Bachelor thesis
Computing Science

Radboud University

Combining Knowledge From Pre-trained Networks And Weight Agnosticism For Evolving Transferable Models

Author:
Nadezhda Dobreva
s1033115

First supervisor/assessor:
Dr. Prof. Elena Marchiori
elena.marchiori@ru.nl

Second assessor:
Dr. Tom Claassen
tomc@cs.ru.nl

March 25, 2022

Abstract

Gaier and Ha [11] introduced the evolution of weight-agnostic neural networks (WANNs) which can perform a task without the need for weight training. They also suggested the possibility of evolving a WANN with high generalization ability across multiple domains. In this thesis we determine whether the baseline WANN evolved as a classifier of handwritten digits is already capable of performing more than one task. Furthermore, we propose a new approach based on evolutionary algorithms that combines the original WANN method with the knowledge of pre-trained networks to achieve higher generalizability. Our algorithm starts from a pre-trained model, keeping only the sign of its weights, and evolves neural networks by repeatedly pruning them. To ensure the increased performance on multiple tasks, the fitness measure is calculated using joint evaluation.

We report a boost in generalization capability over multiple tasks: The accuracy on a classification task the original model was not trained on increases more than 3 times! However, additional techniques (such as fine-tuning) are required to achieve high accuracy on more than one task. The results indicate that our method produces networks that are significantly better than the baseline WANN but still not good enough for performing multiple tasks. Moreover, we conduct an ablation study to investigate the contribution of our algorithm's components. It shows that weight agnosticism and joint evaluation contribute to maintaining a balance between the performances on the tasks. The compression achieved of our resulting model also suggests the possibility of using our algorithm as a pruning technique.

Keywords: Machine Learning · Genetic Evolution · Weight Agnosticism · Network Pruning · Generalization Ability

Contents

Chapter 1

Introduction

Neural Networks (NNs) find application in a myriad of problems and fields such as image classification, natural language processing, content generation, etc. However, training the weights and achieving the optimal model for a task is a very power-, time- and memory-consuming process.

In their research, Gaier and Ha [11] evolve weight-agnostic neural networks (WANNs) in which the architecture, not the weights, encodes the information needed for performing a task. They use genetic evolution to search for the optimal structure by adding or removing neurons and connections. The networks are evaluated by assigning a shared weight value to the edges and measuring the accuracy. The research shows a WANN classifier that has 91% accuracy on the MNIST dataset.

The approach in [11] avoids the expensive weight-training process, significantly lowers the space required to store a network and results in a model with a strong inductive bias that is easy to fine-tune into even better performance by e.g. using transfer learning. An alternative method to overcome the aforementioned issues is using the knowledge from a pre-trained neural network. For image classification tasks in particular, many neural network architectures pre-trained on datasets such as ImageNet are publicly available and used in practice.

Gaier and Ha suggest the possibility of evolving a WANN in a way that enables using it for a multitude of tasks. This is the starting point of our research. Although related to multi-task learning, our goal is to explore whether weight-agnostic networks are more generalizable than models obtained by a standard training procedure, and how to improve the baseline WANN.

Firstly, we investigate whether WANNs generalize better to related domains than pre-trained neural networks. Specifically, we are interested if a WANN evolved for classifying the MNIST digits has a higher accuracy on a subset of the EMNIST Letters dataset than a neural network that was trained on MNIST. Therefore, we compare MNIST WANN achieved by [11] and a simple model pre-trained for MNIST classification. Moreover, we evolve another WANN for classifying both MNIST and 10 classes of EMNIST Letters to gauge whether *joint evaluation* helps increase the WANN's generalizability.

Next, we explore whether combining the WANN approach with the use of pre-trained neural networks is a more effective method for evolving a WANN than the original. Thus, we propose

"reversing" the algorithm: Start from a pre-trained network and, retaining only information on the sign of its weights, search for a "good" sub-network using a genetic algorithm (GA) that prunes the task-specific parts of its structure. The fitness of the GA evaluates the performance of a network in a weight-agnostic way by jointly measuring its accuracy on two classification tasks (MNIST and 10 classes of EMNIST Letters). The resulting network we call Pruning-Attained Domain- And Weight-Agnostic Neural Network (PADAWANN).

Finally, since our proposed algorithm is related to network compression via pruning, we also compare its performance with standard pruning techniques.

We perform an extensive set of experiments and comparative analysis on WANN, WANN with joint task evaluation, PADAWANN and pruning strategies. Furthermore, an ablation study on PADAWANN is conducted to assess the added value of each of its components. Our results can be summarized as follow:

- Both the WANN model evolved using MNIST and the one evolved using joint evaluation do not generalize well to more than one task: They have high accuracy on one dataset and random-guessing-level accuracy on the other. This indicates that weight-agnostic NNs do not directly generalize to "related" tasks.
- Our PADAWANN has higher performance than WANN on both datasets, showing that combining pre-trained networks and weight-agnostic evolution is beneficial. However, this approach also does not produce highly generalizable NNs. Furthermore, that generalizability extends only to the tasks used in the evolution of the model.
- PADAWANN has 3.3x higher accuracy on the 10-letter data than the NN pre-trained on MNIST and lower accuracy on MNIST. But when considering as fitness only the accuracy on MNIST, PADAWANN almost matches the pre-trained network's performance.
- The model pre-trained on MNIST also performs better on MNIST than WANN and models obtained using network pruning. However, WANN, PADAWANN and the models created via pruning are much smaller than the pre-trained network. Note that WANN does not use information about the pre-trained model since it grows an architecture starting from a very small one.
- Although PADAWANN's accuracy on MNIST is 5% lower than models obtained by L1-norm pruning, it is 13% higher than the MNIST accuracy of randomly pruned models.

Our results indicate that the ability to perform multiple tasks of weight-agnostic networks is not achievable using the baseline approach [11]. We show an improvement in generalization capability by combining weight agnosticism with knowledge from pre-trained networks which scales well when more than two tasks are used. However, even when data from multiple tasks is used to guide the search, finding an NN architecture with high generalization ability remains a challenging task. It is important to note that our conclusions are within the specific proof of concept experiment using MNIST and a subset of EMNIST Letters.

The rest of this paper is organized as follows: First, in Chapter 2 we introduce the core concepts and theory that we build on in this thesis. Chapter 3 shows what methods other researchers have already tried and tested. Chapters 4 through 6 are dedicated to the experiments we conduct and their results. Lastly, a discussion of our findings and directions for further work are presented in Chapter 7, and our conclusions are summarized in Chapter 8.

Chapter 2

Preliminaries

In this chapter we introduce the fundamental concepts and terminology this thesis works with and builds on. We focus on Neural Networks (NNs), network pruning, network architecture search (NAS) and genetic algorithms (GAs).

2.1 Neural Networks

Artificial Neural Networks (ANNs, or simply NNs) are computer systems inspired by biological neural networks, and more specifically by the way neurons in the brain receive, process and transmit signals to each other via their synapses. NNs mimic this behaviour using "nodes" connected by edges. The transformation happening at each artificial neuron is dictated by an *activation function* which defines how the weighted sum of the inputs is turned into an output, and helps the network learn complex patterns. The artificial neurons are usually aggregated into multiple levels. When that is the case, the net is called a Deep Neural Network.

2.1.1 Basic Functionality

The input neurons in the first layer receive the "raw" input signal. They process and possibly transform it and then send it to the neurons of the next layer, which also process and pass on the signal. This procedure repeats until it reaches the last layer, where it is transformed one last time by the activation functions of the output neurons that generate the NN's result. The *weight value* assigned to an edge defines how important a connection is. Thus, while the signal is being transmitted between neurons, those weights can increase or decrease its strength. In other words, the weight values encode most of the information needed for the network to perform its task. The structure of the network also contributes to its performance, but we will go into more details on this later.

This process of taking in an external input, processing it, and producing output is called forward propagation. Most often training an NN consists of letting the network do the task in question, evaluating its performance, and then performing the so called "back propagation." This operation involves going backwards from the output layer towards the input layer and adjusting the weights of the edges according to the calculated error information. These few steps are repeated until the error is minimal. A set of training data is used for learning the

best values of the weights, while the evaluation is done on a previously unseen test set.

One measure of the NN's performance is *accuracy* – the portion of correctly classified samples. We can compare the performance of two classifiers by applying a test of significance. The statistical McNemar test [28] is commonly used for this purpose [9, 29]. The so called p-value is calculated based on the number of samples one classifier got wrong and the other got right. If $p < 0.05$ the idea of the two models having equal performance is rejected, making them significantly different.

An example network is presented in Figure 2.1. Its layers are fully-connected meaning that every node from one layer is connected to every node from the next. This is a characteristic of its *topology*. A neural network's topology refers to the way its neurons are connected and it is crucial for the network's performance. "Structure" and "architecture" are often used as synonyms for topology.

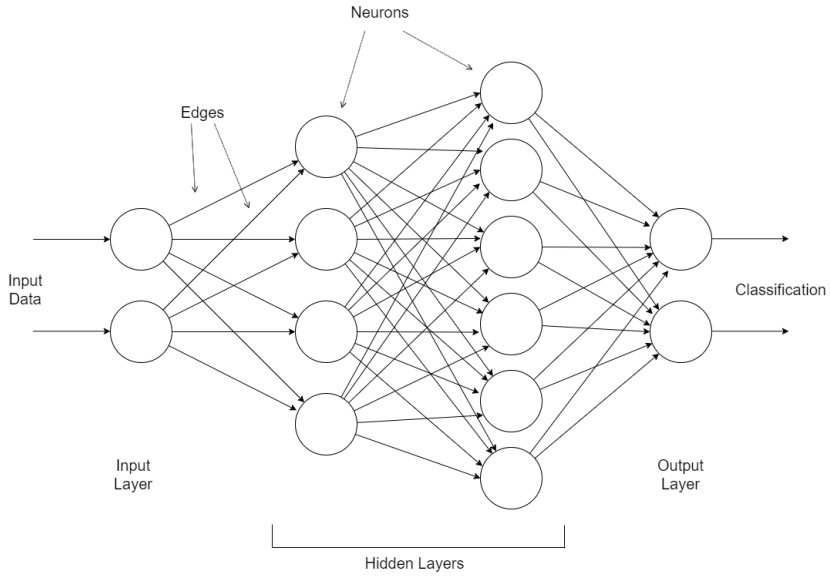


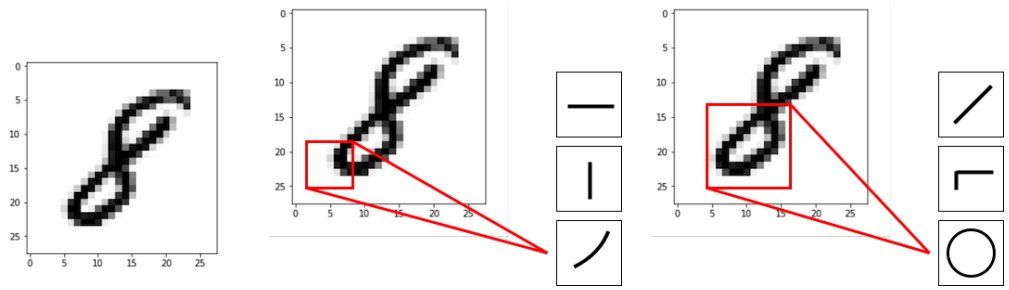
Figure 2.1: The structure of a simple deep neural network for classification.

Depending on their intended use, there are different types of neural networks. A classifier is a model whose task is to take a sample and identify which category, out of a set of possible ones, it belongs to. For instance, an NN which takes in a picture of an animal and as result decides what type of animal is depicted is a classifier. Just like in this example, we will focus solely on networks that classify images.

2.1.2 Convolutional Neural Networks

The preferred model for image classification tasks currently is the *Convolutional Neural Network* (CNN). It is a neural network that takes images as an input and learns their most important aspects and features to develop the ability to tell them apart. It is capable of capturing spatial and temporal dependencies [13][Ch.9], which is necessary for image classification.

The input of a CNN is an image (or in technical terms, data with three dimensions: height, width, number of channels that corresponds to the color scale). Each neuron processes data in a specified "receptive field" of the image. The neurons are ordered in layers in such a way that enables it to first detect simple patterns and in the subsequent layers increasingly complex ones. To illustrate, we present an example with the MNIST digit 8 (fig. 2.2a), which a CNN is trying to classify. The first layer neurons identify the patterns occurring on their own slice of the image (fig. 2.2b). The layers after that look for more complicated shapes that occur on a bigger part of the image (fig. 2.2c). Only after identifying circles in the upper and lower half of the image, the convolutional network classifies the input as class '8'.



(a) A sample from the MNIST dataset.

(b) A single neuron looking for patterns in its receptive field.

(c) Searching for more complex patterns in the image.

Figure 2.2: Processing MNIST digit 8.

This procedure is accomplished with the help of the convolutional layers which are not used in typical NNs. This layer performs the dot product between the receptive field of a neuron and a set of learn-able parameters (kernel) [13][Ch.9]. A collection of kernels is called a filter.

2.2 Neural Architecture Search

Neural Architecture Search (NAS) is a method for creating an NN's architecture automatically instead of manually by a human (which is a rather time-consuming process, prone to mistakes). Although in some cases NAS is used for finding optimal weights, usually NAS focuses on the neurons, their connections and distribution into layers. Some research deals solely with the structure and even disregards the weights completely by keeping them constant, or sampling them randomly [11, 20, 30].

Different methods have been proposed for optimizing a network's architecture. One commonly used approach for NAS is deploying algorithms that evolve the structure (such as Genetic Algorithms, which will be explained in depth in section 2.4) [4, 45, 41, 40]. Pruning (section 2.3) can also be utilized for architecture search especially when whole filters are removed [49], because the pruned architecture is more important for the model's efficacy than the leftover weights [25]. Moreover, a combination of pruning and GAs has also been considered [3, 12].

2.3 Network Pruning

Neural Networks typically have a large amount of parameters which makes them memory intensive. Furthermore, training NNs can be computationally heavy. That is because the process requires repeatedly doing 1) a forward pass with memorizing the floating point values of millions of weights and activation functions, and 2) back propagation where they are all adjusted. It also involves thousands of data samples. Because of those factors and limited hardware resources and computing power, training is a slow process for large architectures and training data.

A model is overfitting if the accuracy achieved on training samples is high, but once tested on unseen samples, its accuracy drops. This occurs when the network does not generalize well because it has introduced more parameters and is searching for more features than needed, or uses more complicated approaches to identifying a feature than are necessary [19]. Consequently, an overfitting model has poor performance when tested on unseen before samples. Not only that; because of the unnecessary additional connections, it needs more computational power and storage space. One way to mitigate this problem is reducing the complexity of the model.

Network Pruning is a method for simplifying and sparsifying a neural network by removing redundant branches. The purpose of pruning is to increase the speed of the network and decrease its storage size. Moreover, network pruning can positively affect the NN's performance. It helps the model generalize better [42, 32] and avoid overfitting. This is because the removed redundant edges are typically ones formed on the basis of extremely specific occurrences in the training data which are not representative of the data's features.

