Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра математики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: **«Метаевристичні алгоритми для обрізки нейронних мереж»**

Виконала: студентка 4-го року навчання
освітньої програми «Прикладна математика»,
спеціальності 113 Прикладна математика

Котляренко Анастасія Юріївна

Керівник: Швай Н. О.
кандидат фіз.-мат. наук, доцент

Рецензент: _____

Кваліфікаційна робота захищена
з оцінкою _____

Секретар ЕК_____
(підпис)

«_____» _____ 20___р.

Київ - 2025

Міністерство освіти і науки України
Національний університет «Києво-Могилянська академія»
Факультет інформатики
Кафедра математики

ЗАТВЕРДЖУЮ
Зав.кафедри математики,
проф., доктор фіз.-мат. наук
_____*Олійник Б.В.*
(підпис)
“____”_____ 2021

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
для кваліфікаційної роботи
студенту 4-го курсу, факультету інформатики
Котляренко Анастасії Юріївні

Тема: «Метаевристичні алгоритми для обрізки нейронних мереж»

Зміст кваліфікаційної роботи:

Annotation
1. Introduction
2. Literature Review
3. Methodology
4. Experimental Section

Conclusion
References

Дата видачі “____”_____ 2025 Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Графік підготовки кваліфікаційної роботи до захисту

Графік узгоджено «_____» _____ 2025р.

№ з/п	Перелік робіт	Термін виконання етапу	Підпис наукового керівника	Дата ознайомлення наукового керівника	Примітка
1.	Отримання теми кваліфікаційної роботи.	15.10.2024			
2.	Ознайомлення з темою кваліфікаційної роботи.	22.10.2024			
3.	Розробка плану та структури роботи.	20.11.2024			
4.	Робота з науковою літературою, опис основних означень.	16.12.2024			
5.	Дослідження результатів отриманих в літературі.	15.01.2025			
6.	Робота над текстовим оформленням результатів.	19.04.2025			
7.	Попередній аналіз кваліфікаційної роботи. Виправлення помилок.	17.05.2025			
8.	Попередній захист кваліфікаційної роботи.	23.05.2025			
9.	Захист кваліфікаційної роботи.	05.06.2025			

Науковий керівник _____
(ПІБ)

Виконавець кваліфікаційної роботи _____
(ПІБ)

Contents

Abstract

This work studies neural network pruning with metaheuristic optimization methods. Pruning was formulated as an optimization problem with a target function that is a weighted sum of neural network accuracy and sparsity. This problem was solved with stochastic metaheuristic methods (Genetic Algorithm and Particle Swarm Optimization) that generate binary masks. Obtained results demonstrate that pruning with metaheuristic methods is comparative with L_2 pruning when finetuning is possible and is significantly more performant when no post-pruning finetuning is available.
Keywords: neural networks, pruning, metaheuristic, Genetic Algorithm, Particle Swarm Optimization, L_2 pruning.

1 Introduction

Neural networks are already capable of solving problems of classification, regression, optimization, comparison, inference, forecasting, and much more. They have become an extremely powerful tool capable of even replacing humans in some cases. To handle complex tasks, neural networks perform a large number of computations. As a result, they require extended processing time, as well as a significant number of resources and memory, which is sometimes redundant.
One of the ways to reduce a neural network is pruning. This means removing neurons, filters, or channels that are less important to reduce the number of parameters. One of the most popular methods is L_2 pruning, which removes filters at once, but this method does not always give great performance, especially when the model's structure is not well aligned with the importance of some parts.
In this paper, we explore two metaheuristic algorithms: Genetic Algorithm and Particle Swarm Optimization.

1.1 Relevance

Nowadays, many neural networks help people in all spheres of life, but not all of them have the ability to use powerful servers. Therefore, pruned neural networks can work just as well without using a lot of memory, for example, on a smartphone, laptop, or tablet. Metaheuristic methods are well suited for this task because they do not require knowledge of the model, take into account the relationships between different parts, and do not require gradient calculations.
Although there is already quite a bit of research on this topic, these methods are not yet well understood. In particular, they have not been compared much with classical pruning methods. In this paper, we consider GA and PSO methods with different hyperparameters and compare the results in terms of accuracy, sparsity, and pruning time with the classical L_2 approach.

1.2 Research Aim and Objectives

The aim of this work is to investigate the effectiveness of metaheuristic algorithms, namely the Genetic Algorithm and the Particle Swarm Optimisation, for pruning

the parameters of the convolutional neural network LeNet5 in order to reduce the model size without significant loss of accuracy on the MNIST dataset.

2 Literature Review

2.1 Neural Network Pruning: Overview and Classification

From the first days of neural networks research, it was mentioned that many parameters in networks could be removed without significant loss in performance. With the advent of deep learning models, this observation became more relevant. The Lottery Ticket Hypothesis is an idea that supports this view. It says that inside a large randomly initialized neural network, there is a smaller subnetwork, like a "winning ticket", which can perform just as well or even better than the full model[4].

One of the most effective methods of reducing the complexity of a neural network is pruning, which involves removing less important parameters. The goal may be to reduce the size of the model, minimize the computation time, and lower memory consumption without greatly reducing accuracy.

2.1.1 By granularity

Pruning is distinguished by granularity, namely structured and unstructured. Structured pruning removes individual weighting factors, regardless of their location. This allows for high resolution but requires specialized hardware or libraries [7]. In turn, structured pruning removes filters, channels, neurons or layers. This approach is better for practical applications, as it resizes the tensors and can be implemented on a not powerful machine[19][20].

2.1.2 By importance evaluation

Another key characteristic is how the importance of parameters is assessed. Magnitude-based pruning is one of the most common approaches, which removes parameters with absolute values[8]. There are also methods that take into account gradients, for example, the product of weight and weight gradient or activation effect, which analyses how much a parameter affects the neuron's output[12]. Methods that estimate the change in the loss function when weight is removed are Hessian-based saliency methods, such as Optimal Brain Damage[17] and Optimal Brain Surgery[11].

2.1.3 By application stage

The pruning of neural networks is also classified by the application stage. One of the most popular is post-training pruning. First, the model is trained on the dataset, and then it is pruned with possible fine-tuning[7]. Moreover, this method was used in our study. The other method is iterative pruning, which means that the model is pruned and fine-tuned for several iterations. This allows to maintain high accuracy of the model[5]. SNIP and similar methods perform pruning before training, and the importance of connections is measured based on the loss gradient[18]. There is also dynamic pruning, in which the network structure is reduced during training.

2.2 Metaheuristic Algorithms in Model Compression

Metaheuristic algorithms are becoming more popular in neural network pruning because of their ability to efficiently search for good combinations of parameters in large and complex spaces. In pruning these methods help with finding binary masks that increase sparsity with little loss of model accuracy. Two of the most common methods include Genetic Algorithm and Particle Swarm Optimisation[29]. The performance of each candidate is evaluated based on the model's accuracy on a validation set and the sparsity level of the network.
Genetic Algorithms inspired by natural selection include processes such as crossover, mutation, and selection of population. These algorithms are useful when it is complicated to define the importance of each weight analytically, without relying on gradients or norms. GA looks for an optimal mask globally and this mask determines which weights should be kept and which should be reset to zero[22].
The Particle Swarm Optimisation method simulates the behaviour of a group of particles that improve their position based on their own and global experience. Also, a bit of randomness is added to make more unpredictable masks[15].
In addition to these metaheuristics, Simulated Annealing, Ant Colony Optimization, Differential Evolution, and others are also used to prune neural networks. These methods have their own advantages and algorithms that reduce the network in one way or another without calculating gradients.

2.3 Applications of Genetic Algorithms and Particle Swarm Optimisation in Pruning

Recently, researchers have been exploring the use of metaheuristic algorithms for deep neural network pruning. These approaches are especially interesting because they can perform black box optimization, which means that they do not use gradients to estimate the weight importance[24]. In metaheuristic methods, the model itself acts as a black box, we do not know in detail how the parameters of the method affect the output, but simply calculate the accuracy as an output metric and optimize it.

