity techniques on more challenging multi-class classification tasks.

2.3.4 CIFAR-10

CIFAR-10 [52–54] is a well-known dataset in the field of machine learning, used for advanced image classification tasks. It consists of 50,000 colored 32×32 pixel images categorized into 10 different classes (*i.e.*, Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, and Truck classes). The main challenge of CIFAR-10, compared to datasets like EMNIST, MNIST, and FashionMNIST, lies in its color images, which adds complexity due to the varied backgrounds and textures. In our experiments, we use CIFAR-10 to assess the effectiveness of sparsity techniques on color images with more intricate visual features.

2.3.5 OctMNIST

The OctMNIST dataset is part of the MedMNIST [55, 56] collection, a standardized set of biomedical image datasets for machine learning.

MedMNIST is a comprehensive biomedical image dataset designed to support machine learning research in the medical domain. It comprises 18 distinct datasets, including 12 datasets for 2D images and 6 datasets for 3D volumes, each formatted to facilitate seamless integration with deep learning models. These datasets are specifically curated for a variety of medical applications, such as disease classification, tissue segmentation, and anatomical identification. The MedMNIST Python library provides four standardized image size configurations (28×28, 64×64, 128×128, and 224×224 pixels).

We focused our analysis on 28×28 and 224×224 resolutions for optical coherence tomography (OCT) retina images only. This choice allows a fairer comparison across all datasets used in this study, while also enabling an investigation of the model's performance when dealing with higher-resolution samples.

These images show four retinal conditions which are converted to four neurons in the output layer of the network: healthy tissue, diabetic macular edema, drusen, and choroidal neovascularization. The dataset's balance of medical relevance and manageable size makes it practical for controlled experiments. The dataset is divided into three subsets, namely training (89%), validation ($\sim$ 10%), and test (1%) as per the official partitioning scheme provided in the MedMNIST Python library.

2.4 Models architecture

The present study primarily focuses on multilayer perceptron (MLP) models, while also extending the analysis to Convolutional Neural Networks (CNNs) to examine how the observed trends generalize to more complex models.

MLPs, with their simple and fully-connected structure —where each neuron in a layer is directly linked to every neuron in the following layer— serve as a fundamental baseline for understanding the effects of sparsity techniques. This simplicity allows us to isolate and analyze the impact of pruning in a controlled setting, without the additional complexities introduced by advanced architectures such as CNNs or ResNet-based models, which are well known for their superior performance in image classification tasks.

For our experiments, we employ two MLP model sizes, namely a small-scale model (Sect. 2.4.1) and a medium-scale one (Sect. 2.4.2). These provide a range of complexity levels that allow us to evaluate sparsity techniques under different resource conditions. While one might expect the inclusion of larger-scale models, our focus on smaller MLP architectures is intentional, as these allow for a clearer observation of the relationships between pruning, energy consumption, and efficiency in constrained environments.

To complement this analysis, we also include the VGG-16 architecture (Sect. 2.4.3) as a CNN case study, enabling the evaluation of sparsity techniques in a more complex, convolution-based setting. This comparison between MLPs and CNNs provides a broader understanding as to how diverse pruning techniques and approaches perform across different network types, from simple fully-connected architectures to deeper convolutional models.

2.4.1 Small-scale architecture

We begin our analysis with a small-scale MLP model, designed to accommodate input data with a dimensionality of $28 * 28 = 784$, which aligns with the size of images in datasets such as MNIST. The input layer of the model consists of 784 neurons, ensuring that each pixel in the input image is represented by a single neuron. The network also includes two hidden layers, each containing 16 neurons, and an output layer with 10 neurons, matching the number of classes in typical classification tasks like MNIST.

In total, this 4-layer architecture comprises 826 neurons. When fully-connected, the model has 12,960 connections, which is the minimum number of connections required to achieve an accuracy of up to 90% in our experimental setup. By decreasing the size of the network the final accuracy of the test phase will drop. This small-scale architecture is intentionally designed to balance simplicity and computational efficiency while providing a meaningful framework for evaluating sparsity techniques. By focusing on this compact network, we can effectively study the impact of sparsification on model performance, leveraging input sizes and structures similar to those found in widely used datasets like MNIST.

2.4.2 Medium-scale architecture

To enable a comparison, we also employ a medium-scale MLP model. This model is structured with 4,000 neurons in the first hidden layer, 1,000 neurons in the second hidden layer, and 4,000 neurons in the third hidden layer.

As an example, using the MNIST dataset, we calculate the total number of neurons and connections in the medium-scale MLP model. This 5-layer architecture, which includes input, 4000-1000-4000 hidden layers, and output layers, comprises 9,794 neurons in total. When fully-connected, the model has 11,176,000 connections, which is the minimum required to achieve the desired performance on the MNIST dataset. These values provide a clear quantitative foundation for evaluating the scalability and computational requirements of the proposed sparsity techniques in a high-dimensional setting.

2.4.3 VGG-16 architecture

In addition to the MLP models, we also use the VGG-16 architecture to extend our study to CNNs. We included this model just as an example and for the sake of completeness to show how our approach might be obscured in interpretation when more complex architecture are used.

The original VGG-16 consists of 16 layers, including 13 convolutional layers and 3 fully-connected layers. The convolutional layers are responsible for feature extraction, while the fully-connected layers perform the classification.

Fig 2. The VGG-16 architecture used for our experiments, with layers where sparsity techniques were applied highlighted by a red border and labeled as "pruned component".

In our experiments, we extend the original VGG-16 by adding two additional fully-connected layers to the classification part of the network. This modification increases the complexity of the classification process and makes the pruning effects more observable when applying different sparsity techniques. We selected VGG-16 for two main reasons. First, it provides a balanced level of complexity—large enough to represent real-world CNN behavior, yet not too large to make the experiments impractical. Second, it allows us to compare our findings from the MLP models with a CNN architecture under similar sparsity techniques. By applying both global and layer-wise pruning to the classification part of the network, we can analyze whether the same trends observed in MLPs also appear in more complex CNNs. This helps us evaluate the generality of our observations across different model types. The overall structure of the modified *VGG-16* architecture, including the additional fully-connected layers and the pruned sections in each approach, is illustrated in Fig 2.

2.5 Experimental design

This research investigates the impact of applying sparsification on hidden layers in MLP models. Two pruning approaches are considered for this purpose. In the first one, the pruning is applied globally, meaning all layers are pruned to the same sparsity level including input-layer, hidden-layers, and output-layer. In the second one, the same sparsity level is applied only to hidden layers in the MLP model architecture. To ensure the reliability of our results and observations, we expand our work by exploring CNN models as well. Both pruning approaches are applied, but only onto the classification part of the CNN, as this is the one sharing the most similarity with MLPs. When the global approach is performed, all layers in the classification part are pruned; whereas in the layer-wise pruning we retain all connections in the output layer and those connecting the feature extraction part to the first layer of the classification part.

The pruning approach proposed in the present study (*i.e.*, the *layer-wise* pruning) is grounded on two key hypotheses. First, the initial layer of the MLP networks often contains significant and valuable information that can substantially influence the model's performance during training. Second, the final layer makes the model able to select the most appropriate output class, as each neuron in this layer is directly connected to a single output class. Thus, we restrict the application of sparsity techniques (namely, SET and Pre-training) exclusively to the hidden layers, preserving the integrity of the input and output layers. This approach ensures that the model retains its capacity to process critical information at the beginning and make accurate predictions at the end, while still achieving the benefits of sparsity in the intermediate layers.

To ensure a fair comparison across all combinations of datasets (Sect. 2.3), and models (Sect. 2.4), we employed a constant training configuration. In both, the dense and the sparse version of the model, the same hyperparameters were applied to keep the evaluation consistent and directly comparable. Each model was trained with a batch size of 4, a learning rate of 1×10^{-3}, and a test batch size of 4. The training process was run for 60 epochs to ensure stable convergence and enable a fair comparison between dense and sparse models.