One approach, unstructured pruning, is to find the least important connections and prune them individually [16, 17, 14, 5]. A frequently utilized technique is L1-Norm unstructured pruning, where the edges with smallest absolute weights are considered "least important" and removed. Structured pruning, on the other hand, is about pruning groups of edges and neurons (whole filters) together at once [2, 26, 23]. Pruning can be done as part of a two-step process where removing connections is followed by retraining the net. The two steps can even be repeated multiple times in an iterative manner. Another way to prune a network is using a Genetic Algorithm [18, 46].

2.4 Genetic Algorithms

A Genetic Algorithm (GA) is a method for solving search and optimization problems [38]. The GA takes an initial population of candidate solutions to a problem (called "individuals") and evolves them towards better solutions. The individuals have a set of properties (or "genes") which the GA alters towards improvement as they are passed from one generation to the next.

GAs are based on the principle of evolution and *natural selection* [38]. The fittest individuals according to a fitness measure are selected for passing on their characteristics to the next generation. Sometimes two individuals are used for the creation of a new one – an operation called *crossover*. "One-point crossover" is when a random point on the parents' genes is chosen and the child inherits all genes to the left of that point from one parent and to the right from the other. Another type is uniform crossover, in which every gene can be inherited

from each parent with equal probability.

When it comes to creating the new population, a frequently used strategy is *elitism*. It allows the best individuals from a generation to pass on to the next one without modification in order to preserve their genes and increase the convergence speed [1]. *Culling*, on the other hand, removes the most inferior solutions from the population to create space for better individuals. *Mutation* is another characteristic of evolutionary algorithms; There is a pre-defined chance that a network's genes will undergo a random change. This strategy allows for completely random parameter values to be introduced. It also ensures genetic diversity and that the GA does not get stuck in a local minimum or maximum (i.e. finding a solution that is fitter than the ones nearby, but possibly worse than solutions at a greater distance in the search space).

2.4.1 Evolving NNs

A GA can be deployed on an initial population of neural networks to evolve a network structure that is best fit to performing a certain task [43]. This method of evolving an NN architecture, and possibly weight values, already has a lot of dedicated research [31, 33, 44]. In that case the genes are the network's parameters: number of neurons per layer, number and types of layers, number of edges and their weight values, activation functions of neurons.

In the initialization of the algorithm a population of sparse and simple neural networks is randomly generated. Next, they are evaluated on a task using the fitness function which summarizes how close this individual is to accurately doing its task. Based on the scores, only the highest-ranking networks are selected for "reproduction" to create a new generation. The new individuals share most of the features of their parents. The possible mutations are of the form of adding a new connection, or neuron, or a whole layer or filter (in the case of CNNs). An activation function can possibly also be changed.

The GA can be terminated based on a termination criterion, for instance: once a network with accuracy above a certain threshold is found; after a number of iterations; if the successive populations no longer produce better results than the ones before them; etc.

2.4.2 Pruning NNs

A GA can be used for pruning neural networks as well [18, 46]. This process is almost the inverse of evolving NNs, although it uses the same genes and also aims to produce a network with optimal performance (but sparser than the initial one).

The usual approach is taking a pre-trained network and creating an initial population out of it by copying it multiple times and pruning some part of the structure from each copy. The individuals are evaluated with the fitness function and only the best performing ones are chosen to create the next generation. The networks of the new population share most of the features of their parents, but some part of the architectures are pruned to make the networks simpler (and possibly mutation has occurred). Once again, the GA can be terminated based on multiple termination criteria.

Chapter 3

Related Work

In this chapter, we describe the state-of-the-art research already done by scientists for generalizing networks, pruning methods, and topology search using GAs.

3.1 Weight-Agnostic Neural Networks

Weight-agnostic neural networks (WANNs) form the core of our research. A WANN [11] is a model, the architecture of which contains all the necessary information for performing a given task and can carry it out even without training the weights.

Gaier and Ha [11] describe how to evolve a WANN: They start with a random population of sparse networks with no hidden nodes, run NEAT (a GA explained in more depth in the next subsection) on them, evaluate the networks based on their performance with a random distribution of weight values, and create the new population out of the fittest models. In this setting, NEAT is used for structure optimization. It modifies the network by inserting a node, adding a connection or changing the activation function. The evaluation is done using several tests with different shared weight values, and calculating the mean performance by averaging its results over all those tests.

The paper achieves models which perform tasks impressively well without any weight-training. One of the resulting networks scores around 82% accuracy on the MNIST dataset with randomly initialized weights, and 92% when the weight values are chosen to be the best performing ones [11]. The outcomes of this research indicate that the topology of NNs contributes to their performance, and not only their weights.

Furthermore, Gaier and Ha suggest evolving weight-agnostic NN models for multiple tasks. However, they do not conduct any experiments to investigate whether the baseline WANN has this property.

3.2 Network Evolution

Genetic Algorithms, like NEAT, are used in a lot of research and generally follow the typical structure of a GA described in section 2.4, but differ in the choice of fitness function, and method of selection and mutation of networks.

NEAT (NeuroEvolution of Augmenting Topologies) is a widely used algorithm for genetic evolution which can optimize the structure and the weights of a neural network [39]. The modifications of the architecture and the weights happen separately, thus the algorithm can be easily utilized for optimizing either or both. However, it can be used only on feed-forward networks without convolutional layers.

Fang et al. work with convolutional neural networks in their research [10], where they use a GA-based method to evolve network structures. The weights of the initial networks are derived using "weight inheritance" from a genotype string with all bits set to 1 which gets fully trained in the beginning (a baseline). Then the individuals are partially trained and evaluated. Tournament selection is used: Two by two, chosen at random, the networks are compared and the ones that perform better are selected for crossover and, possibly, mutation. The individuals are then re-evaluated and the process of selection, crossover and mutation is repeated. At each generation, the weights of an individual are inherited from the baseline instead of its parents. The best model obtained by Fang et al. is tested on multiple datasets, to assess its generalization ability and efficiency of reuse. Its accuracy is only about 0.6% less than the state-of-the-art when it comes to CIFAR-10 and SVHN, and 2% less for CIFAR-100.

Real et al. implement a GA in another way [31]: During each evolutionary step, two by two, models are chosen from the population and compared based on their accuracy on a validation set. The worse one is immediately removed from the population, while the better one gets modified by a mutation. This new network gets trained and becomes part of the new generation. The research uses a large number of mutations, including altering the learning rate, resetting the weight values, inserting or removing a layer, changing the filter size on a random convolution, altering the number of channels in a convolution. The result of Real et al.'s research are two models with performance on CIFAR-10 and CIFAR-100 comparable to the hand-designed architectures of state-of-the-art models.

A very different approach to creating the new generation is shown in [40]. Suganuma et al. create a random "parent" individual, train it, and create an offspring population by applying mutations on it. The "children" are trained and the parent is also mutated. The individual with the highest fitness out of the parent and children is selected as the new parent and then a new offspring population is created.

3.3 GAs for Neural Network Pruning

The method we propose in this thesis can be seen as a population-based stochastic approach for pruning neural networks using a genetic algorithm. GAs have been utilized for NN pruning for decades now, as seen in Hancock's work [18]. In his research in 1992 he shows that using a GA to prune connections in an ANN can improve its performance and increase its generalization ability on unseen data. Creating a highly generalizable network is also the focus of Bebis and Georgiopoulos. They propose combining pruning and a GA, and using a fitness function which actively encourages the reproduction of networks with high generalization performance [3]. This research, however, focuses solely on feed-forward neural networks.

Another work on feed-forward networks is given by Garcia-Gimeno et al., who prune via a GA, with aim to create an ANN for microbial growth prediction in food [12]. The mutations of the networks are in the form of connection pruning, which is a strategy we utilize as well.

The fitness function aims to both increase the generalizability and reduce the number of edges. Thus, comparing two networks with similar performance will result in picking the sparser, less complex one. The study shows that pruning using a GA is a very effective method.

A more recent research by Yang et al. focuses on CNNs [46]. The proposed algorithm selects the K top scoring individuals of the population to reproduce the next generation. Crossover is performed on the K networks, two by two: The better scoring model passes on its "genes" as they are and essentially gets copied into the new generation. The worse network's genes are used for creating a new network, but there is a 50% chance of a gene to be replaced by one of the better model's. Pruning is performed on all networks except the one with highest performance (the elite). Finally, the new generation is retrained and the process is repeated.

Yang et al. achieve an outstanding reduction in computation of an MNIST classifier at the price of a very small decrease in accuracy (less than 0.15% drop). Although impressive, the research does not test the generalization ability of the resulting model to other tasks.

3.4 Standard Pruning Techniques

In contrast with population-based stochastic network pruning, standard pruning techniques involve a single network. The NN is pruned by the application of pruning operators.

The most common way to prune a network is described by Han et al. [17]. First, the network is fully trained. Then the connections within the network are assessed and the ones deemed unimportant are pruned[1]. Finally, to make up for the loss of accuracy due to deleted parts of the structure, the model is retrained. The last two steps can be repeated multiple times. Han et al. show that this method can remove millions of edges without any accuracy loss [17] on a LeNet-5[2] model for MNIST classification. Since we evolve our networks in a weight agnostic manner, we do not make use of pruning which relies on the weight values. Instead, we conduct random unstructured pruning.

Li et al., on the other hand, showcase the usage of structured pruning, which involves removing filters of CNNs that are identified as having small effect on the output accuracy [23]. For each convolutional layer, they prune the filters with the smallest sum of absolute kernel weights and the corresponding kernels. A new kernel is created and the remaining kernel weights are copied to the new model. They achieve an impressive speed up and no drop in the network's accuracy after pruning 50% of six convolutional layers of VGG-16[3].

Most pruning techniques are similar to the described ones [24], although some have modifications. [22] proposes that the pruning of the network happens right after initialization, before training has taken place. Instead of fine-tuning the network, [25] reinitializes and retrains it entirely after every pruning pass. Pruning edges and pruning whole channels can also be combined into the so called mixed pruning [47]. Yang et al. show that this approach leads to a higher than the other methods compression of the original model: They achieve 90% compression of LeNet-5 (for MNIST classification) with only 0.18% accuracy drop.

[1] This is the basic idea behind L1-norm pruning, in which the weights with smallest absolute value (thus, the ones that contribute the least) are removed.

[2] A pioneering 7-level CNN first proposed in 1998 [21].

[3] VGG-16 is a large and commonly used CNN for image classification [37].

3.5 Multi-task and Transfer Learning

The method we propose uses a pre-trained neural network to create a population of networks the pruning of which is guided by a fitness measure based on their accuracy on the original task (MNIST) or on multiple tasks (MNIST and 10 EMNIST letters). In the latter setting, this approach can be viewed as the intersection of multi-task and transfer learning.

3.5.1 Multi-task Learning

Multi-task learning is concerned with building and training NNs with the purpose of being used for performing a diverse set of tasks. One way to create a multi-tasking network is via hard parameter sharing. [6] describes the usual network structure achieved with this strategy: There are a few task specific output layers while the rest are shared for all tasks. The model is trained on all of the tasks so that the shared layers encode information about all of them and there is a minimal risk of overfitting. Based on the desired task, the data is processed by a different set of output layers.

A different approach is presented in [7] and [50]. Using joint training, they create highly generalizable neural networks that can perform multiple tasks. The model learns the data information by looping over multiple tasks: A random task is selected, a random sample is taken and used for evaluation, the error is back-propagated, and this process is repeated. [7] does not deal with convolutional neural networks since it focuses on Natural Language Processing. However, [50] shows that this approach is useful for CNNs: By jointly training a VGG model, they achieve accuracy of more than 60% on four separate scene-centric datasets.

Mallya et all. present yet another method for creating a multi-task image classification network out of an initial "backbone" one, by learning different masks per task [27]. They learn binary masks (1 when 0 it should take part in classifying the current dataset and 0 otherwise) in a differential fashion and optimize for each task. The research uses a pre-trained network as their backbone and trains it for multiple new tasks which results in masks that are applied on the backbone to get it to work on other datasets.

3.5.2 Transfer Learning

Transfer Learning (TL) is a technique in machine learning in which a model trained on one dataset is re-purposed for another (similar) dataset. It works best if the original data used for learning has general features and the two data distributions are similar [48]. The trained model is usually modified as follows: Most of its layers are "frozen" (the weight values are kept) and the last few layers together with the classification one are retrained on the new task. This technique is especially useful when the data available for a particular task is minimal and using such little data on its own for training the model risks overfitting.

Transfer learning has proven to give CNNs for image classification and recognition tasks a performance boost. For instance, [36] shows that training a CNN on the generic dataset ImageNet, and then applying transfer learning in order to use the model for medical image classification leads to state-of-the-art performance. Han et al. also display the merits of transfer learning for tasks with very limited datasets: they train a CNN on large datasets to give it a general structure then very successfully make it work on small datasets [15]. Other research confirms the usefulness of transferring features for CNNs [35, 8, 34].

Chapter 4

Generalizability of WANNs

In this chapter we investigate the generalization ability of the WANNs to multiple tasks. We use the algorithm proposed in [11].

4.1 WANN Algorithm

First, we present a high level description of the WANN algorithm.

1. An initial population of minimal topologies is created. All of them have only an input and an output layer.
2. The networks are evaluated in multiple rollouts. At each rollout, a different single shared weight value is assigned to the connections and the performance of the networks is determined by their cumulative cross entropy loss.
3. Using tournament selection, the algorithm chooses networks probabilistically to create the next generation. Out of two models with similar fitness, the sparser one is chosen.
4. The individuals are varied with one of the following mutations: inserting a neuron, adding a connection, or changing an activation function.
5. If the maximum number of iterations is reached, stop. Otherwise, go back to step 2.

Throughout the evolution the networks become increasingly complex and perform better. For the image classification setting specifically, the algorithm makes use of elitism and culling, but does not utilize crossover. The shared weight values used for evaluation are from the set $\{-2, -1, -0.5, 0.5, 1, 2\}$.

4.2 WANN for Handwritten Digit Classification

Gaier and Ha evolve a feed forward deep weight agnostic network with 762129 non-zero edges and 873 neurons, spread into 22 layers. This champion has 91% accuracy on the MNIST dataset[1]. The resulting WANN has different performance depending on the shared weight

[1]They conduct a series of tests, most of them on reinforcement learning, but the classification test is done on MNIST.

value given to its edges which leads to the suggestion of using it in the following manner: "Each weight value of the network can be thought of as a distinct classifier, creating the possibility of using one WANN with multiple weight values as a self-contained ensemble." [11, p.8]. Furthermore, they state the possibility of developing a WANN encoding information about many different tasks "that can easily be fine-tuned for a particular downstream task in its environment later on" [11, p.9].

We are interested by these prospects and what they could mean for the further development of multi-tasking and highly generalizable networks. Thus, we investigate whether the baseline WANN approach is already sufficient for creating such network that can perform multiple tasks. We evaluate the accuracy of the champion WANN evolved as an MNIST classifier on another dataset: 10 classes from EMNIST Letters[2]. We choose EMNIST Letters because of its relative simplicity and we believe it to be somewhat related to MNIST.