2.3.1 Usage of GA in Pruning

Yang et al.[30] propose in the study a multi-objective pruning framework based on the GA algorithm for CNN comparison. This work aims to achieve two goals to preserve the accuracy of the model and to remove as many parameters as possible. Experiments have shown that a multi-objective search using the heuristic GA method on the standard MNIST dataset reduces the model by 95.42% while almost maintaining the classification accuracy.

Hancock[10] in his work was one of the first to propose to prune a neural network using metaheuristic methods, namely the Genetic Algorithm. A full connected net is first trained using backpropagation on the target problem. The GA removes connections before retraining the network. The pruned model achieves good results on the validation set. However, the pruned network does not train well from random start weights. To solve this issue a gradually increasing random weight component is introduced during the GA process. The results of this paper showed that the network after GA performs reliably even when starting from random initial weights. The author also highlights the permutation problem, which means that neural networks that behave identically can have different internal representations, which complicates for algorithm to find the optimal structure. This problem is partially solved by using pre-trained weights.

2.3.2 Usage of PSO in Pruning

Tu et al.[26] developed a method in which a modified discrete version of the particle swarm optimization is used to find the optimal masks that determine how much weight to hold. The results showed that this approach can reduce the size of the model without losing accuracy.
Huang et al.[13] propose PSOPruner, a pruning framework for CNNs designed for photovoltaic module defect classification. Their approach uses the PSO algorithm to improve the pruning ratio on a layer-by-layer basis, unlike the earlier mentioned PSO-based pruning method that relies on binary masks to remove individual connections. PSOPruning uses real numbers to determine the best pruning ratio for each layer and showed improved classification performance.
In another paper, Mayanglambam et al.[21] combine the PSO algorithm with clustering to detect outliers based on KNNs. This approach is not explicitly aimed at pruning neural networks, but it demonstrates the flexibility of PSO. In the study PSO was used by the authors to optimize cluster selection and remove noisy and less important data points, improving the performance of the KNN classifier.

2.4 Challenges and Gaps in Existing Research

Although metaheuristic methods have been used for neural network pruning for many years, several important challenges remain. First, a lot of GA and PSO approaches rely on binary representations, which limit the search space and require more complex architectures. This makes it difficult to apply them to larger models like ResNet or YOLO. Second, most experiments are still conducted on simple, well-known datasets. That does not reflect the complexity of real-world tasks. It complicates the evaluation of methods in practice. Third, the issue of outliers and noises is not taken into consideration. In spite of the fact that in many cases it can negatively affect model robustness. Finally, there's a lack of comparative studies between different metaheuristics and traditional pruning methods. Researchers tend to use different definitions of sparsity or objective functions, which makes it difficult to evaluate and compare pruning strategies across studies.

3 Methodology

3.1 Problem Statement and Research Objectives

3.1.1 Formalisation of the optimisation problem

Let $f(\theta, x)$ indicate a neural network with parameters $\theta \in \mathbb{R}^n$ and input a vector $x \in \mathbb{R}^d$. We define a binary pruning mask $m_i \in \{0,1\}^n$, where $m_i = 1$ indicates retention of parameter θ_i, and $m_i = 0$ indicates pruning. The pruned network is represented as

$$f_{\text{pruned}}(x) = f(m \odot \theta, x),$$

where $\odot$ denotes element-wise multiplication.
The goal is to find a mask m that minimizes the multi-objective loss function and balances between model performance and model sparsity:

$$\mathcal{L}(m) = \lambda_1 \cdot \text{Error}(f(m \odot \theta, x)) + \lambda_2 \cdot (1 - \text{Sparsity}(m)),$$

where
$\lambda_1, \lambda_2 \in [0,1]$ are hyperparameters that control the trade-off between accuracy and sparsity;
$\text{Error}(f(m \odot \theta, x))$ represents the prediction error of the pruned model;
the sparsity indicator shows us how much the model has shrunk and helps us discard masks that are not important. $\text{Sparsity}(m)$ measures the proportion of parameters that are pruned and calculated as the following:

$$\text{Sparsity}(m) = \frac{1}{n} \sum_{i=1}^{n} (1 - m_i)$$

3.1.2 Definition of Search Space

The optimization is performed over a discrete hypercube in n dimensional space, that defines the discrete search space of all possible binary masks:

$$m \in \{0,1\}^n.$$

A continuous relaxation of binary mask $z \in [0,1]^n$ can be used with a threshold

value:

$$m_i = \begin{cases} 1, & \text{if } z_i > \delta, \\ 0, & \text{otherwise,} \end{cases}$$

where $\delta \in (0, 1)$ is a threshold parameter that determines whether an element in the vector z is interpreted as 1 (retained) or 0 (pruned).

3.2 Description of Pruning Algorithms

3.2.1 GA pruning algorithm

The Genetic Algorithm is a part of the metaheuristic and was inspired by natural biological processes such as selection, crossover, and mutation. In the problem of neural network pruning the goal of the approach is to make a binary mask $m \in \{0, 1\}^n$ where $m_i = 1$ means preserving the weight of θ and $m_i = 0$ means zero it out. Therefore, each individual in the population represents a vector of length n that corresponds to the number of parameters in the network. It is important to note that this method is stochastic, meaning that the results can vary with each new run. This happens because of random starting values and processing of it with some randomness.
The algorithm includes the following main steps[6]

1. Initialization of the population. The process of population initialization begins by randomly generating an initial set of binary masks. The size of population is chosen in advance.

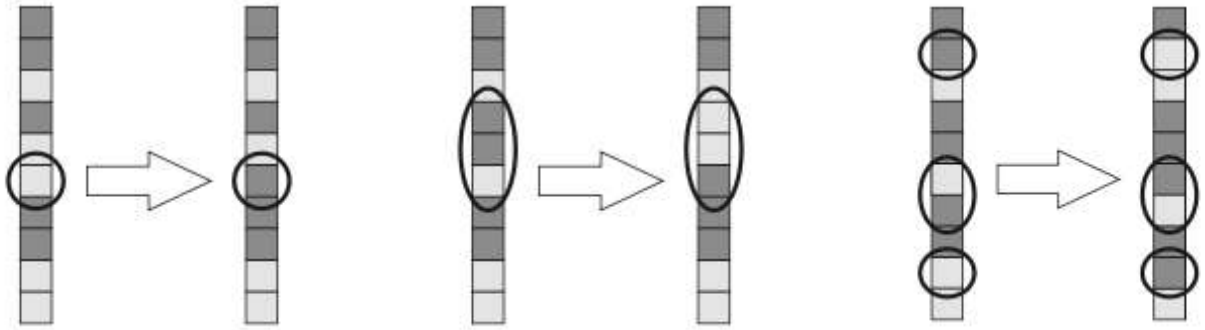
2. Evaluation (Fitness Calculation). For each mask, the accuracy is calculated on the validation set. Sometimes the multi-objective loss is used.

3. Selection. The best masks are chosen using tournament selection with the highest fitness[28].

4. Crossover. In the selected pairs some bits are taken from one of the parents, others from the other and we are getting a new child.

5. Mutations. Some individual bits are accidentally flipped with a small predefined probability p_{mut}. This helps maintain genetic diversity in the population.

Figure 1 shows different types of mutations. In Figure 1a, a single gene is mutated, i.e., only one element in the mask is changed to the opposite. Figure 1b shows the replacement of a group of genes, and Figure 1c shows the mutation of several genes in different locations, single and sequential.