For the CNN model, the batch size was increased to 64, and the learning rate was set to 1×10^{-4}. We initially trained the VGG-16 architecture for 60 epochs, the same as for the MLP models. However, training the CNNs for 60 epochs led to overfitting across all datasets. To address this, we reduced the training rounds to 45 epochs to prevent overfitting.

The results discussed in Sect. 3 are based on the average values obtained from ten independent runs of the experiments.

In terms of metric monitoring and evaluation of the performance of the model in all experiments, we used a total of six metrics, namely *test accuracy*, *inference time*, *total inference energy*, *training energy*, *CO_2 emissions*, *power usage*, and *maximum peak memory*. These metrics provide a comprehensive understanding of the model's behavior, efficiency, and resource usage.

To measure energy consumption during model training, we employed the *Zeus* library [57–59], a tool designed for GPU energy monitoring and developed as a Python library. This library uses the NVIDIA Management Library (NVML) to collect power usage directly from the GPU hardware, ensuring reliable data collection. As shown in prior research [59], Zeus implements just-in-time (JIT) profiling, enabling real-time energy measurement with minimal computational overhead. While effective for single-GPU setups, Zeus has limitations in high-performance computing (HPC) environments where GPUs are partitioned for parallel jobs, as it cannot isolate energy usage for individual virtual GPU instances.

To better understand this limitation, we conducted experiments on a dedicated machine, the *Lenovo Legion 5* with an *NVIDIA GeForce RTX 3060* GPU, *16GB* RAM, and an *Intel Core i7-11800H* CPU. This setup allowed for more accurate energy measurement, as the entire GPU was used. We found that training the model on the *Lenovo Legion 5* used about one-third of the training energy compared to the HPC system. This issue arises from the limitations of the HPC system, where resources are often shared.

In real-world settings, especially with low-resource devices, one typically uses all available hardware for a single training task, which helps optimize energy efficiency. The method used for measuring these metrics is consistent across both hardware (*i.e.*, , and GPU) configurations.

All of our training experiments include three standard phases, namely, training, validation, and testing. Algorithm 2 shows how we measure energy for each epoch

Algorithm 2 Energy metric measurement in training phase

```
1: training_energy ← 0
2: for epoch ∈ {1, . . . , args.epochs} do
3:     monitor.begin_window("epoch")                      ▷ Start energy tracking for epoch
4:     All Training Functions
5:     for (inputs, labels) ∈ train_data do
6:         Calculate the loss value
7:     end for
8:     measurement ← monitor.end_window("epoch")          ▷ End energy tracking
9:     training_energy ← training_energy + epoch_energy
10: end for
```

separately. This ensures that no extra operations in the training phase interfere with our measurements. In addition, all code used in this study is publicly available on GitHub (available at `https://github.com/amirzenoozi/layer-wise-pruning-in-mlps`).

We recognize that using an MLP with a fixed configuration for training is a limiting factor and does not provide the best performance for every dataset or pruning scenario. For example, when working on a more complex dataset, we expect lower performance due to a more complex task compared to classifying digits or clothes. Furthermore, using CNNs leads to a higher accuracy [19, 56, 60, 61] for real-world problems because of the feature extraction part of the models, which does not exist in the MLP architectures. However, the model's behavior is consistent and can be observed in all experiments, regardless of the dataset. This allows us to focus on how pruning affects the model's overall sustainability, not just its performance on a specific dataset.

3 Results

In this section, the results achieved varying (i) sparsity level (*i.e.*, 50% and 80%, (ii) pruning technique (*i.e.*, Pre-Training and SET), (iii) and pruning approach (*i.e.*, global *vs* layer-wise pruning) are presented and discussed. Each element of this test-matrix is benchmarked against the dense models (*i.e.*,, fully-connected ANN with 0% sparsity).

First, in Sect. 3.1, we investigate the model's accuracy in the test phase, its average inference time, and the total inference energy, highlighting the best trade-off combinations.

Next, in Sections 3.2 and 3.3 the energy consumptions and carbon dioxide emissions estimations incurred during the training phase are investigated, respectively. There, we also examine the power consumption and savings associated with the chosen pruning approach. In Sect. 3.4, we compare the peak memory usage across all experiments, to evaluate the impact of the different model settings on memory efficiency.

Additionally, Sect. 3.5 provides a detailed discussion of our observations on CNN networks, where we examine model performance, analyze the trade-offs between the training and test phases, and present a comparison of the energy consumption during training.

Finally, in Sect. 3.6, we discuss the impact of the layer-wise approach on domain-specific datasets using both MLPs and CNNs model architectures, which could lead to future exploration of this method in real-world scenarios.

3.1 Inference Efficiency

The main focus of our research revolves around model sustainability (especially at inference time) and how pruning techniques affect its performance across different settings. Herein, we focus on the trade-offs concerning test accuracy, inference time and energy usage for the different models.

Table 1 shows the results for the MNIST and FashionMNIST dataset. In it, the model's performance across three key metrics, namely test accuracy (ACC), average inference time (AIT), and total inference energy (TIE) are summarised. We report those outcomes for both model sizes (*i.e.*, small and medium scale), sparsity levels (*i.e.*, 50% and 80%), approaches (*i.e.*, global and layer-wise), and pruning techniques (*i.e.*, pre-training and SET). Note that sparsity level refers to the fraction of pruned neural connections. The outcomes are reported as relative percentual difference to the benchmark (*i.e.,* classic training with the initial fully-connected dense network, with no sparsity involved).

To evaluate the average inference time and total inference energy we ran the experiments on the CPU rather than the GPU. This approach allowed for a fair, consistent and comparable measurement across all experiments, as it removed any performance bias that might arise from using different hardware configurations, as is the case of high permance computing clusters.

The table shows that sparsity techniques can be efficiently applied to the hidden layers of our MLP model while maintaining competitive performance. When compared to the dense version of the model using the same architecture and same datasets, the sparsified models show only minimal degradation in accuracy. Specifically, both Pre-Training and SET approaches maintain accuracy within 2% of the dense baseline at 50% and 80% sparsity levels for the layer-wise approach. This suggests that significant network compression can be achieved without substantially compromising model performance.

The pruning approach choice plays an important role in terms of improvements, especially for inference efficiency. Layer-wise pruning consistently matches or outperforms global pruning in most configurations across all metrics.

In detail, when the MNIST dataset is considered, the best performance is observed in the small-scale MLP model, with 50% sparsity, where the inference time improved by approximately 33%, while the accuracy dropped by only 0.49% in the layer-wise scenario for both SET and Pre-Training techniques.

Similarly, for the FashionMNIST dataset, the best results were achieved with the medium-scale MLP model at 50% sparsity in the layer-wise scenario, where the average inference time improved by around 23%, with only a 0.65% decrease in accuracy.

These results indicate that layer-wise pruning not only preserves model accuracy but also offers significant computational advantages during inference.

The standard deviations across multiple runs are small, with variations within ±0.5% for accuracy (ACC), ±3.2% for average inference time (AIT), and ±7.2% for total inference energy (TIE). These fluctuations in inference time and energy consumption can be attributed to minor variations in hardware load and resource scheduling during the execution of the experiments. Since the inference phase was performed on a standard CPU setup rather than a fully isolated environment, background processes and system-level energy management will have slightly influenced the measurements.

While the results presented in Table 1 provide detailed numerical comparisons, they do not fully capture the trade-offs between performance and efficiency metrics including for instance average inference time and total inference energy. To address this limitation and provide a more comprehensive understanding, we have generated additional diagrams (see Fig 3) to visualize these trade-offs among different pruning techniques and sparsity levels for the combination of each dataset and model.

Table 1. Model performances for the MNIST and FashionMNIST datasets. Test accuracy (ACC, %), average inference time (AIT, s), and total inference energy consumption (TIE, J), expressed as relative percentual differences to benchmark values. The rows with a different background represent the best trade-off, and bold cells highlight the best results for each metric.