The implementation of WANN and its results are publicly available[3] for reproducing the experiments. We extend the code to include calculating and logging a network's accuracy and enable tests on EMNIST Letters[4].

The obtained results are displayed in Table 4.1.

	-2	-1	-0.5	0.5	1	2
MNIST	90%	**91%**	81%	82%	89%	88%
EMNIST 10 Letters	6%	7%	6%	8%	7%	**9%**

Table 4.1: The accuracy of the champion MNIST WANN on the MNIST test dataset and the 10-letter EMNIST subset dataset for the 6 different shared weight values.

WANN has a very high accuracy (91%) on MNIST when its shared weight value is -1. However, we can see that its accuracy on 10Letters is below random guessing no matter the assigned shared weight value. Therefore, evolving the model in a weight-agnostic way on its own does not increase its generalization ability and fails to obtain a network with multiple skills.

4.3 WANNs With Joint Evaluation

The low accuracy on a dataset the model has never seen before is to be expected. Because of that, we modify the genetic algorithm to asses a network's fitness based not only on the performance of the network on MNIST, but also on the first 10 classes from EMNIST Letters. We call this adapted fitness evaluation "joint evaluation" and the resulting model – JointWANN.

[2]We use only 10 classes from the original 26-class dataset so that there are no modifications to the produced WANN structure needed. We simply take the first 10 classes to avoid cherry-picking.

[3]They can be found on the project's official GitHub repository.

[4]The original code tests the networks on MNIST and only calculates the loss. All changes made to the original WANN code are documented in our own repository.

We run the algorithm with the default GA hyperparameters to fully reproduce Gaier and Ha's experiment. The results of JointWANN are displayed in Table 4.2 but it is important to note that due to time limitations we only conduct one trial. The resulting model has 290521 edges connecting 539 neurons into 10 layers. It is much simpler than WANN, but the GA converged and stopped improving early on which could indicate that it fell into a local maximum.

	-2	-1	-0.5	0.5	1	2
MNIST	9%	10%	**11%**	7%	6%	9%
EMNIST 10 Letters	62%	**66%**	51%	33%	57%	57%

Table 4.2: The accuracy of the JointWANN on the MNIST and 10-letters EMNIST test datasets for the 6 different shared weight values.

As we can see, this JointWANN is performing overall worse compared to the WANN Gaier and Ha evolve, with random-guessing level accuracy on one task and only 66% on the other. It can be said that it is an adequate classifier of the 10 letters from EMNIST, but not an MNIST one. We hypothesise that the EMNIST dataset is slightly more complex and because of that the loss calculated from the performance on the 10 letters dominates the fitness.

Our results show that even when evaluated on more than one dataset, the WANN is unable to encode information for more than a single task.

Chapter 5

The PADAWANN

Given the results of the previous chapter, indicating WANN is not as general as desired on "related" tasks, in this chapter we investigate whether using knowledge from a pre-trained network will improve the results. For this purpose, we reverse the evolutionary process: While WANN grows an NN topology, we start from a pre-trained one to build a population of architectures which then get pruned during the evolution.

5.1 Algorithm

5.1.1 Network Pruning

We evolve a PADAWANN from a model trained on an image dataset for classification. Thus, the structure of the model used as a starting point already encodes information about the features of visual data. This provides a head start in the search for a general model for image classification. Moreover, there are already dozens of hand-crafted models for the purpose of image classification and it is reasonable to make use of them instead of trying to recreate them with no guarantee of achieving the same success.

We make use of unstructured pruning[1]. A single pruning step is performed on every model chosen for creating the next generation. 0.025% of the network's edges are randomly selected for pruning with all edges from the convolutional and dense layers having an equal probability of being picked. This parameter value of 0.025% was experimentally found to be fitting for our problem.

5.1.2 Weight Agnosticism

The PADAWANN is evolved in a weight-agnostic way. We discard the pre-trained weights (although we keep their signs) and initialize them to a shared weight value sampled from the set $\{0.5, 1, 2\}$[2]. Training the model during the evolution with a shared weight value instead

[1] Using L1-norm pruning would be almost equivalent in the sense that the edges have a shared absolute value and thus all of them will be considered equally important and have the same chance of being pruned.

[2] Those are the absolute values of the weights used by Gaier and Ha. We use the positive values only because, as already mentioned, we keep the original weight signs meaning that we only set the weights' absolute value.

of the pre-trained weights of the initial network is what guarantees the weight agnosticism of the model.

Due to the fact that the weights are irrelevant, we do not update them via back-propagation.

5.1.3 Evolution

Our algorithm consists of the following steps:

1. Take a pre-trained model and create the initial population by copying it multiple times and performing a single pruning step on each copy.
2. Remove the worst performing networks (culling a specified percentage of the population) and copy the best performing ones to the next generation without modifying them (elitism; preserving a specified percentage of the population).
3. Use tournament selection to pick which networks will create the next generation: Comparison is done between randomly chosen models, the number of which (i.e. the tournament size) can be specified as an argument to the algorithm.
4. One-point crossover with user-determined probability p is carried out between two tournament winners and the offspring is mutated. If no crossover happens, there is a single tournament and the network with highest fitness is mutated.
5. Mutation with probability of 1 happens in the form of pruning. The type of pruning is already explained in section 5.1.1.
6. If the maximum number of iterations is exceeded, stop. Otherwise, go back to step 2.

The fitness of the networks is calculated using the accuracies obtained via joint evaluation[3]. For calculating the accuracy of a network on a certain dataset, we conduct multiple rollouts, using a different shared weight value each time, and take the highest obtained accuracy score.

The fitness function is defined in the following formula:

$$fitness(model) = \frac{\sum_{d \in dataset} accuracy\ of\ model\ on\ d}{\sum_{d \in dataset} 100 - accuracy\ of\ model\ on\ d} \tag{5.1}$$

The fitness measure was found suitable through experimentation with multiple potential fitness functions and it takes into account the performance on all tasks. It can be thought of as a "scaled sum" of the accuracies of a network on the datasets it is evolved on. The scalar ensures that the GA favors models with higher train accuracy across all datasets and penalizes low accuracies on even one of them. Note that this number on its own has no meaning and well defined-range; It is only important to maximize it with respect to the initial model's fitness value.

5.1.4 Creating The PADAWANN

The algorithm we have designed is illustrated by the pseudo-code in Algorithm 1. The output is the ready-to-use PADAWANN.

[3]This technique is akin to the joint training used in the multi-task field [7, 50] with the addition of the weights sharing one value. Moreover, no retraining is carried out.

Algorithm 1 Evolving a pre-trained model into a PADAWANN

Input: pre-trained NN *net*
Output: PADAWANN
`evolve()`:

pop ← copies of *net*, which get pruned once ▷ This contains the "current" population
best ← None ▷ This is the best model (according to the fitness function)
iter ← 0 ▷ The current iteration
while *iter* < `MAX_ITERATIONS` **do**
    next ← ∅ ▷ This is the next generation
    next ← *next* ∪ {the models with best accuracy from *pop*}
    pop ← *pop* – {the models with worst accuracy from *pop*}
    while the size of *next* < `POP_SIZE` **do**
        if crossover happens **then**
            individual ← the offspring of two tournament winners
        else
            individual ← choose from *pop* via tournament selection
        end if
        prune *individual*
        next ← *next* ∪ {*individual*}
    end while
    if *model* with highest fitness from *next* has higher fitness than *best* **then**
        best ← *model*
    end if
    pop ← *next*
    iter ← *iter* + 1
end while
file ← *best* ▷ Store the PADAWANN

We implement the genetic algorithm from scratch, but utilize useful Python machine learning libraries (mainly PyTorch) that facilitate working with NNs. All relevant information about the code and used libraries and pointers to outside materials can be found in Appendix A.

5.2 Preliminary Experiment Information

Due to the stochastic nature of genetic evolution, every described experiment is repeated 10 times and the numbers given are averaged over those trials. The hyperparameters used (which we will refer to as "default") are given in Table 5.1 and they are utilized in all tests unless stated otherwise.

Number of Generations	Population Size	Elitism	Culling	Pruning	Crossover	Tournament Size
50	20	30%	30%	0.025% (unstructured)	50% (one-point)	8

Table 5.1: The default hyperparameters of the GA used in our experiments.

The values we choose for elitism, culling, crossover probability and pruning amount in our experiments are heuristic ones based on preliminary tests. Due to time limitations, we do not fine-tune the hyperparameters.

5.2.1 Initial Models

For the majority of the experiments we use a convolutional neural network pre-trained on MNIST (henceforth, MNIST-Net) as initial model for the GA to fairly compare our approach to weight agnosticism to Gaier and Ha's. Moreover, we want to test the algorithm on a simple problem as a proof of concept to gauge the plausibility of our goal before tackling more complex tasks, and MNIST is widely accepted as one of the simplest datasets for image classification. Since the classification of the MNIST digits is a relatively simple task, we train a relatively simple model ourselves. It has 98% accuracy on the test MNIST data and 1.2 million non-zero edges.

For another set of tests, we make use of a model pre-trained on 10Letters (henceforth, 10Letters-Net). It has the same architecture as MNIST-Net and 96% accuracy on the test 10Letters dataset. The last network we work with is a three-layer feed-forward model with no convolutions. It has 97% accuracy on MNIST and just over 50,000 non-zero edges.

5.2.2 Datasets

Note that due to time limitations, we use a small sample of the train data for the evolution; We use 700 samples per class for both datasets.

In the joint evaluation we evaluate the networks on MNIST and on a subset of the EMNIST Letters dataset. We take only the first 10 classes from it to match the number of MNIST digits classes and output features of the model; We refer to this dataset created by us as 10Letters. We choose 10Letters because of its relative simplicity and our belief that it is related to MNIST, which should we think would facilitate the domain transfer task. MNIST and EMNIST Letters have a lot of features such as straight lines, curves, circles that are alike (as seen in fig. 5.1), and are both Grayscale. We want to first gauge the feasibility of PADAWANN when the datasets are at least slightly similar, before considering generalizability to any image data.

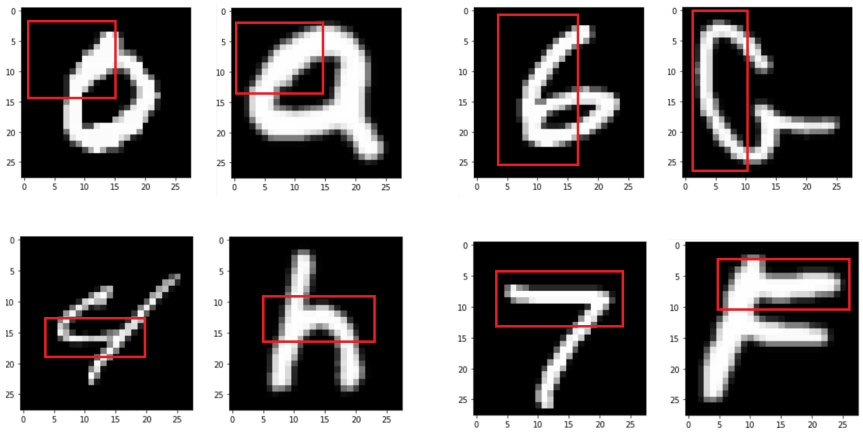


Figure 5.1: Examples of the common features of the MNIST and EMNIST 10Letters datasets.

Other two datasets we make use of for generalization testing are another EMNIST subset (which we call Another10 Letters) and FashionMNIST – also 10-class Grayscale data. Generalizing to datasets with a different number of classes, for which it is required to use less output nodes or to add new ones, is not considered in this thesis.

The first new dataset is also a subset of EMNIST Letters made up of classes 11-21 (corresponding to letters 'k' through 't'). This dataset, Another10 Letters, is "related" to the ones the model already knows from its training and evolution. Therefore, we expect accuracy higher than the initial model's. Moreover, we conduct a test on another less related dataset. We choose FashionMNIST because it also contains Grayscale images with 10 classes but has more complex features.

Because the test datasets we use are balanced, the accuracy of the networks is a good measure to compare them by which is why we focus on it for our main analysis.

5.2.3 Sign of the Weights

The results of our preliminary experiments showcase a great difference in the network's performance when the signs of the original weight values are kept compared to when they are not. In true weight-agnostic fashion the sign is discarded as it is part of the weight information of the initial model. However, the accuracies and loss achieved when the signs are not kept are very bad, as seen in Table 5.2. Figures displaying network's evolution over the generations can be found in Appendix B, section B.2.

MNIST Accuracy	10Letters Accuracy	Loss	Number of Edges
20.3% (± 5.46%)	14.34% (± 1.75%)	464483.35 (± 1032441.97)	1016651 (± 47906)

Table 5.2: The average accuracy and sparsity of the PADAWANN when the signs are discarded and the GA is run with the default parameters.

The loss is very high, indicating the models find it hard or impossible to converge, and the number of edges does not seem to get reduced by much. The low accuracies are not surprising as the network used has two fully-connected layers with all non-zero weights. Setting that network's weights values to a shared constant one is expected to lead to suboptimal performance as all samples would be given the same class. This is illustrated in the confusion matrices given in Figure 5.2.

It appears necessary to provide some additional guidance to the network (in the form of keeping the signs) when evolving the PADAWANN from a pre-trained model. In light of that fact we report only on the experiments conducted with the original weight signs kept.

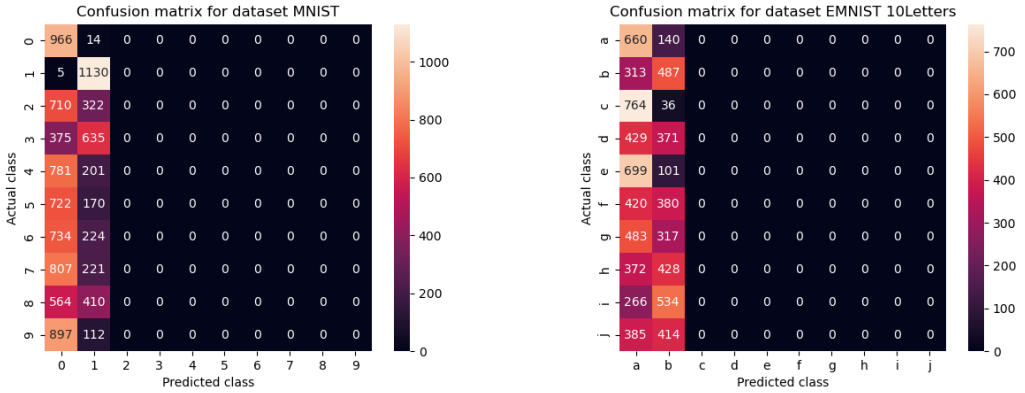


Figure 5.2: The MNIST and 10Letters confusion matrices of the model evolved in a fully weight agnostic way.

5.2.4 Feed-forward Networks

During our preliminary tests we experiment with starting the evolution from a feed-forward model without any convolutional layers. The results are summarized in Table 5.3 and the evolution is illustrated in Appendix B, section B.5.