(a) Single-gene mutation (b) Multi-gene mutation (c) Multi-gene mutation

Figure 1: Types of mutation[28]

6. Replacement. The new generation replaces the old one.

7. Stopping. The algorithm is a repeated fixed number of times or until the accepted accuracy is reached

Pseudocode for GA:

Input: M – pretrained model, D_{val} – validation loader,
G – number of generations, N – population size, s – elite size,
p_{mut} – mutation probability, λ_1, λ_2 – fitness weights

Initialize population P with N binary masks
for each mask m in P **do**
    Load model with original weights
    Apply m as binary mask
    Compute fitness of m as weighted sum of accuracy and sparsity
end for
for $g = 1$ to G **do**
    Evaluate fitness for each mask in population
    Sort population by fitness
    Select top s masks using tournament selection as elite
    Initialize *new_population* with elite masks

12: **while** population size $< N$ **do**
13: Select two parents from elite randomly
14: Perform crossover to create a child
15: **for** each bit in child **do**
16: Apply mutation with p_{mut} probability (flip the bit)
17: **end for**
18: Add child to *new_population*
19: **end while**
20: Replace population P with *new_population*
21: **end for**
22: **for** each mask m in final population **do**
23: Evaluate accuracy
24: **end for**
25: **return** mask with highest accuracy

One of the biggest advantages of a Genetic Algorithm is that they do not need to calculate gradients or special weight estimates[25]. Instead, they try different combinations of weights and evaluate the model with each set. Another advantage of GA is the ability to use it for structured and unstructured pruning, as the algorithm can remove not only individual weights but also filters and channels[14]. Also, GAs are well suited for tasks where a large search space is used. Through the use of selection, crossover, and mutation, GAs can find good results where classical methods get stuck[30].

Despite the advantages, this approach also has some disadvantages. One of the biggest disadvantages is that this algorithm is computationally expensive, as the entire model is calculated for each individual, especially if there are a large number of generations[1]. Another disadvantage is that it is stochastic, meaning that the results vary depending on the initialisation of the model and random factors.

3.2.2 PSO pruning algorithm

The Particle Swarm Optimization (PSO) algorithm belongs to metaheuristics and was inspired by the behaviour of a wasp swarm, a flock of birds, and a shoal of fish. Similar to GA, it is stochastic and the results change with each new run. The algorithm represents each candidate solution, called a particle, as a vector

of real-values $z \in [0,1]^n$, where n is the number of parameters in the model. This vector is binarised by threshold δ during the algorithm and a binary mask $m \in 0, 1^n$ is formed.

$$m_i = \begin{cases} 1, & \text{if } z_i > \delta \\ 0, & \text{otherwise} \end{cases} \quad \text{for } i = 1, \ldots, n$$

Each particle has its own velocity vector $v \in \mathbb{R}^n$ and updates its position based on the best personal and best global position in the swarm. The rule for updating the velocity and position per iteration t is as follows[1]:

$$v^{(t+1)} = w \cdot v^t + c_1 \cdot \left(p^{\text{best}} - z^t\right) + c_2 \cdot r_2 \cdot \left(g^{\text{best}} - z^t\right)$$

$$z^{(t+1)} = z^t + v^{(t+1)}$$

where:

- w - is the inertia weight, which controls how much the particle keeps moving in the same direction as before. The higher w the faster the particle moves.
- c_1 and c_2 are the acceleration coefficients. They determine how strongly the particle is pulled toward its own best-known position and the global one. These control the balance between individual cognitive learning and group behaviour social learning.
- r_1 and r_2 are random values sampled from a uniform distribution $U(0,1)$. They add randomness to the movement and help avoid getting stuck in local minima.[13]
- p^{best} refers to the personal best position where it had the highest accuracy.
- g^{best} refers to the global best position of a swarm found in the current iteration.

After each update, the vectors are converted to binary values according to the δ threshold. As a result, a mask m is formed, which determines which model weights remain and which are removed.
Pseudocode for PSO:

Input: N – number of particles, T – number of iterations,
w – inertia coefficient, c_1, c_2 – cognitive and social coefficients,
δ – threshold for binary mask, $\mathcal{L}(m)$ – fitness function,
D_{val} – validation loader,

Initialize N particles with random real-valued positions $z_i \in [0,1]^n$
Initialize velocities $v_i \leftarrow 0$ for each particle
Initialize $personal\_best \leftarrow z_i$ for each particle
for each particle i **do**
    Generate binary mask m_i from z_i using threshold δ
    Load model
    Apply mask m_i and evaluate fitness f_i
    Store f_i in $personal\_best\_scores[i]$
    if $f_i > global\_best\_score$ **then**:
        $global\_best \leftarrow z_i$
        $global\_best\_score \leftarrow f_i$
    end if
end for
for $t = 1$ to T **do**
    for each particle i **do**
        Sample $r_1, r_2 \sim U(0,1)$
        Update velocity $v_i \leftarrow w \cdot v_i + c_1 \cdot r_1 \cdot (personal\_best[i] - z_i) + c_2 \cdot r_2 \cdot (global\_best - z_i)$
        Update position: $z_i \leftarrow z_i + v_i$
        Clip z_i to $[0,1]$
        Generate binary mask $m_i \leftarrow (z_i > \delta)$
        Load model
        Apply m_i and evaluate fitness f
        if $f > personal\_best\_scores[i]$ **then**
            $personal\_best[i] \leftarrow z_i$
            $personal\_best\_scores[i] \leftarrow f$
        end if
        if $f > global\_best\_score$ **then**
            $global\_best \leftarrow z_i$

29: $global\_best\_score \leftarrow f$
30: **end if**
31: **end for**
32: Save best accuracy from model pruned with $global\_best$ to history
33: **end for**
34: **return** binary mask derived from $global\_best$ using threshold δ

Let us examine the advantages and disadvantages of this algorithm.
PSO, like GA, does not require the gradient or knowledge of the structure of the model[3]. This makes it a flexible method that can be applied to any neural network. Another advantage is that PSO uses real-valued vectors, making it suitable for generating soft pruning masks that can be later binarized[13]. This soft representation makes it easier to control the sparsity of the network with the threshold. Because particles adjust their position based on personal and global information, the algorithm is less likely to get stuck in local minima[1, 27].
Although the algorithm has many advantages, it also has some limitations. It is difficult to control the sparsity or to know in advance how long the model will be fitted[29] since the result depends on the distribution of the vector. Also, since a binary mask is used, it is difficult to optimize accurately, because we do not store the exact parameters in the mask. In the case of large models with a large number of particles, the algorithm will be time-consuming[1] and in the case of a small population, it will not achieve good results. Also, the metric on which the particles are oriented plays an important role in this algorithm, and sometimes it is difficult to choose its value. If the hyperparameters are not chosen correctly, the algorithm will not work stably. Particles may get stuck in local minima or move too chaotically[31].

3.2.3 L_2 pruning

L_2 pruning is a method of structural pruning. We implement this method as a baseline approach for comparison with metaheuristic methods. It is one of the most popular methods for pruning of neural networks. This technique is based on the hypothesis that weights with lower L_2 norms contribute less to the model's prediction, so they can be removed without significant loss of accuracy. The L_2

or Euclidean norm of the vector is calculated using the following formula:

$$\|W\|_2 = \sqrt{\sum_{i=1}^{n} W_i^2}$$

where W_i - individual weights within the filter.
In the connected (linear) layer, the weight of each neuron is represented as a vector:

$$\|\mathbf{w}_i\|_2 = \sqrt{\sum_{j=1}^{n} w_{i,j}^2}$$

where $w_{i,j}$ is j-th weight of the i-th neuron.
In the convolutional layer (Conv2d), each filter is represented by a three-dimensional tensor, and the L_2 norm of the filter is calculated using the following formula:

$$\|W_i\|_2 = \sqrt{\sum_{c=1}^{C_{\text{in}}} \sum_{u=1}^{k} \sum_{v=1}^{k} W_{i,c,u,v}^2}$$

where C_{in} is the number of input channels and $k \times k$ is the special size the filter
The main stages of L_2 pruning are as follows.