Model	Scenario			MNIST			Fashion MNIST		
	Sparsity	Approach	Technique	ACC	AIT	TIE	ACC	AIT	TIE
Small-scale MLP	Benchmark (0% Sparsity)			96.55%	1.43e-4s	725.61J	87.15%	1.64e-4s	892.00J
	50%	Global	Pre-Training	-1.67	-13.98	-3.70	-1.4	-7.31	-17.88
			SET	-0.9	-14.68	-10.34	-1.06	-3.65	-20.44
		Layer-wise	Pre-Training	**-0.49**	**-33.56**	-40.40	**-0.2**	**-10.36**	-23.38
			SET	**-0.49**	**-33.56**	-44.57	-1.31	-9.14	**-23.95**
	80%	Global	Pre-Training	-6.4	-10.48	-45.69	-3.38	-7.31	-20.40
			SET	-3.28	-27.97	-47.19	-2.81	-7.92	-18.34
		Layer-wise	Pre-Training	-1.8	-27.27	**-56.45**	-1.20	-9.14	-22.09
			SET	-1.19	-20.27	-55.56	-1.49	-9.75	-22.84
Medium-scale MLP	Benchmark (0% Sparsity)			98.15%	2.741e-3s	898.56J	83.65%	2.91e-4s	1134.02J
	50%	Global	Pre-Training	-1.40	-7.29	-30.28	-0.99	-12.37	-19.55
			SET	-0.92	-8.39	-23.99	-0.77	-13.05	-17.77
		Layer-wise	Pre-Training	-0.92	-19.33	-23.88	-0.65	**-23.02**	**-27.35**
			SET	**-0.91**	-12.40	-22.41	**-0.53**	-19.24	-24.47
	80%	Global	Pre-Training	-5.55	-12.03	-19.89	-2.98	-16.49	-15.62
			SET	-5.21	-10.58	-23.19	-2.60	-15.46	-19.02
		Layer-wise	Pre-Training	-2.8	**-21.52**	-37.09	-1.63	-20.61	-22.71
			SET	-2.17	-19.70	**-43.68**	-1.27	-19.58	-22.23

Since our study evaluates multiple metrics, including accuracy, average inference time, and total inference energy, it becomes necessary to reduce the dimensionality of the comparison while keeping the most relevant information. For this purpose, we introduce the *Efficiency Index*, which combines AIT and TIE into a single metric. Both of these metrics reflect the efficiency of the model, where lower values indicate better performance in terms of speed and energy usage. The Efficiency Index is calculated using the following equation:

$$\text{Efficiency Index} = 1 - \left(\frac{1}{2}\left(\frac{\text{AIT}}{\text{AIT}_{\text{max}}} + \frac{\text{TIE}}{\text{TIE}_{\text{max}}}\right)\right) \quad (2)$$

where, AITmax and TIEmax represent the maximum values of AIT and TIE across all experiments, respectively. This normalization ensures that both metrics are on the same scale, with values ranging between 0 and 1. A higher Efficiency Index indicates a more efficient model that performs faster and consumes less energy.

Fig 3 illustrates the trade-off between model accuracy (%) and Efficiency Index for

Fig 3. Comparison of accuracy–efficiency trade-offs across two models and two datasets. Each point represents one pruning experiment. Labels follow the format *Model_Sparsity_PruningApproach_Technique*, where (S) denotes Small-Scale, (M) denotes Medium-Scale, (G) denotes Global pruning approach, (L) denotes Layer-wise pruning approach, (PT) represents the Pre-Training technique, and (SET) refers to the Sparse Evolutionary Training technique. Red dots marks the global pruning approach, while blue dots marks the layer-wise approach, and the green "×" symbol marks the benchmark dense model (*i.e.*, 0% sparsity).

different pruning configurations. From these visualizations, it becomes clear that models trained with the layer-wise approach achieve a better trade-off, showing higher accuracy while maintaining lower energy and time consumption compared to the global pruning method. By analyzing each pair of experiments with the same sparsity level, model scale, and pruning technique, we can clearly see that the layer-wise approach consistently outperforms the global pruning method under identical training conditions.

The comparable test accuracy between sparse and dense models, combined with the improved inference times from layer-wise pruning, suggests that our sparsity approaches provide practical benefits for efficient model deployment.

One main reason for this behavior in the layer-wise approach is that keeping all the connections in the first and the last layer (*i.e.*, input features and output classes, respectively) helps the model to correctly match the input with the appropriate output class. This ensures that the model can better map the features from the input to the final decision. Indeed, by keeping all the connections in the first layer, a better feature extraction from the input neurons is possible. By preserving these connections, the model can capture and process raw input data more effectively, which is crucial for accurate predictions. The consistent performance across two sparsity levels further demonstrates the robustness of these techniques.

3.2 Training efficiency

This section evaluates energy consumption incurred for training, across the experimental configurations, on the MNIST and on the FashionMNIST datasets. By comparing the energy consumption of all considered techniques and scenarios to that of the corresponding dense version, we observe an increase in the metric for all pruning techniques, which was to be expected. This behavior is due to an increased number of calculations undertaken during the training phase, incurred to find the weakest connections to be eliminated.

Fig 4 shows the energy usage during the training phase for the small and medium scales models, on the two datasets under scrutiny. The charts use zoomed *y-axes* to highlight performance differences, with ranges starting from 150,000J to 400,000J for all the plots. Across all experiments, layer-wise pruning consistently showcased lower energy requirements compared to global pruning. Each bar represents the average value obtained from the experimental repetitions, while the range markers indicate the minimum and maximum values observed across all runs, showing the range of results. Moreover, changing the pruning approach from global to layer-wise shows the similar impact on saving the energy for both small- and medium-scale models.

These results were collected using the GPU over an HPC system for the training phase, while the inference time and energy, discussed in Sect. 3.1, were measured using the CPU to demonstrate the model's performance and applicability in low-resource situations.

Fig 4. Comparison of training-phase energy profiles across two models and two datasets. Each sub-figure presents five experiments including the two pruning techniques (*i.e.*, Pre-Training and SET) and two sparsity levels (*i.e.*, 50%, and 80%) and dense version of the model (*i.e.*, 0% sparsity). In each figure, we zoom the y-axis to show the differences between the global pruning approach that represented by blue bars and the layer-wise scenario that represented by orange bars.

3.3 Environmental impact

By increasing the usage of complex and widespread models, the environmental impact of their development and deployment is becoming increasingly crucial. While performance metrics such as accuracy and efficiency are often the main focus, the carbon footprint associated with training and using machine learning models is frequently overlooked. Measuring CO_2 emissions directly through libraries can be difficult, but there are datasets available [62, 63] that track CO_2 emissions based on energy consumption. By using these datasets, we can estimate the emissions generated by such processes. According to the CO_2 intensity dataset by Peng [62], the average value of this metric between 2000 and 2017, measured in gCO_2 per kWh, for the whole world is 527.93, and we use this value to measure the CO_2 emissions during the training phase in this research.

Using the energy consumption metrics from Sect. 3.2 and the average carbon dioxide intensity in the whole world, we estimate the CO_2 emissions of the model training phase. In all considered scenarios we achieve a reduction in CO_2 emissions varying between 6.48 and 10.48 gCO_2 when using the layer-wise approach compared to the global pruning approach.

In recent years, the growing demand for training complex AI models has raised concerns not only about their computational cost but also about their environmental impact. Power consumption during model training is an important metric that directly affects the overall sustainability of machine learning systems. By converting the energy usage from Joules (J) to Watt-hours (wH), we achieve a better understanding of the electricity consumption required to train models. Training a model typically requires more energy and time compared to the inference phase, making it a considerable factor in energy usage. Moreover, as the number of AI models being trained continuously increases, the cumulative impact on power consumption also increases. This metric is especially important in the context of medium-scale AI research and production environments, where training models can result in significant energy consumption. This metric is able to potentially play a key role in future research focused on reducing the environmental costs of machine learning and ensuring that AI development contributes positively to both technological progress and sustainability.