	MNIST Accuracy	10Letters Accuracy	Loss	Number of Edges
Initial Model	97.29%	8.29%	0.05	50816
PADAWANN	80.314% (± 1.79%)	12.66% (± 2.37%)	11.15 (± 0.95)	34134 (± 1703)

Table 5.3: The average accuracy and sparsity of the PADAWANN when the initial network has no convolutional layers compared to the network it was evolved from.

Although the loss has not increased by much and the model is 1.5x sparser than the initial one, there is barely any improvement in the 10Letters accuracy and the drop in MNIST accuracy is significant. Those results show that the GA is applicable to any type of network, but having a feed-forward model as initial one results in PADAWANNs with worse performance. We hypothesise that this could be due to the fact that the feed-forward model has only fully-connected layers which does not combine well with weight agnosticism. For that reason, we conduct the rest of our experiments on a CNN.

5.3 PADAWANN

5.3.1 Results

The average results of the PADAWANN evolved from a pre-trained convolutional neural network are presented in Table 5.4. The MNIST accuracy is still very high, and the 10Letters accuracy has improved. The model appears to make very wrong predictions sometimes, as told by the loss value, but the number of edges is quite low.

MNIST Accuracy	10Letters Accuracy	Loss	Number of Edges
93.67% (± 0.54%)	22.1% (± 0.85%)	272.8 (± 280.56)	600460 (± 38033)

Table 5.4: The average accuracy and sparsity of the PADAWANN when the initial network is MNIST-Net.

The accuracy evolution of the best model throughout the generations is displayed in Figure 5.3. The evolution of the loss, number of non-zero edges and fitness is included in Appendix B, section B.1 for completeness.

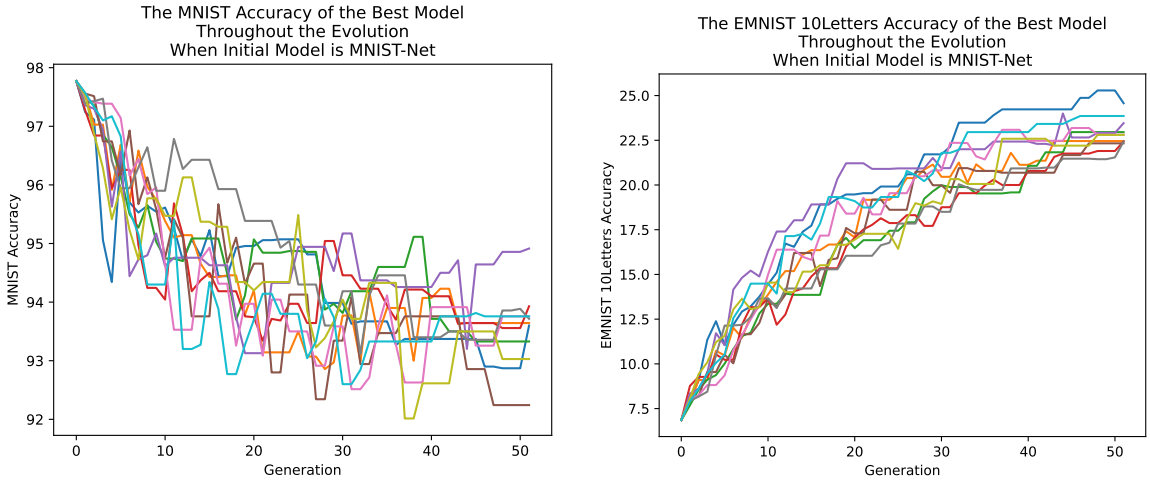


Figure 5.3: The evolution of the PADAWANNs' accuracy on the train MNIST and 10Letters datasets, with each trial represented in a different color.

The best, worst and average accuracy are compared in Table 5.5. The standard deviation of the results from the different trials is very low which gives the average result more validity.

	Best Accuracy	Worst Accuracy	Average Accuracy
MNIST	94.99%	92.99%	93.67% (± 0.54%)
10Letters	23.8%	21.015%	22.1% (± 0.85%)

Table 5.5: Comparison of the best, worst and average accuracy achieved on the MNIST and 10Letters datasets by the PADAWANN.

5.3.2 PADAWANN vs Pre-trained Networks

To put our model's performance in perspective, we compare the accuracy achieved by the PADAWANN to that of the initial MNIST-Net and the 10Letters-Net network (in Table 5.6).

	MNIST Accuracy	10Letters Accuracy
MNIST-Net	98.13%	6.68%
10Letters-Net	7.35%	96.17%
PADAWANN	93.67%	22.1%

Table 5.6: The accuracies of the pre-trained models (with their original pre-trained weights) and of PADAWANN.

The accuracy of the PADAWANN on the MNIST test set is lower than the accuracy of MNIST-Net, and its accuracy on the 10Letters test set is lower than the accuracy of 10Letters-Net. Therefore, the PADAWANN does not perform as good as or better than a network specifically trained on a certain task, but that is expected. In contrast to the other two networks, the PADAWANN's goal is to encode information about multiple datasets instead of specializing on one of them (ensured by the fitness function which favors networks with good performance on both datasets). Thus, the PADAWANNs are no longer looking for MNIST-specific features, but for features of handwritten digits and some letters. Because of this property, we believe it is reasonable for the PADAWANN to be (slightly) underfitting for both datasets. We are essentially doing a trade-off between high performance on one task and better than random guessing performance on multiple.

The accuracy of PADAWANN on MNIST has decreased by about 4.46% from MNIST-Net's. Although the p-value < 0.01 shows a significant difference in performance, we think PADAWANN can still be considered a good MNIST classifier. This is evident in the confusion matrix of PADAWANN, given in Figure 5.4. It is interesting to note that the type of errors made do not seem random and can in fact be related to the newly introduced task. For instance, '1' is classified as an '8' 44 times and the 8th class in 10Letters is actually 'i' – a letter visually similar to the digit '1'. As another example, '5' is classified as a '3' 27 times and the 3rd class in 10Letters corresponds to 'c,' a letter which has the same curve.

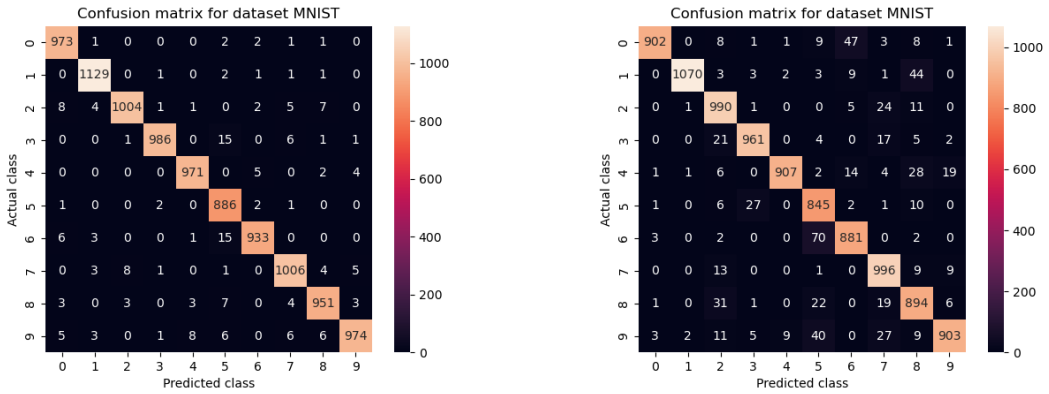


Figure 5.4: The confusion matrices of MNIST-Net (left) and PADAWANN (right) for the MNIST dataset.

While the accuracy of the PADAWANN on 10Letters is only about twice as good as random guessing and therefore the model cannot be used as a reliable classifier for that dataset, the improvement over the initial model is significant (p-value < 0.01). PADAWANN has 3.3x higher accuracy on 10Letters than MNIST-Net, which indicates a higher generalization ability to this new task. This boost in generalizability without ever training the model's structure to recognize the dataset can potentially be increased further by applying strategies from transfer or multi-task learning.

Figure 5.5 compares the confusion matrices of the initial model and the PADAWANN for the new task. As we can see, MNIST-Net tends to classify the samples as class 'a', indicating that the relation between the two domains and tasks is biased and weak, given that model. On the other hand, while PADAWANN seems to classify a lot of the samples as class 'c', it is clearly starting to learn features of the dataset and classify classes 'c', 'f', 'h' and 'i' correctly.

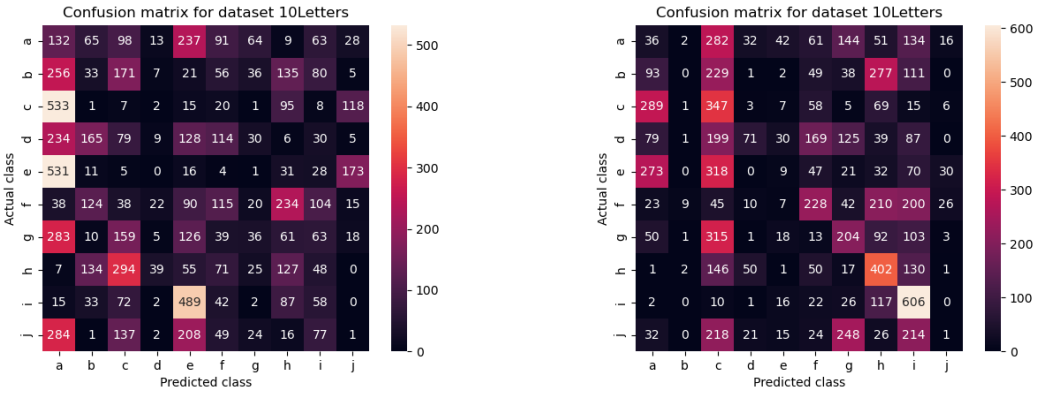


Figure 5.5: The confusion matrices of MNIST-Net (left) and PADAWANN (right) for the 10Letters dataset.

5.3.3 PADAWANN vs WANN

We now compare the digit classifier WANN evolved by Gaier and Ha and the JointWANN we created and documented in Chapter 4 to the PADAWANN. Their complexity and accuracy are summarized in Table 5.7.

	MNIST	EMNIST 10Letters	Number of Edges
WANN	91%	9%	762129
JointWANN	11%	66%	290521
PADAWANN	93.67%	22.1%	600460

Table 5.7: The average accuracy and sparsity of the three models.

The accuracy PADAWANN has on MNIST is higher than WANN's and JointWANN's making it the best *partially* weight-agnostic network out of the three in terms of MNIST digits classi-

fication. Its accuracy on EMNIST 10Letters is lower than the JointWANN's but higher than the WANN's. In terms of overall performance, we believe that the PADAWANN has slightly higher generalization capabilities than the WANNs. Moreover, PADAWANN is sparser than WANN. However, because WANN and PADAWANN are implemented in different ways, using different libraries, it is impossible to formally investigate whether their performance is significantly different or not.

5.3.4 Generalization Over Unseen Tasks

We investigate how generalizable the PADAWANN actually is over new tasks. Essentially, we test whether being evolved with a fitness function involving information on MNIST and 10Letters datasets will increase its capability to generalize to other previously unseen tasks.

The average accuracy of PADAWANN and MNIST-Net when tested on FashionMNIST and other handwritten letters is given in Table 5.8.

	Another10 Accuracy	FashionMNIST Accuracy
MNIST-Net	9.99%	7.82%
PADAWANN	13.79% (± 1.78%)	10.895% (± 2.04%)

Table 5.8: The average accuracy of the initial model MNIST-Net and the PADAWANN evolved from it on the Another10 Letters and FashionMNIST datasets.

As we can see, the accuracies PADAWANN achieves are slightly higher than the initial model's and significantly different. Despite that, for those unseen-before datasets, PADAWANN seems unable to group samples of the same class together. Some classes are never even predicted at all. This shows that the generalization ability of the network extends only to the datasets used in the evolution of the model. The confusion matrices of PADAWANN for the two datasets iluustrate this in Figure 5.6.

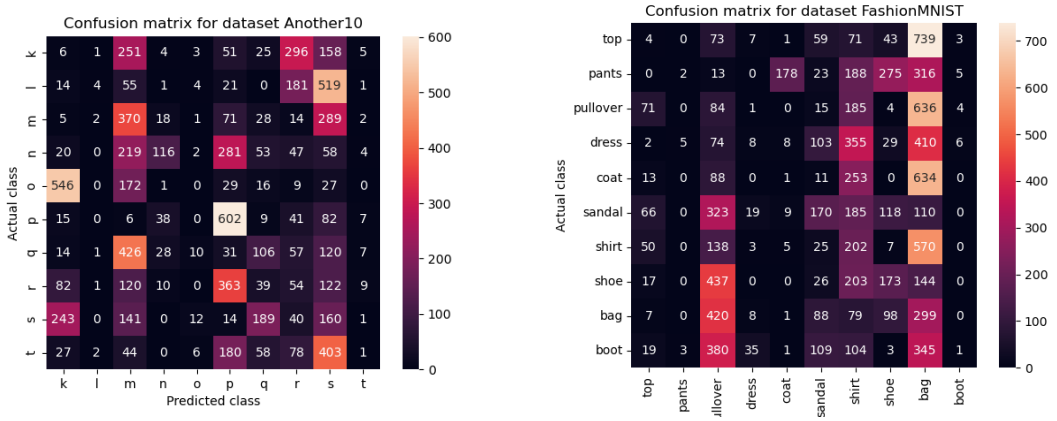


Figure 5.6: The confusion matrices of PADAWANN for the Another10 and FashionMNIST datasets.

5.4 Three-task PADAWANN

Following that conclusion, we test how the PADAWANN will behave if it is evolved with the fitness function considering three datasets. We want to understand whether two is an upper limit to the number of datasets a model can become generalizable to, and if being exposed to an additional dataset will lead to the MNIST accuracy decreasing even further.

We run the GA with MNIST-Net as initial network and the default parameters. We evaluate the models on MNIST, 10Letters and Another10. The results of the evolved 3D-PADAWANN ("Three Dataset PADAWANN") are presented in Table 5.9 and a comparison between the initial model, PADAWANNN and 3D-PADAWANN is presented in Table 5.10. (The evolution of parameters is given in Appendix B, section B.4.)

MNIST Accuracy	10Letters Accuracy	Another10 Accuracy	Loss	Number of Edges
90.99% (± 0.65%)	19.44% (± 2.39%)	24.35% (± 3.6%)	405.8 (± 385)	627897 (± 55000)

Table 5.9: The average accuracy and sparsity of the three-task PADAWANN.

	MNIST Accuracy	10Letters Accuracy	Another10 Accuracy	FashionMNIST Accuracy
MNIST-Net	98.13%	6.68%	9.99%	7.82%
PADAWANN	93.67%	22.1%	13.79%	10.895%
3D-PADAWANN	90.99%	19.44%	24.35%	13.79%

Table 5.10: Comparison of the average accuracies of the initial model, the PADAWANN and 3D-PADAWANN.

The further decrease in MNIST accuracy, in comparison to PADAWANN's, is expected. As there is a larger number of tasks, the model tries to leverage across all of them, and thus, it is natural that the accuracy on the original task drops.