1. The model is trained on the full data set.
2. The L_2 norm is calculated for all filters.
3. A pre-selected percentage of filters with the lowest norm are removed.
4. Fine-tuning for additional training of the model.

L_2 is a structural method, so the whole filters are removed at once, as already mentioned, so this reduces the number of parameters and computational costs, unlike GA and PSO, this method does not have iterations, so it is much faster[7].
The advantages of L_2 pruning include ease of use, as you only need to calculate the L_2 norm for each filter or node[19]. Also, this method immediately drops the filters and this reduces the size of the model and computational costs.
The disadvantages of this method are that it does not take into account how the filter affects the output of the mesh and how the filter interacts with other layers[23], so there is a chance that important filters are removed. Fine-tuning

is always required after pruning, as the method has a significant impact on the model. Also, determining the percentage of model removal in advance may not be optimal.

4 Experiments and results

4.1 Dataset Description (MNIST)

For this study, we chose the MNIST dataset, which consists of 70 thousand images of handwritten digits. The dataset is divided into 60 thousands data used for model training and 10 thousand for testing. Each image is represented in grayscale with a size of 28×28 pixels, as in the Figure 2, resized to 32×32 pixels before being passed into the model. MNIST was chosen for this project because of its convenience, a small amount of data, which allows for quick testing of different model configurations, and easy evaluation of the impact of pruning on accuracy.

Figure 2: Example of MNIST dataset

4.2 Neural Network Architecture LeNet5

In this study, we consider the basic architecture of the LeNet-5 convolutional neural network, which was first introduced by Yann LeCun in 1998[16]. This model is often used as a standard example in machine learning research and for testing new approaches[9]. It has a well-defined structure and relatively low depth, which makes it easier to experiment with. As shown in the Figure 3 the model includes 7 layers, excluding the input layer. The input is a 32×32 pixel image. Next comes the first convolutional layer, which contains 6 filters of size 5×5, followed by a subsampling (also known as pooling) layer. This is followed by a second convolutional layer with 16 filters of size 10×10 and another pooling layer. Finally, the network has two fully connected layers C_5 and $F6$ and a softmax output layer with 10 neurons, corresponding to 10 classes.

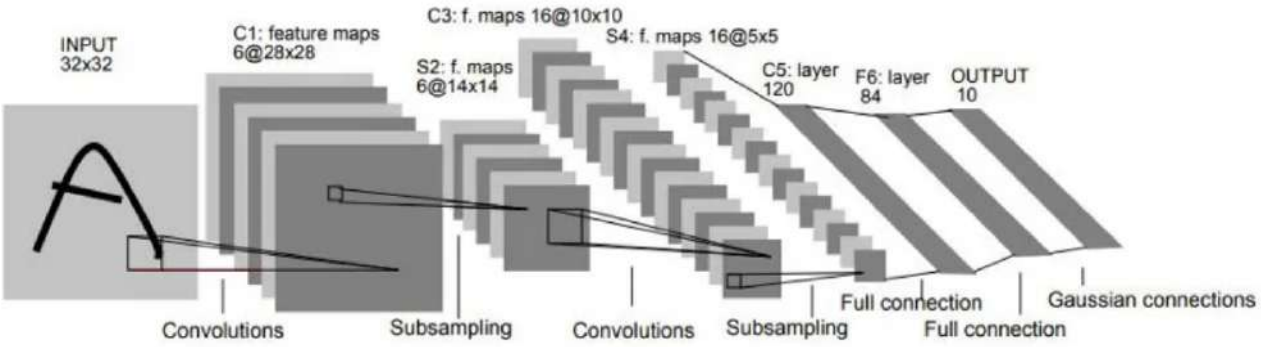


Figure 3: LeNet-5 Architecture [2]

4.3 Baseline Model Training Process

The LeNet-5 model was trained without pruning on the MNIST dataset for 5 epochs. During training, Adam optimizers were used with a speed of 0.001 and a batch size of 64. The categorical cross-entropy loss function was also used to evaluate the accuracy. After each epoch, the accuracy was measured to track model performance. The input images were normalized to the range [0,1] and resized to 32×32 pixels to match the expected input shape of LeNet-5. After five epochs, the model achieved an accuracy of 0.98 on a validation set. This version of the model was saved and reused in the following experiments.

4.4 Fitness Function

In the implementation of both Genetic Algorithm and Particle Swarm Optimization we use the fitness function to evaluate the binary masks. This function combines two objectives: model accuracy and sparsity. The function was used in the following form:

$$\text{fitness} = \lambda_1 \cdot \text{accuracy} + \lambda_2 \cdot \text{sparsity}$$

where:

- $\lambda_1 = 0.7$ to prioritize precision,
- $\lambda_2 = 0.3$ to consider sparsity.

These values were chosen by the observation of the initial experiments in order to find a good balance: we want the model to stay accurate but also reduce its size.

Giving more weight to accuracy ensures we are not losing too much performance, while still making the algorithm to shrunk the model.

4.5 Description of Pruning Methods and Their Parameters

4.5.1 Genetic Algorithm Pruning

In this experiment, a Genetic Algorithm was used for pruning, where each individual in the population is a binary mask that specifies which weights should be kept and which can be zeroed out. The size of the mask matched the total number of parameters in the model, in our case 61706. To find the best configuration, a grid search was performed on four hyperparameters, namely the number of generations, population size, number of elite individuals, and mutation probability. The total number of configurations was $3 \times 3 \times 2 \times 3 = 54$. After training the baseline model for five epochs, each pruning configuration was applied to the trained model, and its accuracy was evaluated on the validation dataset. The masks that achieved the best models were saved in a separate file, and the results of the experiment were saved to the dataset for further analysis and visualization. After the pruning, the accuracy of the best configuration reached 0.933 on a validation set and had a sparsity level of 49.76%.
The highest accuracy was achieved with the following parameters:

- number of generations: 15
- population size: 10
- number of elite individuals: 3
- mutation rate: 0.01

A pruned model often loses its accuracy as the weights are removed completely because the model loses some of its learning ability. To fix this, a short retraining of the model, namely fine-tuning, was performed after pruning. In our case, we retrained the model for three epochs with the same parameters as in the main training phase, namely the Adam optimizer and a learning rate of 0.001. This stage allows the model to adapt to the simplified structure and restore the lost accuracy, fully or partially. After three epochs, the fine-tuning gave a significant

improvement in the results, the accuracy increased to 0.982.
In the Figure 4, you can see how the accuracy changed at each generation for the best parameter configuration. In the first generations, the accuracy fluctuated greatly. After step 5, we can see how the model started to grow. We can assume that the model in the first generations eliminates less successful masks and combines the best solutions, and mutations in these steps have a strong negative impact.