The application of the layer-wise pruning approach led to a reduction in power usage in all of our experiments compared to the global approach. Specifically, in these experiments, the power consumption was reduced by amounts ranging from 1.26wH to 18.28wH, highlighting an improvement in power efficiency for the majority of the cases, supporting the effectiveness of layer-wise approach in optimizing energy usage during model training compared to the global approach.

3.4 Peak memory utilization during training

Understanding peak memory allocation during training is critical for optimizing resource usage and avoiding hardware limitations. This metric captures the maximum memory needed by the model, data, and intermediate computations at any point during the training phase. High memory requests can lead to out-of-memory errors or forced batch-size reductions, directly impacting training efficiency. In this section, we analyze peak memory usage for our small- and medium-scale models, correlating it with architectural choices (*e.g.*, layer width, batch size) and different datasets.

Fig 5. Peak memory allocation during training all models and datasets. The results highlight how model architecture and dataset complexity influence memory demands.

All four charts in Figs. 5 use zoomed *y-axes* to highlight performance differences. The *y-axes* in Figs. 5a and 5c starts from 17 MB, and in Figs. 5b, and 5d starts from 180 MB.

An important observation highlighted in Fig 5 is that the layer-wise approach always peaks less memory compared to the global approach. In addition, the observed increase in peak memory usage with the SET technique in the medium-scale model results from the increased complexity of the model, with three hidden layers and 4000-1000-4000 neurons. The larger model size, combined with SET's dynamic pruning approach, might lead to requiring more memory to store and manage weights.

3.5 Discussion on Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have revolutionized the field of deep learning, particularly in applications such as image classification and object detection [20–24]. In the context of multi-class classification, CNNs are widely used to categorize data into multiple classes, requiring the model to learn complex representations of each class. However, as the depth and complexity of CNNs increase, so does the computational cost, which poses challenges for deployment in resource-constrained environments. To address this, pruning has emerged as an effective technique to reduce model size and improve efficiency without sacrificing significant performance.

Moreover, selecting the right pruning method to keep the balance between model efficiency and performance for CNNs is crucial due to the unique structure of these networks. In addition to the fully-connected layers, which are responsible for classification tasks, CNNs also contain convolutional layers that handle feature extraction. These convolutional layers are highly sensitive to pruning techniques [25], meaning that removing or altering them can significantly affect the model's ability to extract important features.

Designing experiments for MLP architectures is relatively straightforward (Sect. 2.2), as these networks are simpler and have a direct connection between the input and output layers. However, for CNNs, the process is more complex due to their dual structure, which consists of a feature extraction part (*i.e.*, the convolutional layers) and a classification part (*i.e.*, the fully-connected layers).

In our previous work [25], we investigated the performance of CNN models with the same pruning techniques of Sect. 2.1, which were are applied over the convolutional layers globally. For this current study, we aimed to maintain consistency between MLP-based experiments and CNN-based ones. To do so, we decided to apply both layer-wise and global pruning exclusively to the classification part of the network. Specifically, we preserved all connections between the feature extraction part and the first layer of the classification section, as well as all connections in the last classification layer. This experimental setup mirrors the approach used in the MLP model experiments, allowing for a fair comparison. By doing this, we can directly compare the results from the MLP networks to the more complex CNN networks.

To make our experiments more reflective of real-world scenarios, we decided to run the CNN experiments on more complex and varied datasets. We used CIFAR-10 2.3.4 and EMNIST 2.3.3. In this way, we introduce a more practical and diverse challenge to evaluate CNN pruning techniques. For our CNN experiments, we chose the VGG-16 architecture for several reasons (as per Sect. 2.4.3). Furthermore, VGG-16 is suitable for use in low-resource environments compared to other larger models. It is also not too small or too large, allowing us to investigate our research questions effectively without overwhelming computational resources [64–66].

Table 2. Model performances for the CIFAR-10 and EMNIST datasets, evaluated using the VGG-16 architecture. Test accuracy (ACC, %), average inference time (AIT, s), and total inference energy consumption (TIE, J), expressed as relative percentual differences to benchmark values. The rows with a different background represent the best trade-off, and bold cells highlight the best results for each metric.

Model	Scenario			CIFAR-10			EMNIST		
	Sparsity	Approach	Technique	ACC	AIT	TIE	ACC	AIT	TIE
VGG-16	Benchmark (0% Sparsity)			91.36%	1.26e-3s	77.54J	86.25%	1.68e-3s	940.56
	50%	Global	Pre-Training	+0.04	-1.59	-31.42	+0.08	-18.67	-1.94
			SET	+0.11	-1.50	-23.78	+0.16	-3.67	-5.60
		Layer-wise	Pre-Training	+0.09	-1.23	-31.39	+0.17	**-19.33**	-9.69
			SET	-0.02	-1.59	-30.04	+0.12	-8.23	-14.16
	80%	Global	Pre-Training	+0.02	-3.17	-35.00	+0.23	-11.67	-10.49
			SET	**+0.22**	-3.76	-24.76	**+0.27**	-10.33	-14.84
		Layer-wise	Pre-Training	+0.20	-3.87	**-44.57**	+0.18	-9.50	-12.09
			SET	-0.13	**-4.24**	-31.51	+0.12	-12.34	**-16.68**

In terms of model performance, Table 2 refers to experiments conducted on the VGG-16 architecture and two datasets, namely CIFAR-10 and EMNIST. Although the accuracy of the model does not change significantly - and we observe similar performance compared to the dense version - we can see that the sparse networks require less energy and time during the test phase.

Furthermore, since the fluctuation in accuracy is observed to be less than 1 percent in our experiments, we assume that these variations are not directly related to the sparsification process. Such small differences in accuracy are expected, as achieving identical results across multiple training runs is generally difficult. When the model is trained several times, the outcomes tend to be very close, though not exactly the same.

The standard deviations across several runs are minimal, with accuracy (ACC) varying within ±0.3%, average inference time (AIT) within ±3.7%, and total inference energy (TIE) within ±5.2%. Besides the numerical comparison, the trade-off between accuracy and efficiency is further illustrated in Fig 6, which provides a visual representation of how the pruning configurations perform relative to the benchmark. As observed, the layer-wise configurations are positioned more favorably than the global ones, achieving higher accuracy with higher Efficiency Index values. This indicates that layer-wise pruning delivers a better balance between accuracy and efficiency, confirming its advantage over global pruning in terms of both performance and energy effectiveness.

The energy profiles during the training phase are shown in Fig 7, which illustrates the energy consumption of training VGG-16 models on two datasets: CIFAR-10 and EMNIST. The y-axis ranges for Fig 7a and Fig 7b differ because using the same range would make it difficult to distinguish the variations in energy consumption between the experiments for CIFAR-10.

The main observation from these figures is that training the VGG-16 on the

Fig 6. Comparison of accuracy–efficiency trade-offs across two datasets on VGG-16. Each point represents one pruning experiment. Labels follow the format *Model_Sparsity_PruningApproach_Technique*, where (S) denotes Small-Scale, (M) denotes Medium-Scale, (G) denotes Global pruning approach, (L) denotes Layer-wise pruning approach, (PT) represents the Pre-Training technique, and (SET) refers to the Sparse Evolutionary Training technique. Red dots marks the global pruning approach, while blue dots marks the layer-wise approach, and the green "×" symbol marks the benchmark dense model (*i.e.*, 0% sparsity).

EMNIST dataset requires more energy compared to CIFAR-10. Based on our experiments, this difference is primarily due to the higher complexity of the EMNIST dataset, which contains 62 classes, compared to CIFAR-10's 10 classes. Additionally, the samples in EMNIST are more similar to each other, which demands more energy to extract features for classification. It is also important to note that the majority of the energy is consumed by the feature extraction part of the model for both dataset, which is a critical component of any convolutional network. Furthermore, applying layer-wise sparsity approach successfully reduced energy consumption during the training phase across all experiments compare to global pruning approach.