PADAWANN is also superior when it comes to 10Letters accuracy, but 3D-PADAWANN comes close with a 2.9x improvement over the initial model. Furthermore, 3D-PADAWANN achieves 2.4x higher accuracy on Another10 than MNIST-Net – more than 10% higher than PADAWANN. The confusion matrices presented in Figure 5.7 show that 3D-PADAWANN has indeed started to learn about the features of the new datasets it was evolved on.

Lastly, we can see that the accuracy on FashionMNIST seems to increase even more than when only two datasets were used, but the confusion matrix in Figure 5.7 shows that it is merely because one class gets predominantly correctly classified.

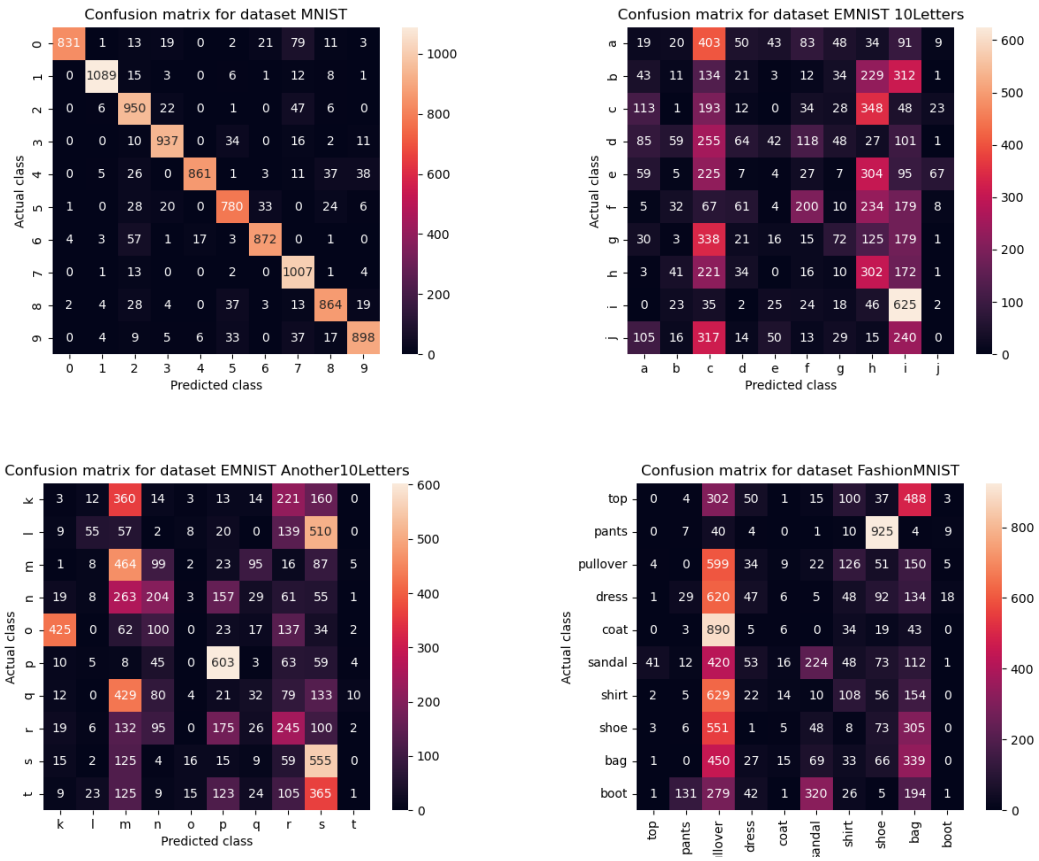


Figure 5.7: The confusion matrices of 3D-PADAWANN for the MNIST, 10Letters, Another10 and FashionMNIST datasets.

These results are somewhat expected and further confirm that we are trading off high accuracy on a single dataset for above random-guessing on multiple.

5.5 10Letters-Net as Initial Network

We run the GA again, starting the evolution from the model pre-trained on EMNIST 10Letters in order to investigate whether the results we have obtained thus far are specific to models trained on MNIST only. We also want to find out whether the task the model was originally trained for determines the results and derive conclusions about the algorithm's symmetry.

The PADAWANN's characteristics are given in Table 5.11. We notice the same trend as for the PADAWANN evolved from MNIST-Net: The accuracy on the dataset the initial network was pre-trained on drops by a few percent (6.92% in this setting), while its accuracy on the never-seen-before dataset increases (2.35x higher in this case). However, the results are not as good as the ones of the PADAWANN evolved from MNIST-Net.

MNIST Accuracy	10Letters Accuracy	Loss	Number of Edges
17.296% (± 2.56%)	89.25% (± 1.54%)	817.48 (± 596.11)	768888 (± 64779)

Table 5.11: The average accuracy and sparsity of the PADAWANN when the initial network is 10Letters-Net.

The accuracy evolution of the best model throughout the generations (displayed in Figure 5.8) seems to be similar to the one we observed when starting from MNIST-Net but with higher variation in the results. The evolution of the loss, number of non-zero edges and fitness is included in Appendix B, section B.8 for completeness.

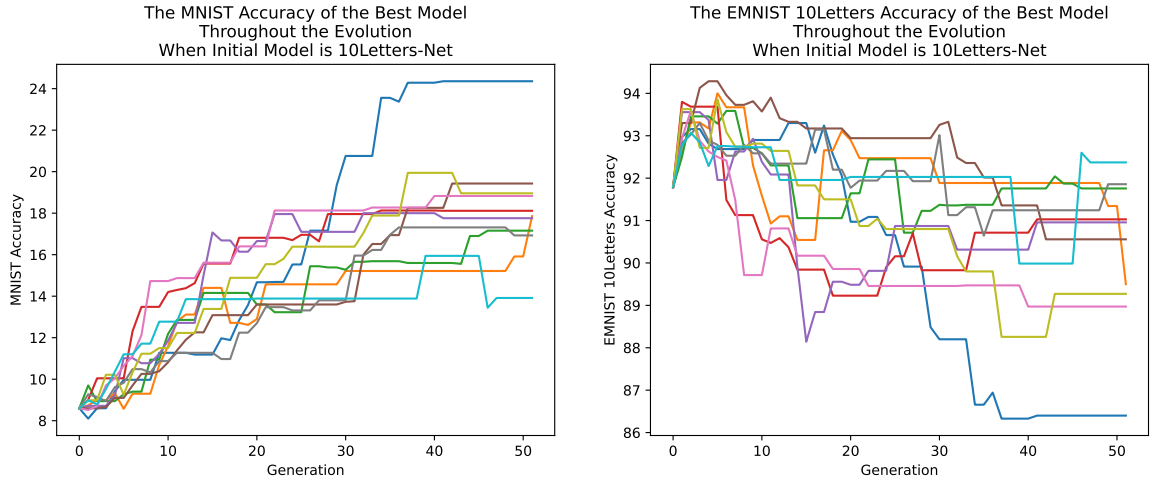


Figure 5.8: The evolution of the PADAWANN's accuracy when evolved from 10Letters-Net, with each trial represented in a different color.

	Best Accuracy	Worst Accuracy	Average Accuracy
MNIST	22.67%	12.55%	17.296% (± 2.56)
10Letters	91.11%	85.59%	89.25% (± 1.54)

Table 5.12: Comparison of the best, worst and average accuracy achieved on the MNIST and 10Letters datasets by the PADAWANN.

These results show that our algorithm yields similar performance when starting with a network pre-trained on MNIST and on 10Letters. However, when MNIST-Net is initial model we obtain significantly better results.

5.6 Transfer Learning

As shown by our results so far, evolving a weight-agnostic network while pruning it increases its generalization ability, but not to an extent that allows us to reliably use the PADAWANN for multiple tasks. Thus, we investigate whether allowing the network to *learn* about the new datasets' features will cause the PADAWANN to outperform the initial model.

For that purpose we retrain the last layer of the PADAWANN (related to classification) using the corresponding train data of the dataset on which we will test the model. The rest of the layers are kept fixed at the value $\in \{0.5, 1, 2\}$ which was found to be optimal for this dataset.

In Table 5.13 we present the average accuracies achieved by the initial model and by the PADAWANN.

		MNIST	10Letters	Another10	FashionMNIST
MNIST-Net	No TL	98.13%	6.68%	9.99%	7.82%
	TL	98.1%	58.69%	72.7%	67%
PADAWANN	No TL	93.67%	22.1%	13.79%	10.895%
	TL	96.48%	55%	67.14%	63.55%

Table 5.13: Comparison between the performance of MNIST-Net and the PADAWANN evolved from it before and after fine-tuning the last layer.

Transfer learning has a large impact, as expected. The results and the improvement over the preliminary version of the PADAWANN are impressive. While the performance of the fine-tuned initial network is slightly better than the fine-tuned PADAWANN's by about 3-5% on each dataset, the PADAWANN is twice as sparse.

Chapter 6

Ablation Study

To further understand the efficiency and contribution of the separate PADAWANN components and GA hyperparameters, we carry out an ablation study. In this chapter we describe those experiments and display the results.

6.1 Hyperparameter Investigation

We conduct a number of tests in order to investigate the usefulness of the GA parameters we utilize (elitism, culling, unstructured pruning, etc.). The results are summarized in Table 6.1. We only specify the parameter that is modified in a certain trial, the rest are the default parameters (as given in Chapter 5, section 5.2) although we scale down the population size to 10 individuals. Because of that we also adjust the tournament size accordingly to 4.

	MNIST Accuracy	10Letters Accuracy	Loss	Number Of Edges	Fitness Value
Default Parameters	93.46% (± 0.76%)	19.73% (± 1.89%)	344.68 (± 496%)	658028 (± 42429)	1.33 (± 0.04)
No Elitism	93.7% (± 1.06%)	17.13% (± 2%)	525.22 (± 780.17)	720954 (± 84299)	1.26 (± 0.03)
No Culling	94% (± 0.82%)	19.44% (± 1.35%)	522.28 (± 403.1)	676044 (± 35924)	1.33 (± 0.02)
Structured Pruning	95.87% (± 0.77%)	13.99% (± 1.62%)	1921.05 (± 1632)	855915 (± 52403)	1.24 (± 0.02)
Without Crossover	92.99% (± 1.06%)	20.02% (± 1.58%)	284.06 (± 392.2)	675996 (± 32766)	1.32 (± 0.03)
Uniform Crossover	92.89% (± 1.22%)	20.09% (± 1.43%)	331.411 (± 453.6)	681010 (± 29999)	1.33 (± 0.03)

Table 6.1: The average results the PADAWANNs obtained with the specified parameters.

As we can see, the performance of PADAWANN does not change much when the different parameters are used. However, when compared to the PADAWANN obtained with default parameters, all other PADAWANNs seem to be significantly different as given by the p-value which is < 0.01 in all five tests.

To investigate the influence of the tournament size on the performance of the GA, we conduct two additional tests with population size 10 and smaller and bigger tournament size. The results are given in Table 6.2.

	MNIST Accuracy	10Letters Accuracy	Loss	Number Of Edges	Fitness Value
Tour Size = 2	93.97% (± 0.84%)	18.72% (± 1.2%)	320.6 (± 490.5)	676672 (± 42905)	1.31 (± 0.03)
Tour Size = 4	93.46% (± 0.76%)	19.73% (± 1.89%)	344.68 (± 496%)	658028 (± 42429)	1.33 (± 0.04)
Tour Size = 6	93.2% (± 0.52%)	20.48% (± 1.54%)	134.67 (± 231.5)	656885 (± 49286)	1.35 (± 0.04)

Table 6.2: The characteristics of the PADAWANNs obtained with differing tournament sizes.

There appears to be a trend: The bigger the tournament size, the higher the accuracy on 10Letters, the sparser the model, and the larger the drop in MNIST accuracy. However, no conclusions can be derived from this as the experiment is very limited.

6.2 Weight Agnosticism

We investigate what influence weight agnosticism has on the model and its generalization ability. We run the GA again but keep the pre-trained weight values of the initial model. The resulting model we will refer to as PGA-NN, short for "Pruning GA neural network". Its averaged characteristics are summarized in Table 6.3.

MNIST Accuracy	10Letters Accuracy	Loss	Number of Edges
83.02% (± 9.26%)	15.42% (± 1.13%)	345.33 (± 211.6)	560583 (± 77727)

Table 6.3: The average accuracy and sparsity of the PGA-NN.

We can see that the algorithm produces a PGA-NN with more than twice higher accuracy score on the 10Letters dataset than the initial model. The MNIST accuracy on the test dataset has dropped significantly from the initial model's and from the PGA-NN's on the train dataset. The difference between the two networks is evident from $p < 0.01$.

The accuracy evolution throughout the generations is displayed, together with that of the PADAWANN's, in Figure 6.1 and a comparison of their accuracies on all four datatsets

before and after applying transfer learning is given in Table 6.4. (The evolution of PGA-NN is presented in figures in Appendix B, section B.7.)

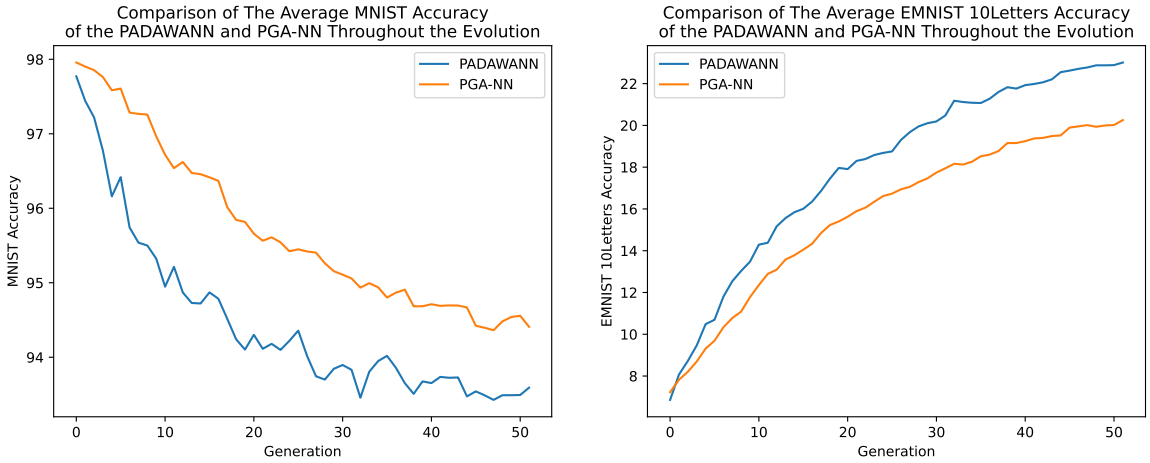


Figure 6.1: Comparison of the train accuracy evolution of PADAWANN and PGA-NN.

		MNIST	10Letters	Another10	FashionMNIST
PGA-NN	No TL	83.02%	15.42%	12.9%	10.72%
	TL	95.07%	54.44%	64.17%	61.3%
PADAWANN	No TL	93.67%	22.1%	13.79%	10.895%
	TL	96.48%	55%	67.14%	63.55%

Table 6.4: Comparison between the performance of the PGA-NN and the PADAWANN on all four datasets before and after fine-tuning the last layer.