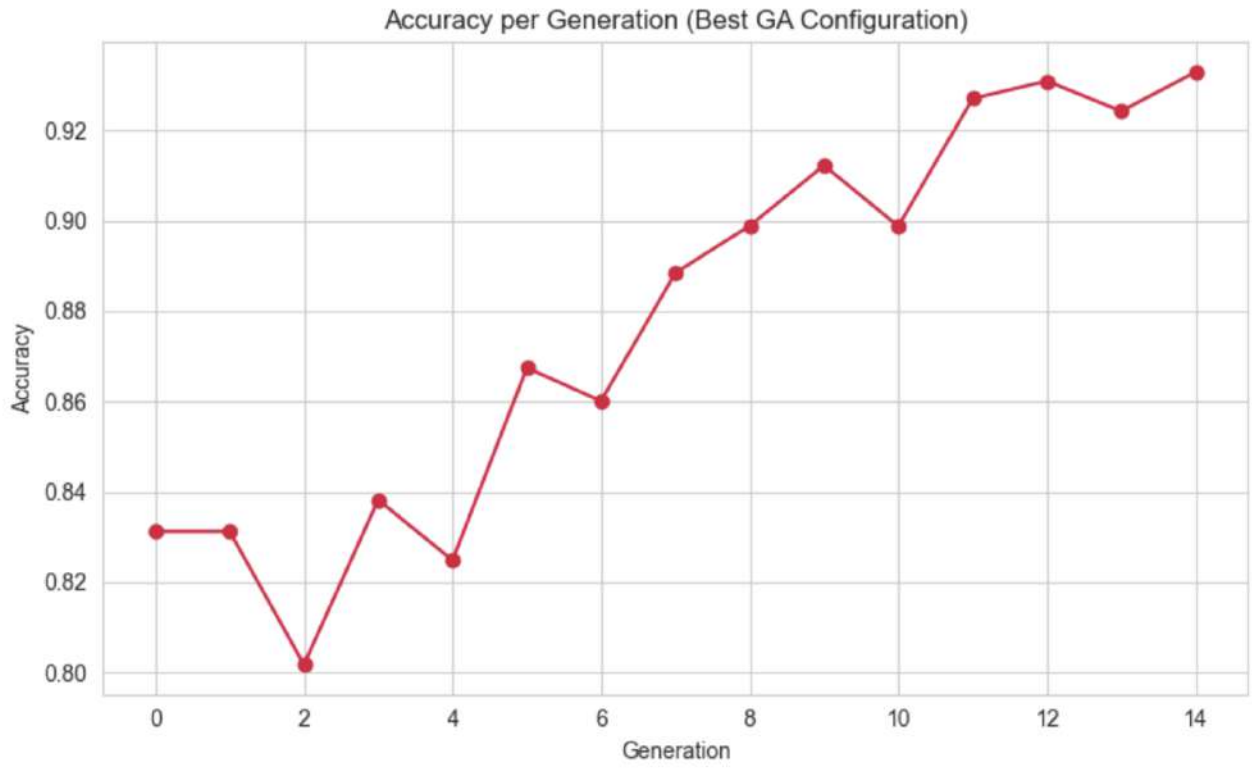


Figure 4: Accuracy per Generation for GA (Best Configuration)

The heat map (Figure 5) shows how the model's accuracy depends on the number of generations and population size. Each cell shows the average accuracy value. It can be seen that the model has the highest average accuracy of 0.933 with a population of 10 and 15 generations, just like in the best configuration. The model with 10 generations and 20 individuals also shows a fairly good value. At the same time, the lowest accuracy of 0.85 is shown by the model with a small number of generations and populations. We can assume that the algorithm does not have time to find an optimized mask yet.

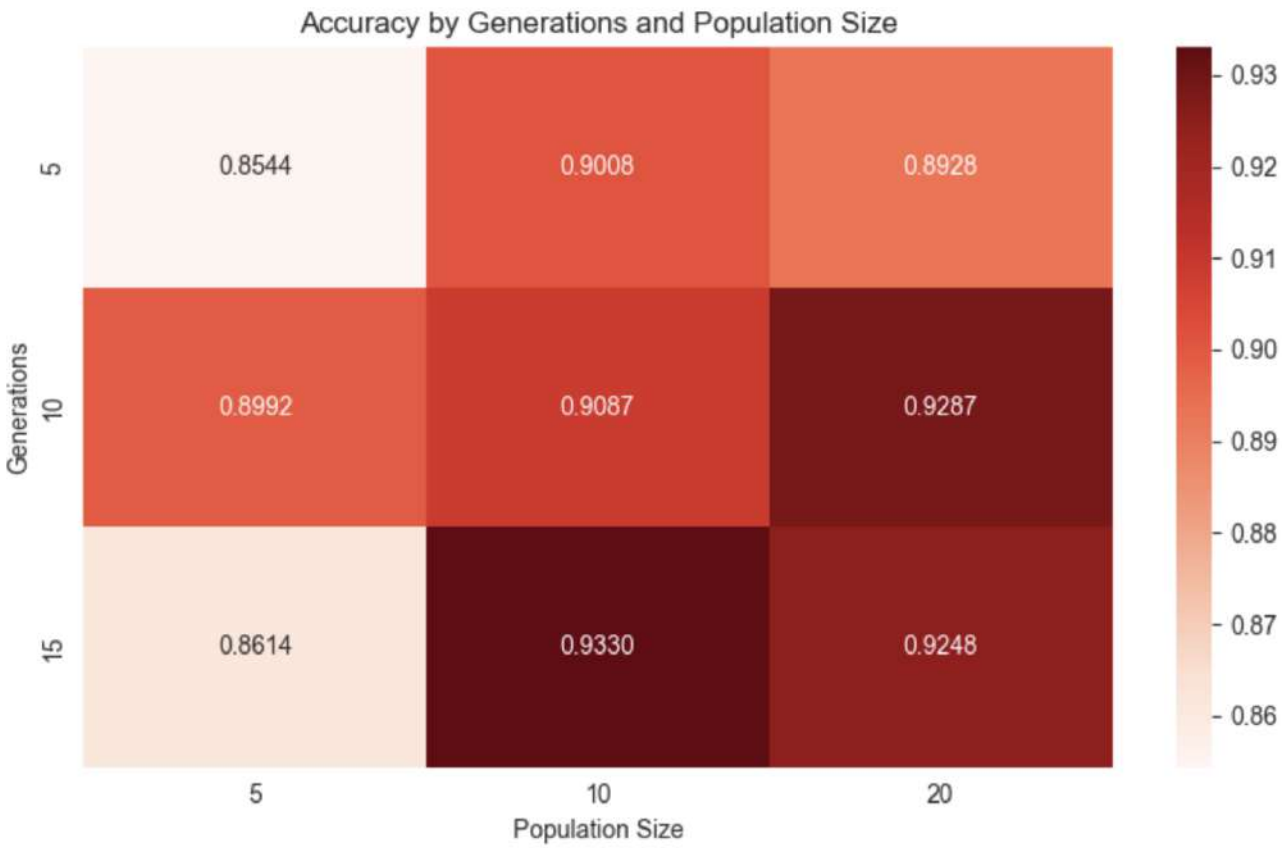


Figure 5: Accuracy by Generations and Population size

The Figure 6 shows the relationship between accuracy and sparsity for different configurations of the Genetic Algorithm. From this graph, we can see that with both values of elite size, it is possible to achieve an accuracy of more than 0.9, but the lowest values of accuracy and sparsity are obtained with elite size = 3. As for the mutation rate, the value of 0.1 has quite a few points with high accuracy and sparsity. The best results are obtained with 0.05 and 0.01 mutation rates. The best configuration is a confirmation of this statement and has a mutation with a probability of 1%.