This finding is consistent with our results from the MLP models discussed in Sect. 3.2. For CNNs, similar to the results seen with MLPs, using the layer-wise sparsity approach led to reduced energy consumption compared to the global pruning approach. Additionally, this approach resulted in faster model performance during the test phase.

Fig 7. Comparison of training-phase energy profiles across two datasets on VGG-16 architecture. Each sub-figure presents five experiments including the two pruning techniques (Pre-Training and SET) and two sparsity levels (50%, and 80%) and dense verison of the model (*i.e.*, benchmark). In each figure, we zoom the y-axis to show the differences between the global pruning approach that represented by blue bars and the layer-wise approach that represented by orange bars.

3.6 Discussion on domain-specific datasets

In this section, we focus on a series of experiments conducted using the OctMNIST dataset (Sect. 2.3.5), a medical image dataset chosen to evaluate model behavior on more complex, domain-specific data. This kind of datasets presented additional challenges for our experimental setup due to their higher resolution and the complexity of the images.

In terms of training parameters, we initially trained the OctMNIST dataset (28×28 pixels and 224×224 pixels) using the same parameters as for MNIST or FashionMNIST. However, this configuration showed poor performance, with the dense model failing to exceed 50% accuracy across all experiments. To address this, we adjusted the hyperparameters based on OctMNIST's greater complexity: increasing the batch size from 4 to 256 and reducing the LR from *1e-3* to *1e-5*. These modifications improved model convergence and performance, achieving a peak accuracy of 62%. Although not a satisfying result, this was expected given the complexity of the medical dataset in comparison to that of the chosen architectures. While increasing the model scale from small to medium resulted in improvements across all performance metrics, namely accuracy, average inference time, and inference energy, the performance of the dense version of the model remained low, and applying pruning techniques, whether global or layer-wise, led to a significant decrease in performance. For example, in some experiments, we observed a nearly 25% drop in accuracy after applying the pruning techniques.

In terms of training energy, changing the dataset to OctMNIST shows contrasting results compared to MNIST and FashionMNIST. We observed that the small-scale model displays higher energy demands when processing more complex datasets. Furthermore, as the input sample resolution increased from 28×28 to 224×224, the difference between the global pruning scenario and the layer-wise scenario became more noticeable.

The higher energy consumption in layer-wise pruning can be explained by two main factors. First, there is extra overhead from computing gradients because layer-wise pruning updates each layer one by one. During backpropagation, correction signals are calculated separately for each layer. In OctMNIST, the complexity of the images causes frequent data transfers between the GPU and memory as the results from each layer are stored and retrieved. This in turn consumes more energy. On the other hand, global pruning updates all layers at once, so there are no extra operations to handle individual layers. This allows for parallel computations, which reduces memory transfers and makes the process more efficient.

Second, idle hardware usage is another important factor that affects energy consumption. GPUs perform best when all their processing cores are active [67, 68]. With global pruning, all layers are computed at the same time, keeping the GPU fully utilized. However, in layer-wise pruning, there are idle periods between each layer's computation while the system prepares the next layer. These idle times are especially noticeable when working with large batch sizes (like 256) and complex medical images, leading to wasted energy. As a result, although layer-wise pruning works well for simpler datasets like MNIST, its sequential approach is less energy-efficient for OctMNIST, particularly when using small-scale models.

Regarding CO_2 emissions and power consumption, both metrics are calculated based on the energy used during the training process. Following the energy consumption metric, we observe that for the experiments where the layer-wise pruning scenario was not as effective, the power consumption only increased slightly. In the worst-case scenario, the power usage increased by 2.01wH, which shows that the negative impact of the pruning scenario on power consumption remained minimal, even when it did not lead to the expected reduction in energy usage. Additionally, the CO_2 emissions in the worst-case scenario, where layer-wise pruning was less effective, increased by less than 1g in total.

While the primary focus of this study is on MLP networks, we also conducted these additional experiments on CNN architectures (using the VGG-16 model) to reinforce and validate our observations. These CNN experiments were performed using the same OctMNIST dataset under similar configurations, particularly focusing on the impact of pruning strategies (*i.e.* global, and layer-wise) on model performance and energy consumption. The top accuracy achieved with various CNN models on this dataset has been reported to be 77.6% [55, 56], which aligns with the dense model's accuracy observed in our study, as presented in Table 3.

When switching to a CNN architecture, the same trends observed in the MLP models are also seen. Table 3 highlights that using a more complex model, like the VGG-16, leads to a significant increase in performance compared to MLPs. The accuracy increases from under 50% in MLP models to nearly 76% in the VGG-16 model. Additionally, when comparing energy consumption during training, we find that the layer-wise pruning method uses less energy than the global pruning approach in all experiments. Fig 8 shows the energy usage during training, where we zoomed in on the y-axis. The current scale in the figure starts from 320,000J to 400,000J to focus on the energy metric more clearly.

Table 3. Model performances for the OctMNIST dataset, evaluated using the VGG-16 architecture. Test accuracy (ACC, %), average inference time (AIT, s), and total inference energy consumption (TIE, J), expressed as relative percentual differences to benchmark values. The rows with a different background represent the best trade-off, and bold cells highlight the best results for each metric.

Model	**Scenario**			**OctMNIST Metrics**		
	Sparsity	**Approach**	**Technique**	**ACC**	**AIT**	**TIE**
VGG-16	Benchmark (0% Sparsity)			75.90%	1.32e-3s	156.76J
	50%	Global	Pre-Training	-1.22	-1.52	-2.23
			SET	-1.62	-3.03	-1.90
		Layer-wise	Pre-Training	**+0.41**	-3.03	-2.29
			SET	-0.27	-2.27	-2.05
	80%	Global	Pre-Training	-1.43	-3.58	-7.34
			SET	+0.27	-3.58	-6.45
		Layer-wise	Pre-Training	-1.08	-4.33	-8.19
			SET	-2.29	**-4.58**	**-8.37**

Fig 8. Comparison of training-phase energy profiles on VGG-16 architecture trained by OctMNIST. Each sub-figure presents five experiments including the two pruning techniques (Pre-Training and SET) and two sparsity levels (50%, and 80%) and Dense verison of the model. In each figure, we zoom the y-axis to show the differences between the global approach that represented by blue bars and the layer-wise approach that represented by orange bars.

In conclusion, our experiments demonstrate that while the MLP architecture shows consistent trends in energy footprint and inference time, the accuracy results are less reliable, especially when using a domain-specific, medical dataset like OctMNIST. Upon switching to a CNN architecture, we were able to achieve accuracy levels that closely match those reported in the benchmark paper [55, 56]. This highlights that model architecture plays a crucial role in achieving high performance. Moreover, the layer-wise pruning technique consistently outperforms the global pruning approach, leading to better energy efficiency and faster inference times. These findings emphasize the importance of selecting the right model architecture and pruning strategy to optimize both performance and efficiency in neural network models. Future work could explore more complex datasets and pruning techniques to further validate these observations and refine model efficiency.

4 Conclusion and Future Work

This study evaluates MLP models across five critical dimensions: model architecture (small-scale 2×16 *vs.* medium-scale 4000-1000-4000 architectures), dataset complexity (FashionMNIST, MNIST, OctMNIST), sparsity levels (50% *vs.* 80%), sparsity techniques (Pre-Training *vs.* SET), and pruning approaches (global *vs.* layer-wise). While the primary focus is on MLPs, we also explore CNN models, such as VGG-16, trained on OctMNIST, EMNIST and CIFAR-10 datasets, in a secondary analysis to provide a broader perspective on pruning strategies and validate the findings in more complex architectures used in image classification tasks.