As we can see, PADAWANN is superior when it comes to everything except sparseness; PGA-NN is slightly more compact. Even after fine-tuning, the PADAWANN remains a few percent better in terms of accuracy. PGA-NN's large drop of accuracy on MNIST between train and test data indicates that when the evolution is not carried out in a weight agnostic way, the model overfits. Furthermore, PADAWANN achieving higher average accuracy over PGA-NN (with $p < 0.01$) indicates that weight agnosticism helps the network gain higher generalizability and balance between the performance on the different tasks. We believe that is because it abstracts it from the pre-trained weight values, which are biased towards one dataset.

To further understand the influence of weight agnosticism, we take MNIST-Net and evaluate its accuracy on MNIST and 10Letters in a weight agnostic way. The results are as follows: 97.69% accuracy on MNIST and 6.4% on 10Letters. Interestingly enough, the accuracy scores barely change from the ones obtained with the pre-trained weights. While the accuracy does not drop significantly, the generalization ability of the models is also not higher. This indicates that weight agnosticism does not boost generalizability on its own, but in combination with the other PADAWANN components.

6.3 Joint Evaluation

6.3.1 Fitness Based Only on MNIST Accuracy

It is our hypothesis that joint evaluation is an absolute necessity when the goal of the GA is evolving a general topology, in order to bypass the bias the network has towards the dataset it was pre-trained on. To test this hypothesis, we run the GA again but the fitness function only considers the accuracy on MNIST.

The accuracy and parameters of the resulting network (which we call MNIST-Only) are given in Table 6.5. Their evolution is presented in Appendix B, section B.6.

MNIST Accuracy	10Letters Accuracy	Loss	Number of Edges
97.95% (± 0.12%)	6.07% (± 0.53%)	1201.48 (± 911.9)	996093 (± 53657)

Table 6.5: The average accuracy and sparsity of the MNIST-Only model.

Since the fitness function no longer considers the network's accuracy over the multiple tasks, the resulting low accuracy on 10Letters is not surprising. It confirms that in order to achieve a generalizable network, we need to show it samples of the different tasks while it is being evolved.

The MNIST accuracy of MNIST-Only is higher than PADAWANN's and more importantly, it is as high as the model it was evolved from while being 1.25x sparser and weight-agnostic. A significance test gives p-value of 0.1 showing there is no significant difference between the performance of MNIST-Only and MNIST-Net. This shows that our approach of evolving the network is also a feasible way to create a (partially) weight-agnostic neural network and can be used interchangeably with the one Gaier and Ha propose [11]. This also applies to the PADAWANN obtained using joint evaluation, which is even more sparse.

After applying fine-tuning, MNIST-Only appears to have a slightly worse performance than the PADAWANN on all datasets except MNIST (as seen in Table 6.6). This result indicates that evaluating the PADAWANN on multiple datasets is not needed if one's aim is to evolve a more compact MNIST classifier. However, if the goal is generalizability, the PADAWANN is superior. The lack of increase in the accuracy on the new datasets confirms our hypothesis.

		MNIST	10Letters	Another10	FashionMNIST
MNIST-Net	No TL	98.13%	6.68%	9.99%	7.82%
	TL	98.1%	58.69%	72.7%	67%
PADAWANN	No TL	93.67%	22.1%	13.79%	10.895%
	TL	96.48%	55%	67.14%	63.55%
MNIST-Only	No TL	97.95%	6.07%	10.7%	7.07%
	TL	98.04%	52.76%	66.42%	58.57%

Table 6.6: Comparison between the accuracy of the initial model, the PADAWANN and MNIST-Only on all four datasets before and after fine-tuning the last layer.

6.3.2 Optimized Mapping Between Domains

Another aspect of the joint evalution we perform is the mapping between the MNIST and 10Letters classes. In all of our experiments, it was random: We use the first 10 classes from EMNIST Letters in the order they are in (e.g. 'a' being class 0, mapping to '0' from MNIST). We investigate whether we can achieve a boost in performance if we utilize an *optimized* mapping between the two domains based on visual similarity between the classes, i.e. 'i' is visually most similar to the digit '1' and will thus have class 1. The full mapping between the 10 classes is given in Appendix A, section A.4.

The evolution of the accuracies of the model created with optimized mapping is compared to the PADAWANN's in Figure 6.2 and a further comparison between the accuracy and the spraseness is made in Table 6.7.

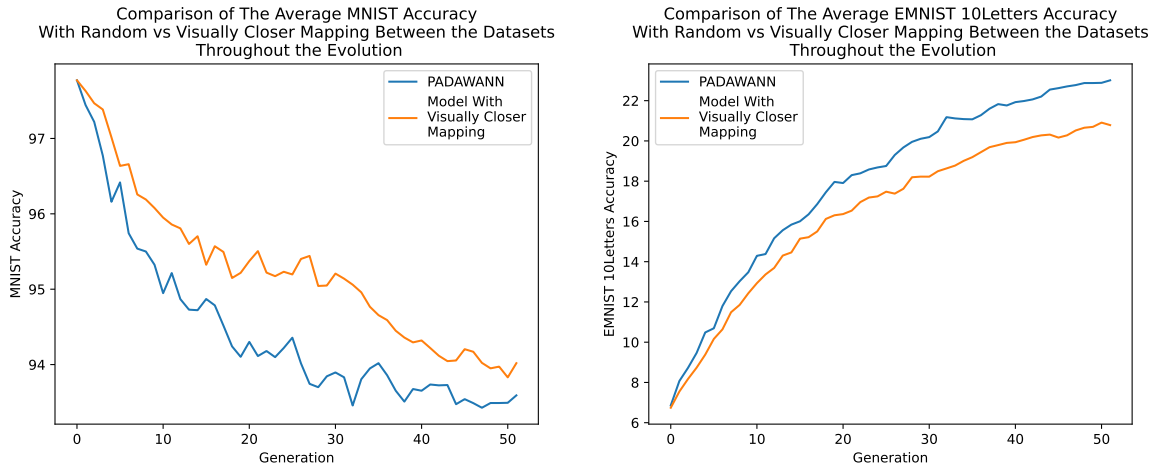


Figure 6.2: Comparison of the train accuracy evolution when the mapping between the MNIST and 10Letters domains is random and optimized.

	MNIST Accuracy	10Letters Accuracy	Loss	Number of Edges
Random Mapping	93.67% (± 0.54%)	22.1% (± 0.85%)	272.8 (± 280.56)	600460 (± 38033)
Optimized Mapping	93.84% (± 0.61%)	20.25% (± 1.13%)	732.58 (± 182.3)	626963 (± 33716)

Table 6.7: The average accuracy and sparsity of the PADAWANN when the mapping between the MNIST and 10Letters domains is random and optimized.

The MNIST accuracy is slightly higher but not significantly ($p = 0.06$), while the accuracy on 10Letters is actually significantly lower ($p < 0.01$) when using a more visually closer mapping. This indicates that the visual similarity we found within the classes is not as optimal as we expected it to be.

6.4 Pruning

To determine the role of evolutionary random pruning in the algorithm, we prune MNIST-Net in a more typical fashion and compare the resulting networks to the PADAWANN evolved from MNIST-Net. We perform unstructured pruning in two ways: random pruning, which is also the strategy utilized in the GA, and removing the weights with lowest L1-norm. Because of the randomness of the first approach, we prune the initial model 10 times to obtain an average estimate of the performance after pruning. The amount of pruned weights is equal to the amount pruned by PADAWANN in order to make a fair comparison.

Table 6.8 presents the results of this experiment. It is interesting to note that only about half the edges pruned by the two other pruning strategies are ones PADAWANN pruned as well.

	MNIST Accuracy	10Letters Accuracy	Edges Pruned In Common
PADAWANN	93.67% (± 0.54%)	22.1% (± 0.85%)	–
PNN-Random	77.27% (± 5.53%)	9.94% (± 2.27%)	304148 (51%)
PNN-L1	98.12%	6.74%	319227 (53%)

Table 6.8: The average accuracy of PADAWANN and the Pruned Neural Networks (PNNs) when using random and L1-norm pruning on the MNIST-Net model, as well as the edges pruned in common with the PADAWANN.

Random unstructured pruning greatly reduces the accuracy of the network on MNIST when there is no genetic evolution involved. That is because pruning edges at random has a high risk of removing ones that are crucial for the classification function of the model. We use the same pruning strategy in the PADAWANN-GA, but natural selection "kills" such unfit individuals. On the other hand, L1-Norm has the expected effect: It simplifies the model while the accuracy on MNIST stays the same (the difference in performance is insignificant with $p > 0.05$). However, the performance on 10Letters of the PNNs is still bad. This is further illustrated by the confusion matrices given in Figure 6.3.

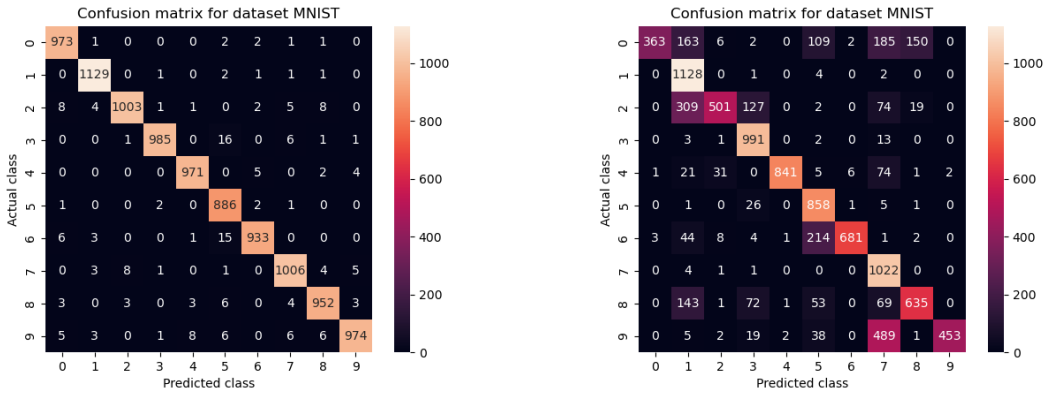


Figure 6.3: The confusion matrices of RNN-L1 (left) and PNN-Random (right) for the MNIST and 10Letters datasets.

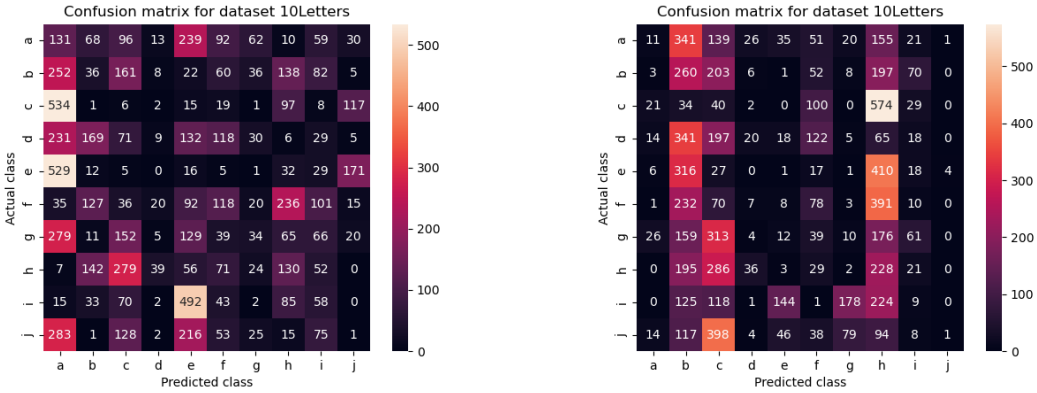


Figure 6.3: The confusion matrices of RNN-L1 (left) and PNN-Random (right) for the MNIST and 10Letters datasets.

We compare PNN-L1 to the PADAWANN in one more way, by evaluating it in a weight agnostic way (keeping the signs). We obtain 97.9% accuracy on MNIST and 6.63% on 10Letters. This shows that the combination of pruning and weight agnosticism only works when genetic evolution is involved.

Our algorithm seems to work as the best pruning technique in terms of performance on more than one task out of the examined ones. We achieve a topology that can be considered better than the initial one in terms of generalizing to other tasks. On the other hand, the typical pruning methods optimize the architecture in regards to its complexity and accuracy only on the dataset it was trained on. On the other hand, the traditional pruning methods are much faster.

Chapter 7

Discussion

In this chapter, we discuss the most important outcomes of our research, reflect on the work conducted for the thesis, and propose directions for further future research.

7.1 Sign of Weight Values

One interesting finding from our research is the importance of the sign of the weights on the performance of the pruned weight-agnostic model. If we disregard the signs when assigning the shared weight value, the network cannot converge and has random-guessing level performance. This issue arises from the fully connected layers of the networks we work with, which contain only non-zero edges. Once all edges are given the same value, the same weighted sum will arrive at all output layer nodes and the model will always predict the same class.

This is not a problem inherent to weight agnosticism; Gaier and Ha assign values with the same sign to all edges [11]. In their experiment, the genetic evolution would simply remove networks whose performance suffers because of very dense layers. Our algorithm, however, deals with networks with (some) fully-connected layers and would have to prune the majority of the edges in specifically those layers in order to improve beyond random guessing. Because of this, the evolution of a pre-trained network into a weight agnostic one appears to be a harder task than building a complex weight agnostic net from a simple one.

Another observation is that giving the pre-trained networks a shared weight value and keeping the sign of the original weights results in a model that has almost the exact same performance as the pre-trained network with its original weights. This can be seen as a confirmation of [51]'s findings that the only important information from the original initialization of a network's weights is their signs, not their weight values.

Keeping the sign produces models with higher accuracy, but has downsides as well: It means that we do not achieve full weight agnosticism because we are preserving some information about the initial model's weights. Furthermore, keeping the weight signs indicates that the resulting PADAWANN will always have a bias towards the domain it was pre-trained on.

7.2 Domain Adaptation

Our results show that the MNIST and 10Letters are too dissimilar and make the domain adaptation difficult. The GA seems to display similar behaviour when the mapping is from MNIST to 10Letters and vice versa. However, the PADAWANN evolved from MNIST-Net enjoys a higher improvement over the initial model. This indicates that, while our algorithm can use a network pre-trained on any dataset as initial model, ones trained as MNIST classifiers are at an advantage. This could be due to the relative simplicity of the dataset.

The tests with an optimized mapping between the classes of MNSIT and 10Letters show no improvement over the random mapping, which further indicates that the two domains are not strongly related. The two tasks are not similar enough for this domain adaptation to fully work. Another indicator for this is the lack of change in performance for different fitness measures. However, because of the jump in accuracy of the PADAWANNs on the unseen-before dataset, we believe that for more closely-related domains our approach could be more successful (e.g. letters in different fonts, on different surfaces/backgrounds).

7.3 "General" vs "Task-specific" Structures

The structure and performance of the models obtained through PADAWANN and through traditional pruning techniques are vastly different. As shown by our results, the usual pruning simplifies the model in a way that maintains the accuracy on the dataset it was trained on, but does not increase its generalization ability over other tasks. On the other hand, the random pruning that PADAWANN goes through results in a model that has a slightly lower accuracy on the dataset it was trained on, but a much higher accuracy on the new datasets.