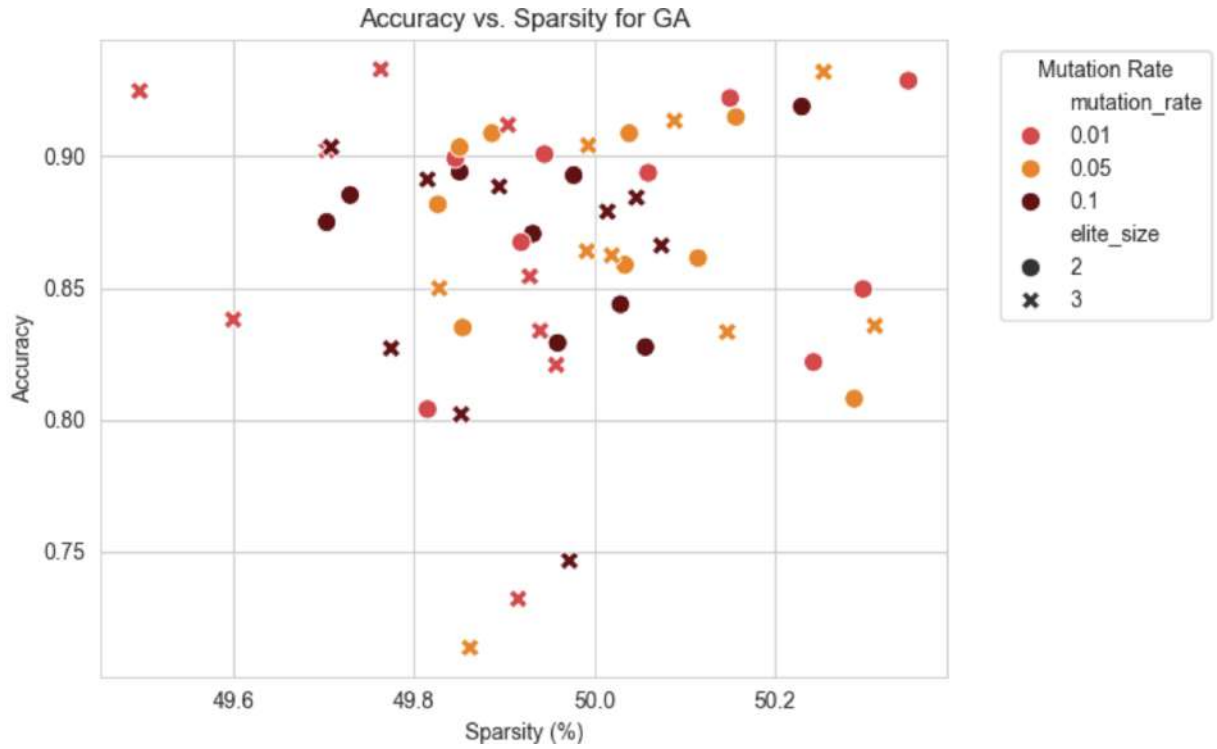


Figure 6: Accuracy and Sparsity by Mutation Rate and Elite size of GA

4.5.2 Particle Swarm Optimization Pruning

In this experiment, we implemented the Particle Swarm Optimization algorithm, which is described theoretically in Section 3.2.2. Each particle represents a vector of real numbers of length equal to the number of parameters in the model. The algorithm zeroes some of the values in the mask, according to a threshold. In this work the threshold is fixed at $\delta = 0.5$, assuming that values above 0.5 indicate important parameters to retain and lower values correspond to less important ones. To select the best configuration of PSO parameters, a grid search was performed on 243 configurations ($3 \times 3 \times 3 \times 3 \times 3 = 243$). The hyperparameters used for this purpose were: the number of particles, number of iterations, inertial weight, and cognitive and social components. For each of the configurations, the accuracy was evaluated after applying the mask to a pre-trained LeNet-5 model, the same one used for the Genetic Algorithm experiment. The best configuration was obtained with the following values:

- number of particles: 15
- number of iterations: 15

- w: 0.5
- c_1: 1
- c_2: 1

This configuration provides a post-pruning accuracy of 0.942 and a sparsity of 50.37%. After obtaining the best mask, it was used to fine-tune the model for three additional epochs with the same optimizer as before, which increased the model's accuracy to 0.985. The accuracy was also maintained for each configuration across iterations. The Figure 7 shows this for the best configuration. You can see how the line is going up, but sometimes it takes more iterations to find a better mask, which is expected since the weights are changed based on the best values.

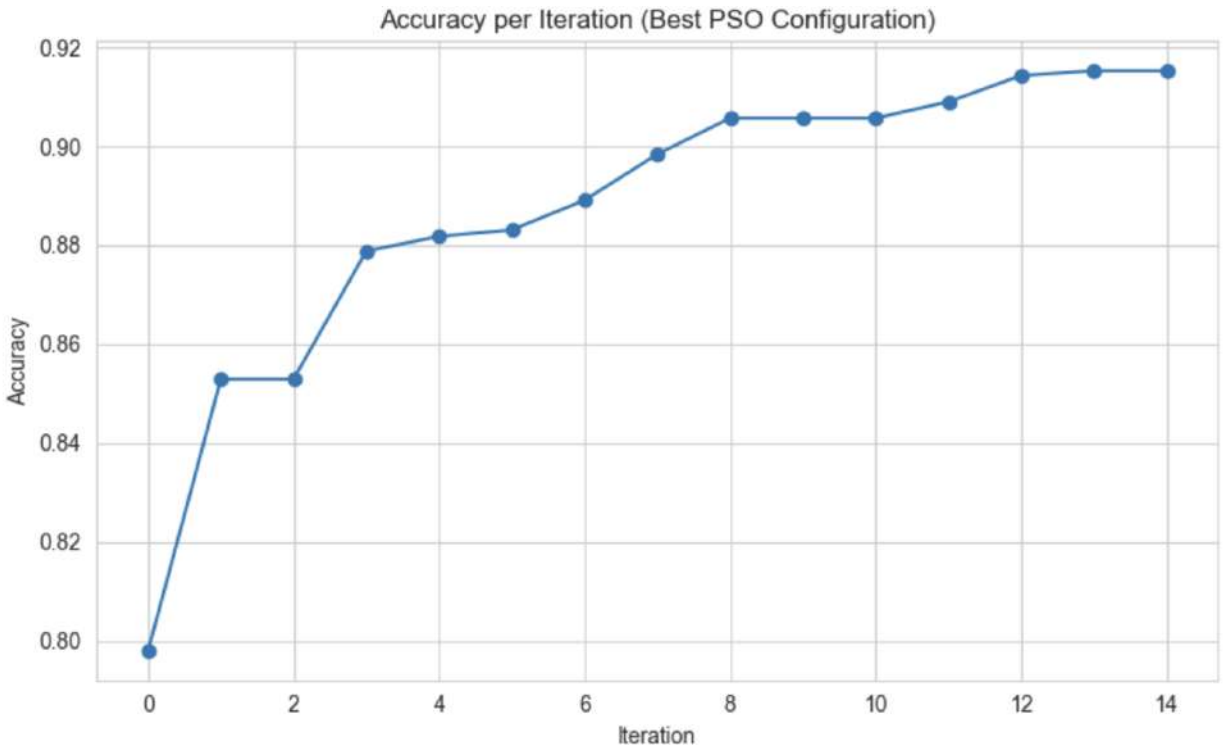


Figure 7: Accuracy per Iteration for best configuration in PSO

The Figure 8 is a heat map that shows the average accuracy of the model depending on the number of particles and iterations. We can see that the best result of 0.9416 is obtained with 15 iterations and 15 particles, just like in our best configuration. There is a clear correlation observed more particles and iterations

lead to better accuracy.