Our experiments reveal that each combination of these factors produces distinct performance-energy trade-offs, with layer-wise pruning consistently demonstrating advantages compared to global pruning in most scenarios. Most importantly, we also showcase that layer-wise pruning can provide significant savings in inference time (up to 33%) and inference energy usage (up to 56%) with minimal accuracy loss (less than 6%) compared to the benchmark. For CNN models, the savings in average inference time and total inference energy were up to 20% and up to 44%, respectively, with accuracy loss remaining below 3%.

When aiming to achieve optimal model performance, it is crucial to recognize the importance of training the model multiple times to ensure fine-tuning and achieve the best possible performance. Fine-tuning the model typically requires several independent runs, each with different seeds or hyperparameters, to identify the optimal configuration. This process is essential to achieve reliable and consistent results. Sparse neural networks are known to result in models that are faster and more energy-efficient during the test phase, which makes them an attractive choice for deployment. In addition, when it comes to the training phase, our findings indicate that layer-wise pruning consumes less energy compared to global pruning. Despite this, both pruning strategies—whether global or layer-wise—still require more energy than the dense model. This is primarily due to the increased complexity introduced during the training phase, where additional operations are required to apply the sparsity techniques across the network. Thus, in continuous or iterative training conditions, where models require frequent retraining to preserve their optimal accuracy, layer-wise pruning proves to be the more sustainable and energy-efficient strategy with minimum performance decrease.

The primary insights from our analysis indicate that layer-wise pruning, when applied with an appropriate model architecture can effectively reduce energy consumption during inference. Additionally, the maximum memory allocation metric provides valuable insight into energy consumption during training, as memory usage directly impacts energy usage. This metric can be a useful alternative in scenarios where direct monitoring of energy consumption is not possible. In addition, this observation is not limited to MLP architectures and the same behavior is observed in CNN architectures with more complex datasets. As we discussed in Sect. 3.5, training the sparse model with the layer-wise approach requires less energy than training the model where sparsity is applied globally across all layers.

While this study provides valuable insights, several areas remain unexplored, offering significant opportunities for further research. One of the potential directions is to explore convolutional layers, by applying pruning techniques specifically to the feature extraction part of the network in a layer-wise manner. While applying the mid-training sparsity technique for these complex architectures needs some modifications, these architectures are able to achieve much higher accuracy in the dense version of the model [61]. Moreover, another area to explore in the future is the use of transformer models. These models have a different architecture compared to MLPs or CNNs, and applying layer-wise pruning to them would require designing specific algorithms that fit their structure.

Current magnitude-based pruning ignores the energy cost of memory access patterns. Future studies should integrate hardware performance counters (*e.g.*, DRAM bandwidth utilization) directly into the pruning loss function, creating an energy-accuracy Pareto optimization framework. For example, weights could be removed not just based on magnitude but on their contribution to energy-intensive memory fetching. This aligns with our finding that memory bottlenecks disproportionately affect small models on complex datasets, and could yield hardware-specific sparse architectures where the 'energy footprint' of each connection is explicitly optimized.

Furthermore, the energy efficiency of sparse networks must ultimately be validated on the deployment hardware. Future work should co-design pruning algorithms with edge device constraints, such as the memory hierarchy of ARM Cortex CPUs or Jetson GPUs. For instance, pruning could enforce bank-aware sparsity patterns that match SRAM burst lengths, minimizing energy wasted on partial memory fetches. This direction directly develops our energy measurements by translating layer-wise pruning benefits into actual battery life gains for portable OCT diagnostic tools, addressing the real-world concept of our findings.

Moreover, exploring the performance of the models pruned using a layer-wise approach or pruned globally on IoT devices, including smartphones or single-board computers, presents a valuable direction for future research. While our focus in this work has been on comparing model performance and training metrics, evaluating these models in real-world environments such as smartphones or single-board computers requires additional considerations. Mobile phones, for instance, often have multiple background processes running due to various installed applications, which can impact the available resources for running machine learning models. Similarly, when deploying models on single-board computers, not only is high model performance necessary, but a robust software architecture that supports running multiple models simultaneously is also required. Future work could involve investigating these aspects more deeply by measuring additional metrics such as real-time inference speed, power consumption, and resource management efficiency, to ensure that models are both high-performing and adaptable to constrained environments.

It would also be valuable to further explore the impact of layer-wise pruning on the feature-extraction part of CNN architectures, which are among the most widely used in classification tasks and industrial applications. Designing a robust strategy to prune this part of the network could provide deeper insights into how layer-wise pruning works in more complex models, possibly leading to more efficient networks. This is especially relevant when using more complex datasets or domain-specific datasets like MedMNIST, a medical dataset, or VisA and MVTech, which are used for anomaly detection in industrial environments. These datasets are more challenging due to their higher resolution and complexity, which makes them suitable for testing how well layer-wise pruning scales in real-world scenarios. This research could provide more insights into how pruning techniques perform in industrial applications and help improve model efficiency.

Looking ahead, a promising avenue for future work could involve examining the generalizability and transferability of pruning strategies from different domains,particularly network science (NS). It would be interesting to investigate how insights from NS could inform the development of more robust pruning strategies in specialized real-world applications.

Supporting Information

S1 File. Additional experimental details and data required for the replication of the study.

References

1. Wu Q, Xie Q, Xia D. Application of Computer Vision and Deep Learning Neural Network in Multi-Modal Information Fusion. In: 2024 International Conference

on Data Science and Network Security (ICDSNS); 2024. p. 1–5. Available from: https://doi.org/10.1109/ICDSNS62112.2024.10691099.

2. Ishitaki T, Obukata R, Oda T, Barolli L. Application of Deep Recurrent Neural Networks for Prediction of User Behavior in Tor Networks. In: 2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA); 2017. p. 238–243. Available from: https://doi.org/10.1109/WAINA.2017.63.
3. Profentzas C, Almgren M, Landsiedel O. MiniLearn: On-Device Learning for Low-Power IoT Devices. In: Proceedings of the 2022 International Conference on Embedded Wireless Systems and Networks. EWSN '22. New York, NY, USA: Association for Computing Machinery; 2023. p. 1–11. Available from: https://dl.acm.org/doi/10.5555/3578948.3578949.
4. Islam MR, Zamil MZH, Rayed ME, Kabir MM, Mridha MF, Nishimura S, et al. Deep learning and computer vision techniques for enhanced quality control in manufacturing processes. IEEE Access. 2024; p. 1–1. Available from: https://doi.org/10.1109/ACCESS.2024.3453664.
5. Moradi M, Moradi M, Palazzo S, Rundo F, Spampinato C. Image CAPTCHAs: When deep learning breaks the mold. IEEE Access. 2024;12:112211–112231. Available from: https://doi.org/10.1109/ACCESS.2024.3442976.
6. Douzandeh Zenoozi A, Majidi B, Cavallaro L, Liotta A. Hybrid edge-cloud federated learning: The case of lightweight smoking detection. In: Lecture Notes in Computer Science. Lecture notes in computer science. Cham: Springer Nature Switzerland; 2025. p. 150–159. Available from: https://doi.org/10.1007/978-3-031-78090-5_13.
7. Zhao H, Zhang S, Peng X, Lu Z, Li G. Improved object detection method for autonomous driving based on DETR. Front Neurorobot. 2024;18:1484276. Available from: https://doi.org/10.3389/fnbot.2024.1484276.
8. Almanzor E, Anvo NR, Thuruthel TG, Iida F. Autonomous detection and sorting of litter using deep learning and soft robotic grippers. Front Robot AI. 2022;9:1064853. Available from: https://doi.org/10.3389/frobt.2022.1064853.
9. Shaik F, Rajesh Y, Gudur NA, Dash JK. Deep CNN in healthcare. In: Deep Learning in Biomedical Signal and Medical Imaging. Boca Raton: CRC Press; 2024. p. 221–236. Available from: https://doi.org/10.1201/9781032635149-14.
10. Wambura S, Li H. Deep and Confident Image Analysis for Disease Detection. In: Proceedings of the 2020 2nd International Conference on Video, Signal and Image Processing. VSIP '20. New York, NY, USA: Association for Computing Machinery; 2021. p. 91–99. Available from: https://doi.org/10.1145/3442705.3442720.
11. Janjic S, Thulasiraman P, Bruce N. Redundancy in convolutional neural networks: Insights on model compression and structure. In: 2018 International Joint Conference on Neural Networks (IJCNN). IEEE; 2018. p. 1–8. Available from: https://doi.org/10.1109/IJCNN.2018.8489083.