The PADAWANN and the pruned models prune only about 50% of the edges in common. We believe that the increase in generalization comes from the other half and thus, our GA prunes a network into a more general one while the usual pruning techniques achieve a smaller and more efficient model, but at the cost of its generality. The difference in accuracy on 10Letters and which edges in particular are pruned could be indicative of the existence of different parts of the network that are specific to a certain task.

7.4 Weight Agnosticism

The difference between WANN and PADAWANN was not computed using a significance test, which is one of the shortcomings of our research. However, the accuracy score and sparseness of PADAWANN is comparable to the network Gaier and Ha evolve. Thus, we can conclude that the algorithm can be used for evolving a (partially) weight-agnostic network in an alternative way to the one suggested in [11]. However, it needs to start from a pre-trained neural network which already has a high performance on a certain task. Furthermore, currently our algorithm is less efficient in terms of its run-time to WANN.

Our findings show that training the network's architecture instead of weights contributes to increasing the generalization ability of the model and keeping the balance between its performance on the different tasks. That definitely raises curiosity of how useful weight agnosticism can be for neural architecture search and transfer learning in general.

7.5 Compression

Because we prune the edges and only preserve the weight-sign information, the PADAWANN is easier to store, fine-tune and use than the model it was evolved from. While space-efficiency and speed-up are not the main focus of our work, the PADAWANN we create is half the size of the initial network in terms of non-zero edges. This "by-product" makes the idea of creating a PADAWANN even more appealing.

Although PADAWANN is 2x smaller than the model it was evolved from, the drop of accuracy on the original network is quite a bit larger than what is achieved by the state-of-the-art [46, 17, 47]. When the fitness function only takes MNIST into account, the drop in accuracy is insignificant and similar to the aforementioned research results. However, the compression is negligible in comparison. This indicates that, while our algorithm can be used as a pruning method, there is room for improvement. Which can happen, for example, by allowing the fitness function to involve the complexity of the model as well.

7.6 Generalization

We show the potential of evolving a model over multiple tasks in a weight agnostic way, starting from a network pre-trained on one of them and keeping only the sign of the original weights. Our results indicate PADAWANN trades off high accuracy on a specific task with better than random-guessing accuracy on multiple. The boost in generalization ability scales well: The evolution on three datasets results in a model that has increased accuracy on both new tasks. However, as more datasets get involved, the accuracy on the original dataset drops further. We hypothesise this could be due to new features overwriting knowledge of old ones or because reusing the whole structure for all tasks is inefficient.

We confirm that the networks we evolve cannot perform well on domains they do not encode any information about (i.e. their features). This is further illustrated by the fact that PADAWANN's generalizability does not seem to extend to datasets which are not used in the evolution of the model. Because of that we can conclude that joint evaluation is a necessary component of our algorithm.

The claim made by Gaier and Ha [11] is shown to be untrue for the WANNs originating from their research. Reasons for this could be that the two domains used are not similar enough; the network's structure focuses on a single task only and finds it hard to encode information about the features of others; the algorithm needs more guidance. Our results show that combining weight agnosticism and knowledge from a pre-trained model brings WANNs closer to learning how to perform multiple tasks. Although PADAWANN is better at performing more than one task compared to the model it was evolved from and WANN, its improved performance is still barely above random-guessing. However, when transfer learning is applied on the network we observe a high accuracy of above 50% on all four datasets used for testing.

Based on the observed difference in performance of the PADAWANN before and after transfer learning, we also conclude that a network's topology cannot be made *general* so that it performs well on multiple tasks without any modifications to its structure or weights, or training the network on those datasets. This is also supported by other literature [27, 36, 51, 50] focused on transfer and multi-task learning.

7.7 Future Work

This research is conducted on only a few relatively easy tasks, because it is intended as a proof of concept. It could be the case that the combination of weight agnosticism and evolutionary random pruning only work well together for very simple data and our results do not scale well for other tasks, or the contrary: Results better than ours would be achieved if the initial model is much more complex and pre-trained on a diverse dataset such as ImageNet. Thus, further research must be conducted for other domains (images in color; containing more classes; having more sophisticated features).

Using datasets with a different number of classes is also not explored in this research, but should be investigated in the future. If the original structure of the model is to be kept, it would be necessary to apply some sort of masking in order to only use a portion of the original number of output features. Applying a strategy similar to [27] might be worth investigating. If the original structure is kept, however, the PADAWANN would not be able to generalize to datasets with a different amount of classes than the dataset the initial model was trained on. Another approach to try out would be one utilized in multi-task learning, where there is a different output layer for each dataset that can be connected to the PADAWANN's output layer depending on which task we want to work on.

This research is quite limited with respect to the number of generations and population size due to constraints on time and resources, but our results show that PADAWANN could benefit from a longer run with more individuals. What is more, fine-tuning the GA's hyperparameters and experimenting with more suitable fitness functions are potential directions for future research. Exploring the relation between population size and tournament size is also an interesting prospect. In our limited tests, we noticed that a higher tournament size is related to higher accuracies, so it is worth considering implementing the GA so that a single best fitting individual is chosen as the "parent" of the whole next generation (similarly to [40]).

The algorithm is quite time-consuming, mainly due to the evaluation procedures. Using a small subset of the train data seems to be a sweet spot, which does not result in low performance of the PADAWANN and speeds up the runtime. However, even then pruning with PADAWANN is very slow in comparison to traditional pruning like L1-norm. Improving this aspect of our algorithm could happen, for instance, by parallelizing the code. An interesting experiment to conduct would be pruning a model using L1 pruning and then using that as the initial network for the GA. It is curious whether this will lose the generalization ability that the PADAWANN normally evolves.

The WANN in [11] is also used as an ensemble which leads to slightly higher results. Investigating whether PADAWANN would also benefit from that is another potential direction for future work.

Lastly, we believe that further experiments with the weight values must be conducted. Making them agnostic is a crucial part of the algorithm, but that can be achieved in ways different from ours. For instance, sampling the shared weight value from a bigger range might lead to better results. Furthermore, setting a random value to all weights and keeping it constant throughout the whole algorithm is worth considering. Those suggestions could possibly even lead to the full utilization of weight agnosticism (i.e. discarding the signs).

Chapter 8

Conclusions

In this thesis we extend the research conducted by Gaier and Ha [11]. We test the generalization abilities of their classification WANN and find it to be sub-optimal. We propose another method for achieving a weight-agnostic generalizable model via evolution and pruning.

The PADAWANNs we create are two times sparser than the model they are evolved from and have improved generalization ability: about 2.3 – 3.3x increase of accuracy on datasets the network was not trained on. Our further ablation study shows that the boost of generalizability is indeed due to the combination of weight agnosticism, evolution and pruning a pre-trained model. Furthermore, our results indicate that evolutionary weight-agnostic random pruning can be used as a new pruning technique meaning our algorithm has dual functionality.

In conclusion, we show that the networks evolved by the baseline algorithm can not be directly reused for multiple tasks. Our proposed approach evolves PADAWANNs that are superior to WANNs, but it is still insufficient for creating networks with high accuracy over more than one task.

We hope our experimental results spark interest and inspire further work into this domain.

Appendix A

Experiment Details

The implementation of this project has been achieved fully in python, because the language supports numerous machine learning libraries that facilitate research of this type. It is all included in our GitHub repository.

A.1 Set-up

The tests were run on the compute cluster of the Science Faculty of Radboud University. The cluster makes use of the job scheduler Slurm[1]. Since we have not parallelized our code, we only use 1 node to run the algorithm on.

A.2 Libraries

We make use of the standard python library for multi-dimensional arrays and mathematical operations on them, `numpy`[2], as well as the widely used open source machine learning library PyTorch[3]. PyTorch facilitates dealing with neural networks by offering:

- A method to represent them. The `Module` class represents many types of NN layers in a convenient way, and allows to get/set the values of their weights, count the number of weights, prepare the layers for training/testing, etc. It is documented here.
- Image datasets. A list of all available datasets that are almost ready-to-use[4] is given here. PyTorch also supplies the class `DataLoader` which provides an easy way to iterate over the dataset in batches. More information about it is found here.
- Pruning functions. They allow for global and L1-norm structured and unstructured pruning (as well as other types that we do not make use of) and for randomly picking the pruned weights. They are part of the `torch` module, and are listed under section "Utilities" here.

[1] Information about Slurm can be found on this university wikia.
[2] The documentation of `numpy` can be found here.
[3] The complete documentation of PyTorch can be found here.
[4] Some preprocessing steps must be done before the datasets are used. More information about that in A.4.

A.3 Model Structure

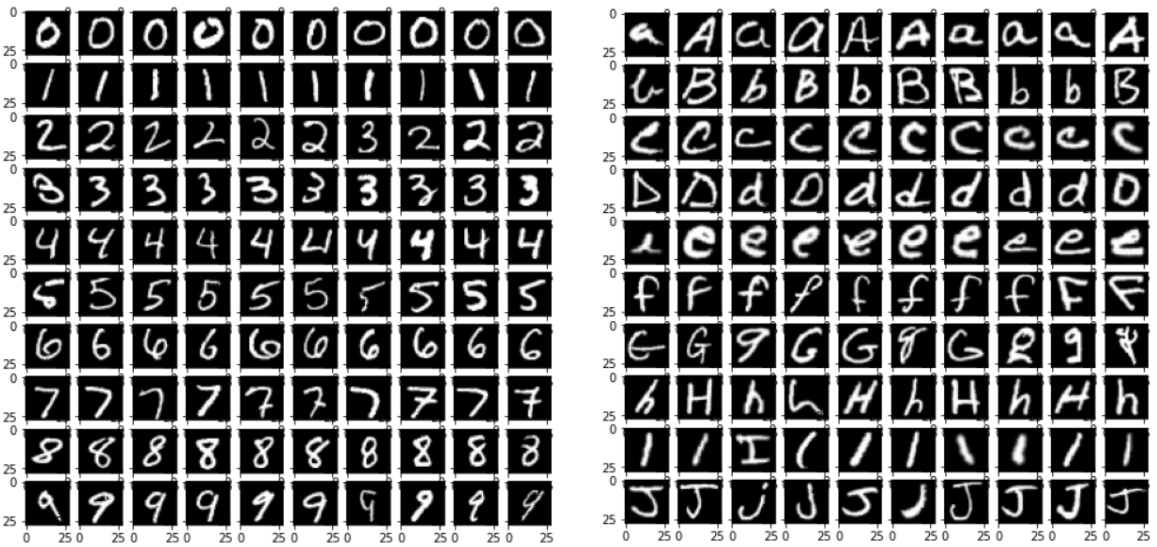
As mentioned in Chapter 5, the architecture of the initial network we used in most of our experiments is taken from the official PyTorch GitHub repository. More precisely, the model can be found here. The structure is created specifically for the MNIST dataset which is why we chose it. It is also not very complex (only six layers: two convolutional, two dropout layers and two fully-connected layers) with 1.2 million edges.

We carry out the model training using the code provided in the repository linked above, with some small modifications to make use of our custom DatasetManager class.

A.4 Datasets

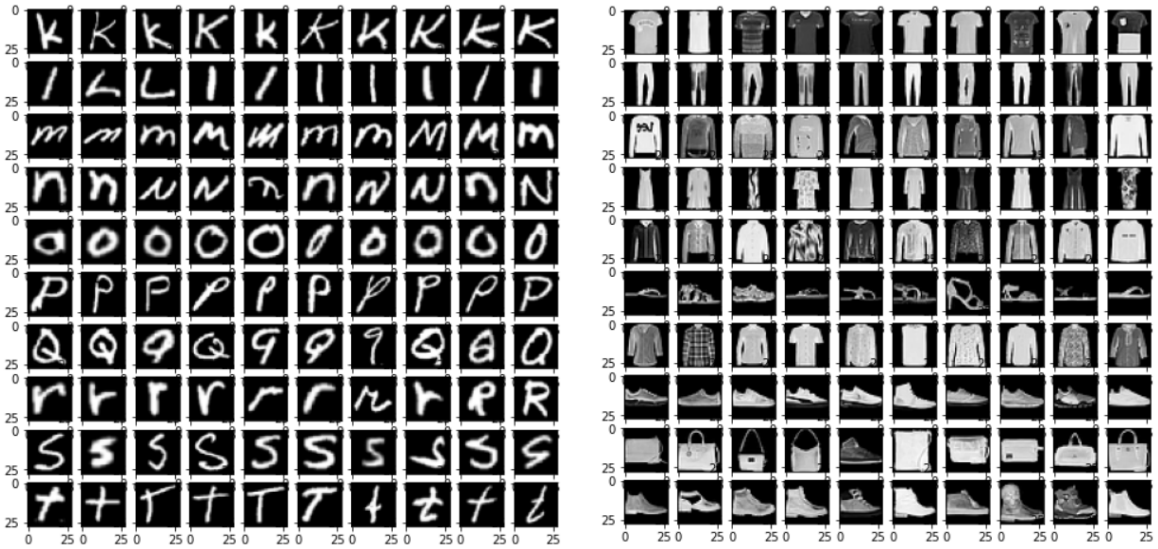
We work with the following three datasets:

- MNIST is a dataset with 70,000 Grayscale images of handwritten digits (Fugure A.1.a). It has 10 classes. More about it can be found here.
- EMNIST is a dataset with 145,600 Grayscale images of handwritten letters. It has 26 classes, but for the purpose of this research, we do not use the last 6. We create the 10Letters dataset from the first 10 classes ('a' through 'j'; Figure A.1.b), and the Another10 dataset from the next 10 classes ('k' through 't'; Figure A.2.a). More information about EMNIST can be found here
- FashionMNIST is a dataset with 70,000 Grayscale images of pieces of clothing or accessories distributed in 10 classes (Figure A.1.b). More about it is presented here.



(a) (b)

Figure A.1: Examples of the ten classes of the MNIST (a) and EMNIST 10Letters (b) datasets.



(a) (b)

Figure A.2: Examples of the ten classes of the EMNIST Another10 (a) and FashionMNIST (b) datasets.

We access the datasets directly through the PyTorch library. Since they are all Grayscale, the pre-processing step we take is minimal: normalize the images and transform them into Tensors[5] (the data type that PyTorch's neural networks work with). We modify the 10Letters and Another10 datasets further by changing their labels to match the $0 - 10$ class distribution of MNIST.

The visually closer mapping between the MNIST and 10Letters dataset that we use in one of the experiments is presented in Figure A.3.

Figure A.3: Example of the ten classes of the MNIST (top row) and 10Letters datasets (bottom) when the mapping between them is "optimized."

[5]Torch Tensors are simply multi-dimensional matrices with elements of the same type. More about them can be found here.

Appendix B

Additional Figures

This appendix contains the auxiliary plots of our experiments. The default parameters are used unless otherwise specified. The different colors represent an individual run of the GA.