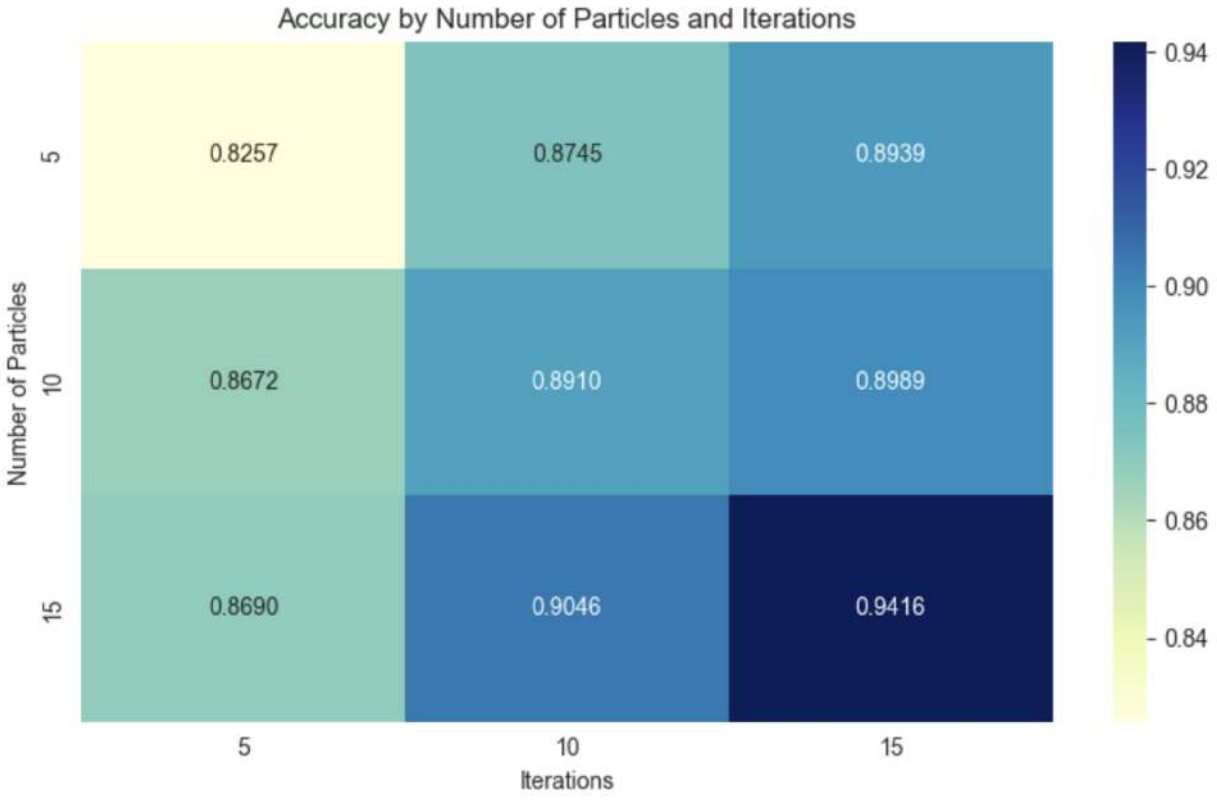


Figure 8: Accuracy by Number of Particles and Iterations

The Figure 9 shows the dependence of model accuracy and sparsity on the parameters inertia weight and number of particles using the PSO algorithm. The colour indicates different values of inertia weight w, and the shape marker corresponds to the number of particles. It can be seen that the most scattered values are at $w = 0.7$, and the two best values for accuracy and sparsity are at $w = 0.3$ and $w = 0.7$, and 15 particles. But in general, the values are quite randomly distributed. Meanwhile, the sparsity does not change much, in the range of one percent, while the accuracy varies greatly depending on the configuration.

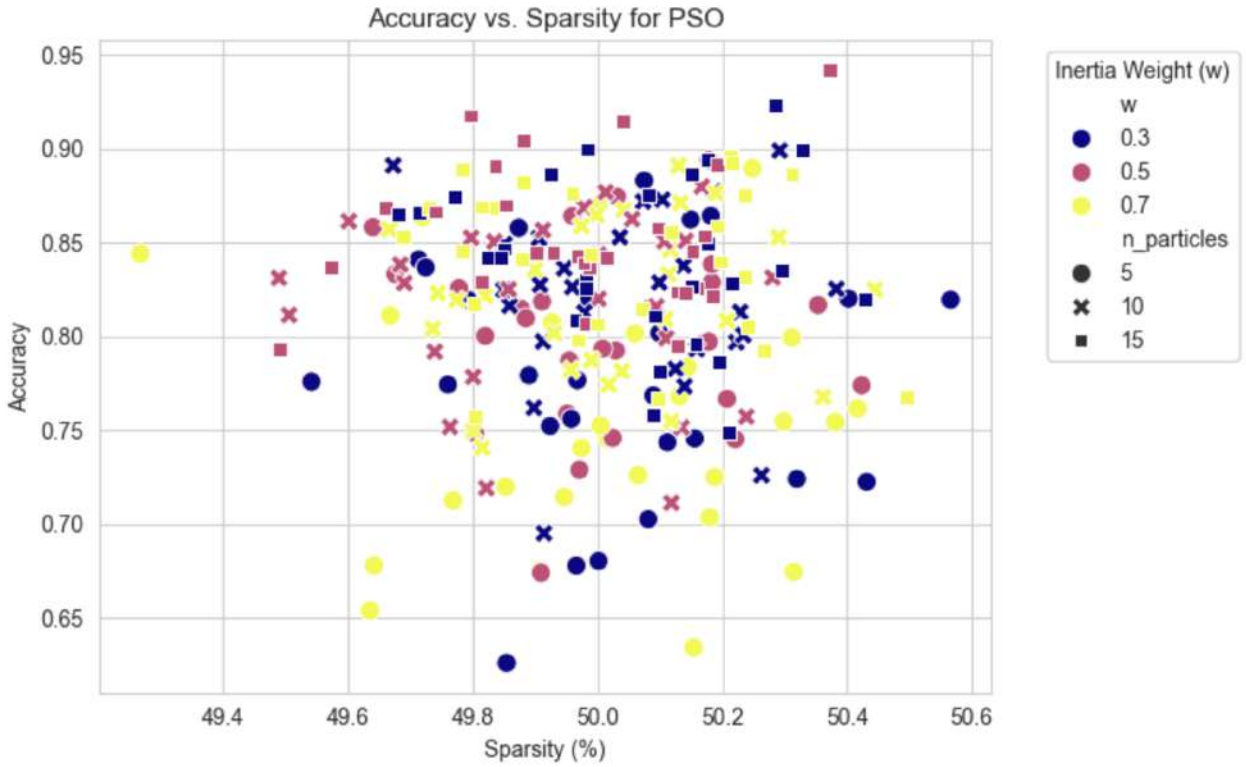


Figure 9: Accuracy and Sparsity by Inertia weight and Number of particles for PSO

4.5.3 L_2 Structured Pruning

In this experiment, we applied structural pruning based on the L_2 norm to compare the results with metaheuristic methods. The pruning was applied to each Conv2d and Linear layer where the number of output channels was greater than one. In each of these layers, 50 percent of the output channels with the lowest L_2 norm were removed. To do this, we used the *ln_structured* method from the PyTorch library with the parameters $n = 2$ and $dim = 0$. After that, the *prune.remove* function was executed. Before pruning, the accuracy of the trained model was 98.03%, but after removing 49.81% of the model parameters, it became 57.3%. Since pruning greatly reduced the precision of the model, fine-tuning was applied for three epochs with the same Adam optimizer and a learning rate of 0.001. As a result, the accuracy increased to 0.984, demonstrating the ability to restore the model's precision to a high value.

In the Figure 10, you can visually see the change in accuracy before and after pruning, as well as after fine-tuning.