12. Han S, Pool J, Tran J, Dally W. Learning both Weights and Connections for Efficient Neural Network. In: Cortes C, Lawrence N, Lee D, Sugiyama M, Garnett R, editors. Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 1. vol. 28 of NIPS'15. Cambridge, MA, USA: MIT Press; 2015. p. 1135–1143. Available from: `https://dl.acm.org/doi/10.5555/2969239.2969366`.

13. Mocanu DC, Mocanu E, Stone P, Nguyen PH, Gibescu M, Liotta A. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. Nature Communications. 2018;9(1):2383. Availble from: `https://doi.org/10.1038/s41467-018-04316-3`.

14. Galchonkov O, Nevrev A, Glava M, Babych M. Exploring the efficiency of the combined application of connection pruning and source data pre-processing when training a multilayer perceptron. East-Eur J Enterp Technol. 2020;2(9 (104)):6–13. Available from: `https://ssrn.com/abstract=3712182`.

15. Fel T, Bethune L, Lampinen AK, Serre T, Hermann K. Understanding visual feature reliance through the lens of complexity. Proceedings Of The 38th International Conference On Neural Information Processing Systems. (2024). Available from: `https://dl.acm.org/doi/10.5555/3737916.3740150`.

16. Furukawa T, Zhao Q. On extraction of rules from deep learner: The deeper, the better? In: 2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech). IEEE; 2017. p. 54–60. Available from: `https://doi.org/10.1109/DASC-PICom-DataCom-CyberSciTec.2017.25`.

17. Lee EH, Kim H. Feature-based interpretation of the deep neural network. Electronics (Basel). 2021;10(21):2687. doi:10.3390/electronics10212687. Available from: `https://doi.org/10.3390/electronics10212687`.

18. Popescu MC, Balas VE, Perescu-Popescu L, Mastorakis N. Multilayer perceptron and neural networks. WSEAS Transactions on Circuits and Systems. 2009;8(7):579–588. Available from: `https://dl.acm.org/doi/abs/10.5555/1639537.1639542`.

19. Douzandeh Zenoozi A, Erhan L, Liotta A, Cavallaro L. A Comparative Study of Neural Network Pruning Strategies for AI Deployment on Edge Devices. In: Cherifi H, Donduran M, Rocha LM, Cherifi C, Varol O, editors. Complex Networks & Their Applications XIII. Cham: Springer Nature Switzerland; 2025. p. 3–14. Available from: `https://doi.org/10.1007/978-3-031-82427-2_1`.

20. Lamichhane BR, Srijuntongsiri G, Horanont T. CNN based 2D object detection techniques: a review. Front Comput Sci. 2025;7(1437664). doi:10.3389/fcomp.2025.1437664. Available from: `https://doi.org/10.3389/fcomp.2025.1437664`.

21. Alzubaidi L, Zhang J, Humaidi AJ, Al-Dujaili A, Duan Y, Al-Shamma O, et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. J Big Data. 2021;8(1):53. doi:10.1186/s40537-021-00444-8. Available from: `https://doi.org/10.1186/s40537-021-00444-8`.

22. Tsirtsakis P, Zacharis G, Maraslidis GS, Fragulis GF. Deep learning for object recognition: A comprehensive review of models and algorithms. International Journal of Cognitive Computing in Engineering. 2025;6:298–312. Available from: https://doi.org/10.1016/j.ijcce.2025.01.004.

23. Ersavas T, Smith MA, Mattick JS. Novel applications of Convolutional Neural Networks in the age of Transformers. Sci Rep. 2024;14(1):10000. doi:10.1038/s41598-024-60709-z. Available from: https://doi.org/10.1038/s41598-024-60709-z.

24. Chai J, Zeng H, Li A, Ngai EWT. Deep learning in computer vision: A critical review of emerging techniques and application scenarios. Mach Learn Appl. 2021;6(100134):100134. doi:10.1016/j.mlwa.2021.100134. Available from: https://doi.org/10.1016/j.mlwa.2021.100134.

25. Douzandeh Zenoozi A, Erhan L, Liotta A, Cavallaro L. A comparative study of neural network pruning strategies for industrial applications. Front Comput Sci. 2025;7. doi:10.3389/fcomp.2025.1563942. Available from: https://doi.org/10.3389/fcomp.2025.1563942.

26. Zhang Z, Tao R, Zhang J. Neural network pruning by gradient descent; 2023. Available from: https://arxiv.org/abs/2311.12526.

27. Soumyalatha N, Manjunath R K. Optimized Convolutional Neural Network at the IoT edge for image detection using pruning and quantization. Multimed Tools Appl. 2024;84(9):5435–5455. Available from: https://doi.org/10.1007/s11042-024-20523-1.

28. Frankle J, Carbin M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv [csLG]. 2018. doi:10.48550/arXiv.1803.03635. Available from: https://doi.org/10.48550/arXiv.1803.03635.

29. Erdős P, Rényi A. On random graphs. I. Publ Math Debrecen. 2022;6(3-4):290–297. Available from: https://www.bibsonomy.org/bibtex/2420b83c1533188c0b54bd1f6eea2b782/krevelen

30. Kalyanam LK, Katkoori S. Unstructured pruning for multi-layer perceptrons with tanh activation. In: 2023 IEEE International Symposium on Smart Electronic Systems (iSES). IEEE; 2023. p. 69–74. Available from: https://doi.org/10.1109/iSES58672.2023.00025.

31. Medeiros CMS, Barreto GA. A novel weight pruning method for MLP classifiers based on the MAXCORE principle. Neural Comput Appl. 2013;22(1):71–84. Available from: https://doi.org/10.1007/s00521-011-0748-6

32. Thomas P, Suhner MC. A new multilayer perceptron pruning algorithm for classification and regression applications. Neural Process Lett. 2015;42(2):437–458. Available from: https://doi.org/10.1007/s11063-014-9366-5.

33. Lee N, Ajanthan T, Torr PHS. SNIP: Single-shot Network Pruning based on Connection Sensitivity; 2019. Available from: https://doi.org/10.48550/arXiv.1810.02340.

34. Fan C, Li J, Zhang T, Ao X, Wu F, Meng Y, et al. Layer-wise Model Pruning based on Mutual Information. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. Stroudsburg, PA, USA: Association for Computational Linguistics; 2021. p. 3079–3090. Available from: https://doi.org/10.18653/v1/2021.emnlp-main.246.

35. Tripp CE, Perr-Sauer J, Gafur J, Nag A, Purkayastha A, Zisman S, et al.. Measuring the energy consumption and efficiency of deep neural networks: An empirical analysis and design recommendations; 2024. Available from: `https://doi.org/10.48550/arXiv.2403.08151`.

36. Kamolov S. Optimizing model pruning for energy-efficient deep learning. Annals of Mathematics and Computer Science. 2024;25:112–119. Available from: `https://doi.org/10.56947/amcs.v25.428`.

37. Li H, Meng L. Hardware-aware approach to deep neural network optimization. Neurocomputing. 2023;559(126808):126808. Available from: `https://doi.org/10.1016/j.neucom.2023.126808`.

38. Widmann T, Merkle F, Nocker M, Schöttle P. Pruning for power: Optimizing energy efficiency in IoT with neural network pruning. In: Communications in Computer and Information Science. Communications in computer and information science. Cham: Springer Nature Switzerland; 2023. p. 251–263. Available from: `https://doi.org/10.1007/978-3-031-34204-2_22`.

39. Blalock D, Gonzalez Ortiz JJ, Frankle J, Guttag J. What is the State of Neural Network Pruning? In: Dhillon I, Papailiopoulos D, Sze V, editors. Proceedings of Machine Learning and Systems. vol. 2; 2020. p. 129–146. Available from: `https://proceedings.mlsys.org/paper_files/paper/2020/file/6c44dc73014d66ba49b28d483a8f8b0d-Paper.pdf`.

40. Liu Z, Li J, Shen Z, Huang G, Yan S, Zhang C. Learning Efficient Convolutional Networks through Network Slimming. In: 2017 IEEE International Conference on Computer Vision (ICCV); 2017. p. 2755–2763. Available from: `10.1109/ICCV.2017.298`.

41. Tmamna J, Ayed EB, Fourati R, Gogate M, Arslan T, Hussain A, et al. Pruning deep neural networks for green energy-efficient models: A survey. Cognit Comput. 2024;. Available from: `https://doi.org/10.1007/s12559-024-10313-0`.

42. Zhang B, Wang T, Xu S, Doermann D. Network pruning. In: Computational Intelligence Methods and Applications. Singapore: Springer Nature Singapore; 2024. p. 131–217.

43. Belay K. Gradient and mangitude based pruning for sparse deep Neural Networks. Proc Conf AAAI Artif Intell. 2022;36(11):13126–13127. Available from: `https://doi.org/10.1609/aaai.v36i11.21699`.

44. Rathor VS, Singh M, Gupta R, Sharma GK, Sahoo KS, Bhuyan M. Towards designing an energy efficient accelerated sparse convolutional neural network. In: 2024 IEEE 36th International Conference on Tools with Artificial Intelligence (ICTAI). IEEE; 2024. p. 969–974. Available from: `https://doi.org/10.1109/ICTAI62512.2024.00139`.

45. Ghosh SK, Kundu S, Raha A, Mathaikutty DA, Raghunathan V. HARVEST: Towards efficient sparse DNN accelerators using programmable thresholds. In: 2024 37th International Conference on VLSI Design and 2024 23rd International Conference on Embedded Systems (VLSID). IEEE; 2024. p. 228–234. Available from: `https://doi.org/10.1109/VLSID60093.2024.00044`.

46. Wu S, Li G, Deng L, Liu L, Wu D, Xie Y, et al. L1 -norm batch normalization for efficient training of deep neural networks. IEEE Trans Neural Netw Learn Syst. 2019;30(7):2043–2051. Available from: `https://doi.org/10.1109/TNNLS.2018.2876179`.

47. Shi Y, Tang A, Niu L, Zhou R. Sparse optimization guided pruning for neural networks. Neurocomputing. 2024;574(127280):127280. Available from: `https://doi.org/10.1016/j.neucom.2024.127280`.

48. Tibshirani R. Regression shrinkage and selection via the lasso. J R Stat Soc Series B Stat Methodol. 1996;58(1):267–288. Available from: `https://www.jstor.org/stable/2346178`.

49. Hoerl AE, Kennard RW. Ridge regression: Biased estimation for nonorthogonal problems. Technometrics. 1970;12(1):55–67. Available from: `https://doi.org/10.2307/1271436`.

50. Xiao H, Rasul K, Vollgraf R. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms; 2017. doi:10.48550/arXiv.1708.07747. Available from: `https://arxiv.org/abs/1708.07747`.

51. Cohen G, Afshar S, Tapson J, van Schaik A. EMNIST: an extension of MNIST to handwritten letters; 2017. Available from: `https://arxiv.org/abs/1702.05373`.

52. Krizhevsky A. Learning Multiple Layers of Features from Tiny Images; 2009. Available from: `https://api.semanticscholar.org/CorpusID:18268744`.

53. Abouelnaga Y, Ali OS, Rady H, Moustafa M. CIFAR-10: KNN-based ensemble of classifiers; 2016. Available from: `https://doi.org/10.48550/arXiv.1611.04905`.

54. Unknown. CIFAR-10; 2009. Available from: `https://doi.org/10.24432/C5889J`.

55. Yang J, Shi R, Ni B. MedMNIST Classification Decathlon: A Lightweight AutoML Benchmark for Medical Image Analysis. In: IEEE 18th International Symposium on Biomedical Imaging (ISBI); 2021. p. 191–195. Available from: `https://api.semanticscholar.org/CorpusID:225094295`.

56. Yang J, Shi R, Wei D, Liu Z, Zhao L, Ke B, et al. MedMNIST v2-A large-scale lightweight benchmark for 2D and 3D biomedical image classification. Scientific Data. 2023;10(1):41. Available from: `https://doi.org/10.1038/s41597-022-01721-8`.

57. Yang Z, Meng L, Chung JW, Chowdhury M. Chasing low-carbon electricity for practical and sustainable DNN training; 2023. Available from: `https://doi.org/10.48550/arXiv.2303.02508`.

58. Chung JW, Gu Y, Jang I, Meng L, Bansal N, Chowdhury M. Reducing Energy Bloat in Large Model Training. In: Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles. SOSP '24. New York, NY, USA: Association for Computing Machinery; 2024. p. 144–159. Available from: `https://doi.org/10.1145/3694715.3695970`.

59. You J, Chung JW, Chowdhury M. Zeus: Understanding and Optimizing GPU Energy Consumption of DNN Training. In: 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23). Boston, MA: USENIX Association; 2023. p. 119–139. Available from: `https://www.usenix.org/conference/nsdi23/presentation/you`.

60. Di Salvo F, Doerrich S, Ledig C. MedMNIST-C: Comprehensive benchmark and improved classifier robustness by simulating realistic image corruptions; 2024. Available from: `https://arxiv.org/abs/2406.17536`.

61. Doerrich S, Di Salvo F, Brockmann J, Ledig C. Rethinking model prototyping through the MedMNIST+ dataset collection. Sci Rep. 2025;15(1):7669. doi:10.1038/s41598-025-92156-9. Available from: `https://doi.org/10.1038/s41598-025-92156-9`.

62. Peng X. Global carbon intensity of electricity.xlsx; 2023. Available from: `https://figshare.com/articles/dataset/Global_carbon_intensity_of_electricity_xlsx/23592966`.

63. Maps E. The Leading Resource for 24/7 CO2 Grid Data; 2017-2024. Available from: `https://www.electricitymaps.com/`.

64. Deb N, Rahman T. An efficient VGG16-based deep learning model for automated potato pest detection. Smart Agric Technol. 2025;12(101409):101409. Available from: `https://doi.org/10.1016/j.atech.2025.101409`.

65. Lu Y. LSEVGG: An attention mechanism and lightweight-improved VGG network for remote sensing landscape image classification. Alex Eng J. 2025;127:943–951. Available from: `https://doi.org/10.1016/j.aej.2025.06.053`.

66. Khan MA, Auvee RBZ. Comparative analysis of resource-efficient CNN architectures for Brain Tumor classification; 2024. doi: 10.1109/ICCIT64611.2024.11021970 Available from: `https://ieeexplore.ieee.org/document/11021970`.

67. Lastovetsky A, Manumachu RR. Energy-efficient parallel computing: Challenges to scaling. Information (Basel). 2023;14(4):248. doi:10.3390/info14040248.

68. Rofouei M, Stathopoulos T, Ryffel S, Kaiser W, Sarrafzadeh M. Energy-aware high performance computing with graphic processing units. In: Proceedings of the 2008 Conference on Power Aware Computing and Systems. HotPower'08. USA: USENIX Association; 2008. p. 11. doi:10.5555/1855610.1855621. Available from: `https://dl.acm.org/doi/10.5555/1855610.1855621`.