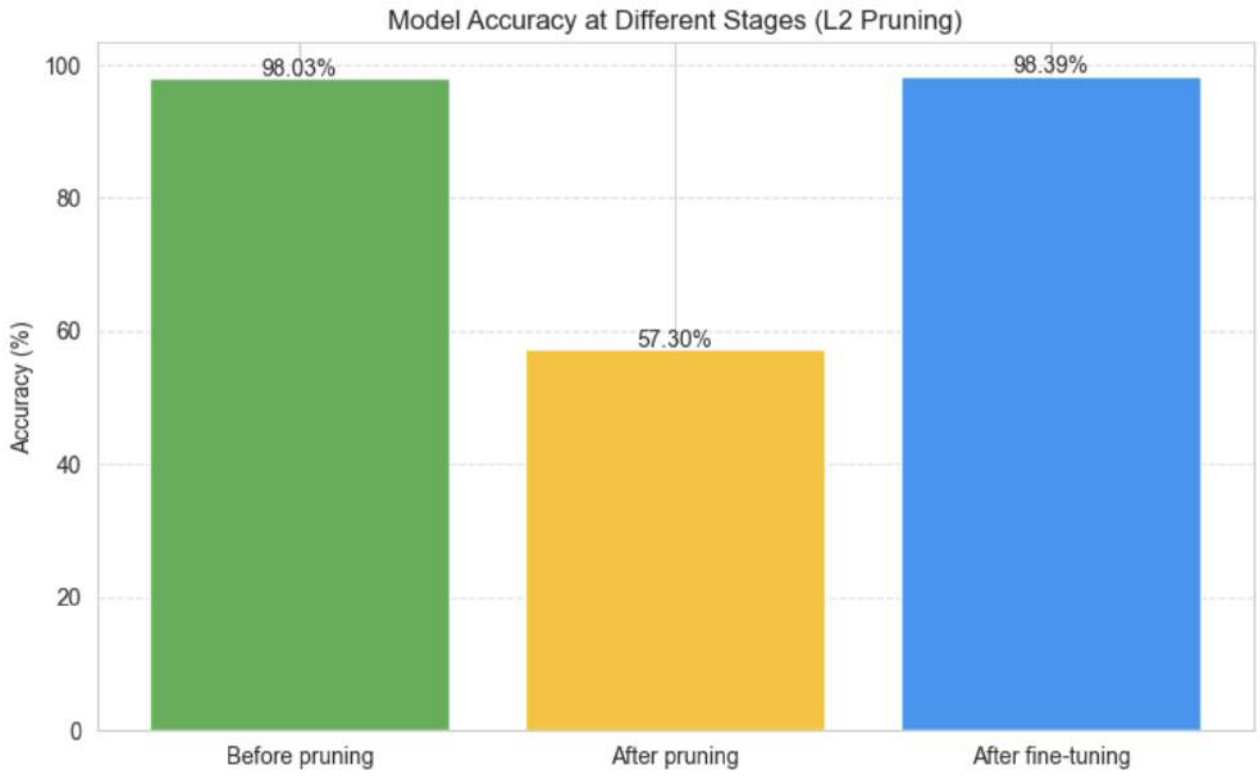


Figure 10: Accuracy comparison for L_2 pruning

4.6 Comparative Analysis

In this subsection, we will compare the algorithms considered in the study: Genetic Algorithm, Particle Swarm Optimization, and structural L_2 pruning. Table 1 summarizes the best results of our experiments.

Method	**Accuracy before pruning**	**Accuracy after pruning**	**Accuracy after fine-tuning**	**Sparsity**	**Time of pruning (Sec)**
GA	98.03%	93.3%	98.19%	49.76%	78.32
PSO	98.03%	94.16%	98.48%	50.37%	123.95
L_2	98.03%	57.3%	98.39%	49.81%	0

Table 1: Comparison of methods

In terms of accuracy, both metaheuristic methods showed good values immediately after pruning the model, which is not the case with the L_2 method. The

GA achieved an accuracy of 0.93 and the PSO 0.94, which demonstrates already good results considering that the model in both cases was reduced by half. The L_2 method also reduced the model by half, but the accuracy dropped to 0.57, so in this case fine-tuning was extremely necessary.

For both GA and PSO, fine-tuning increased the accuracy to about 0.98. In the L_2 pruning, it played an important role by increasing the accuracy of the model to 0.984, which is even slightly better than the pre-pruning result of 0.98.

Regarding sparsity, all three methods reached approximately the same level, around 50%, which allows for an objective comparison of the three methods.

In terms of time, L_2 pruning showed the best result, as it took less than a second. This method does not require iterations and parameter searches. In turn, GA and PSO took 78 and 124 seconds, respectively, due to the use of more computing resources. It should also be borne in mind that grid search, which searches for the best model, also takes some time and requires more resources.

5 Conclusions

The study describes how the metaheuristic methods namely Genetic Algorithm and Particle Swarm Optimization for pruning a neural network alongside the commonly used L_2 norm pruning. These algorithms were implemented in the Python programming language using the PyTorch library to implement the L_2 pruning. All three methods were implemented for use on the convolutional neural network LeNet-5, which classifies images on the MNIST dataset. Their performance was evaluated and compared in terms of accuracy, pruning time, and sparsity. The impact of hyperparameters on metaheuristic methods was also examined.

The results show that metaheuristic methods show competitive results compared to L_2 pruning after fine-tuning. Additionally, GA and PSO demonstrate significantly higher accuracy immediately after pruning, which makes them especially valuable in cases where fine-tuning is not possible, for instance, when training data is unavailable due to legacy, ownership, or data privacy topics. We remark however that this advantage comes with an additional computational cost. Future directions of research might include pruning of other types of architectures, such as Vision Transformer, application of the metaheuristic algorithms specialized in high-dimensional problems, and stability analysis.

References

[1] Abdullah Al-Shaikh, Fawaz Ahmed, Farhan Saeed, and Osama Mohamed. Model pruning using evolutionary algorithms: A review of current trends. *Applied Sciences*, 2020.

[2] Siddhesh Bansod. Lenet-5 architecture explained, 2020. Accessed: 2025-05-24.

[3] Chenghao Chang, Wenhao Xie, Ming Liu, and Xuefeng Liang. Acp: Automatic channel pruning via clustering and swarm intelligence optimization for cnn. *arXiv preprint arXiv:2101.06407*, 2021.

[4] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[5] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks, 2019.

[6] Spencer Gibb, Hung Manh La, and Sushil Louis. A genetic algorithm for convolutional network structure optimization for concrete crack detection. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2018.

[7] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural networks. In *Advances in Neural Information Processing Systems*, 2015.

[8] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks, 2015.

[9] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, 2015.

[10] Edwin R. Hancock. Pruning neural nets by genetic algorithm. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*. Springer, 1997.

[11] Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. pages 164–171, 1993.

[12] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures, 2016.

[13] Weiguang Huang, Jun Song, Zhi Liu, Kai Xiang, Xiangyang He, and Jian He. Psopruner: A layer-wise pso-based pruning method for cnns in pv module defect classification. *IEEE Journal of Photovoltaics*, (5), 2022.

[14] Yaochu Jin and Bernhard Sendhoff. Pareto-based multiobjective machine learning: An overview and case studies. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(3):397–415, 2008.

[15] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*, 1995.

[16] Yann Le Cun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient based learning applied to document recognition. *Proceedings of IEEE*, 86(11):2278–2324, 1998.

[17] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. pages 598–605, 1989.

[18] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Snip: Single-shot network pruning based on connection sensitivity, 2019.

[19] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations (ICLR)*, 2017.

[20] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pages 2736–2744, 2017.

[21] Sushilata D. Mayanglambam, Shi-Jinn Horng, and Rajendra Pamula. Pso clustering and pruning-based knn for outlier detection. *Soft Computing*, 2023.

[22] Seyedali Mirjalili. Genetic algorithm. In *Evolutionary Algorithms and Neural Networks*. Springer, 2019.

[23] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Importance estimation for neural network pruning. In *CVPR*, 2019.

[24] Luis Miguel Rios and Nikolaos V Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3), 2013.

[25] Luis Miguel Rios and Nikolaos V Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3), 2013.

[26] Juanjuan Tu, Yongzhao Zhan, and Fei Han. A neural network pruning method optimized with pso algorithm. In *2010 Second International Conference on Computer Modeling and Simulation (ICCMS)*. IEEE, 2010.

[27] Marnus van Zyl and Andries P. Engelbrecht. Set-based particle swarm optimisation: A review. *Mathematics*, 11(13):2980, 2023.

[28] Thomas Weise. *Global Optimization Algorithms - Theory and Application*. 2009.

[29] Xin-She Yang. *Nature-Inspired Optimization Algorithms*. Elsevier, 2014.

[30] Yibo Yang and Cong Deng. Multi-objective pruning for cnns using genetic algorithm. *arXiv preprint arXiv:1906.00399*, 2019.

[31] Y. Zhou, X. Sun, F. Li, et al. An adaptive particle swarm optimization-based hybrid long short-term memory model. *Soft Computing*, 26(7):3343–3359, 2022.