B.1 PADAWANN Results

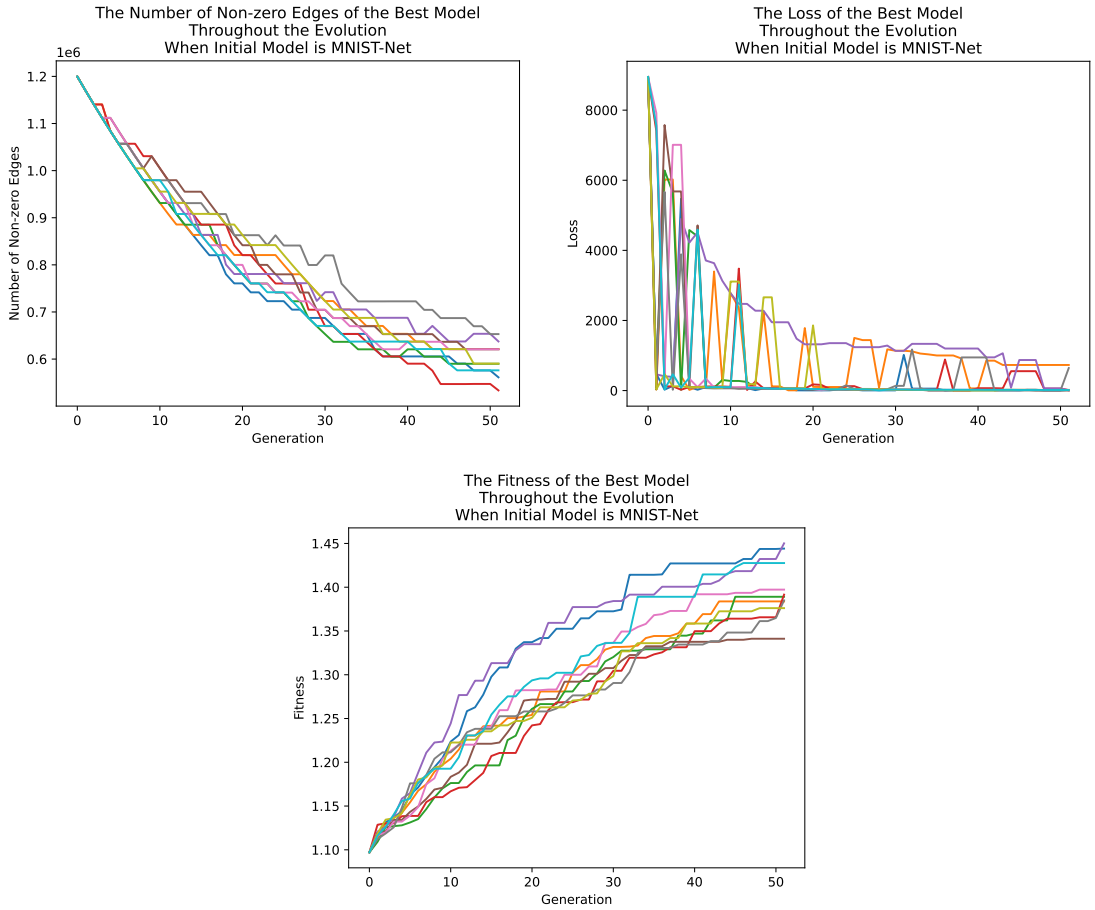


Figure B.1: The evolution of PADAWANN when evolved from MNIST-Net.

B.2 Discarding the Signs

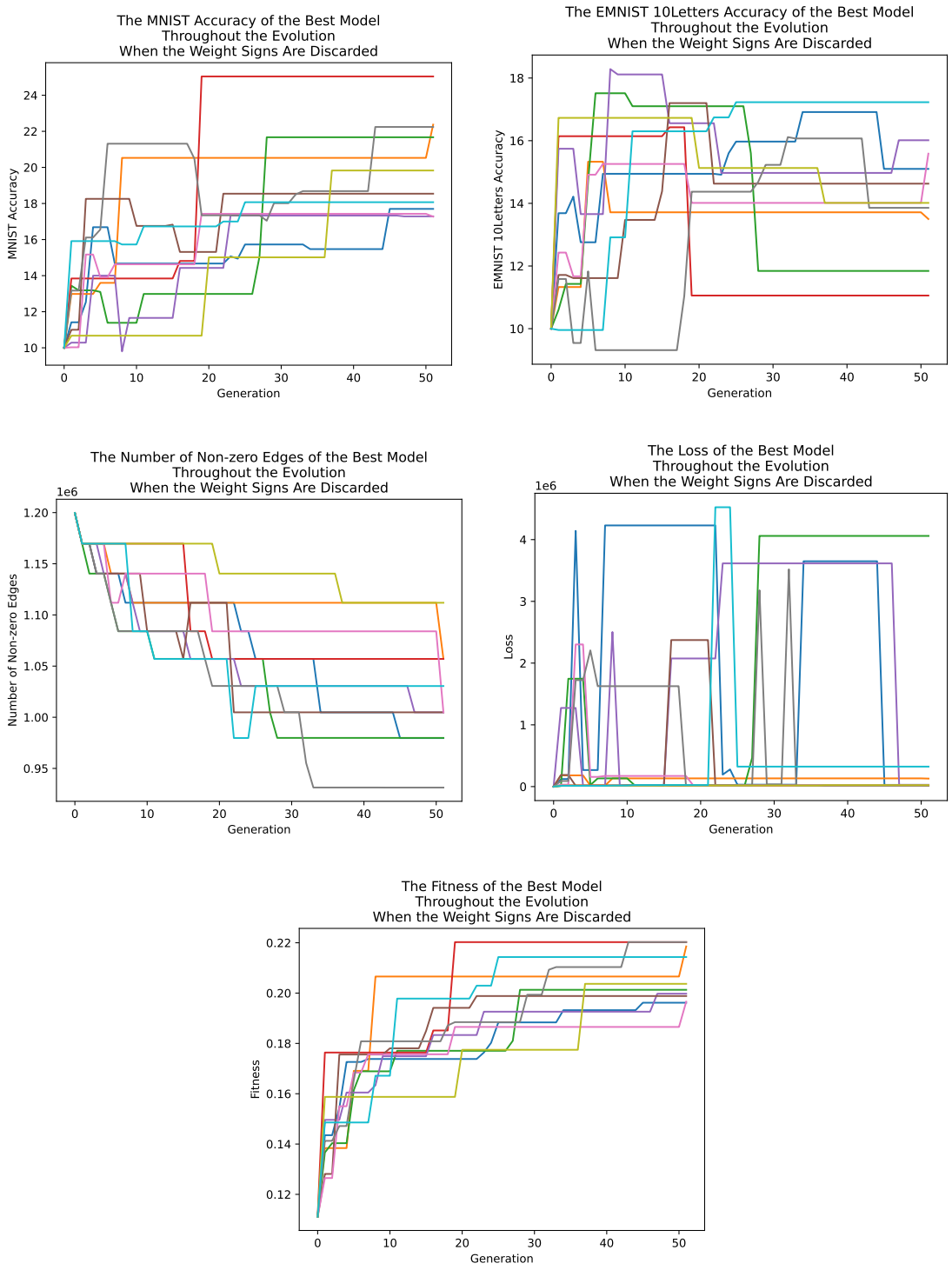


Figure B.2: The evolution of the PADAWANN's parameters and accuracy when the initial network's weight signs are discarded.

B.3 Optimized Domain Mapping

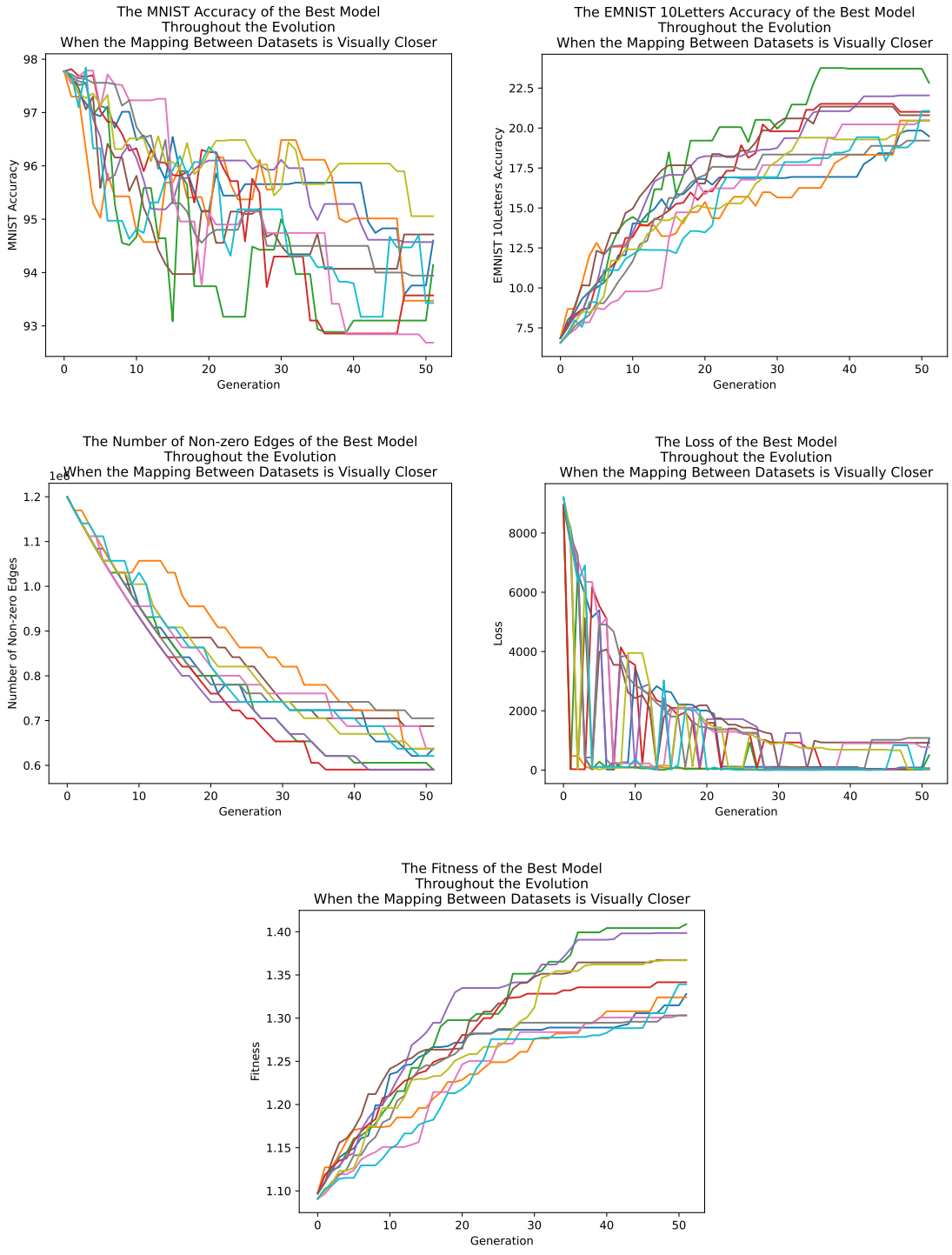


Figure B.3: The evolution of the PADAWANNs' parameters, loss and fitness when evolved from MNIST-Net with an optimized mapping between the MNIST and 10Letters classes.

B.4 3D-PADAWANN Results

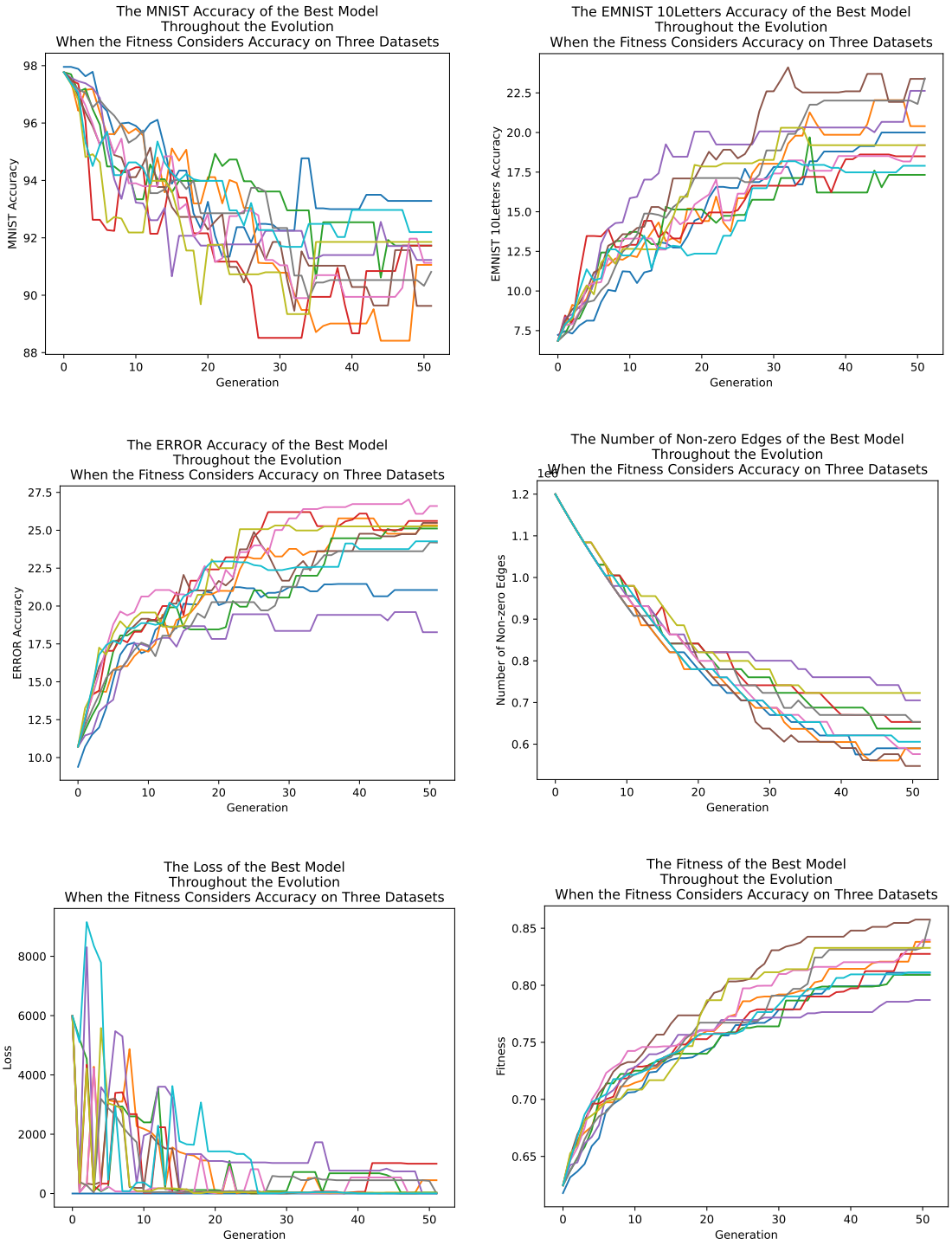


Figure B.4: The evolution of 3D-PADAWANN when the evolution considers three datasets: MNIST, 10Letters and Another10.

B.5 Non-convolutional Networks

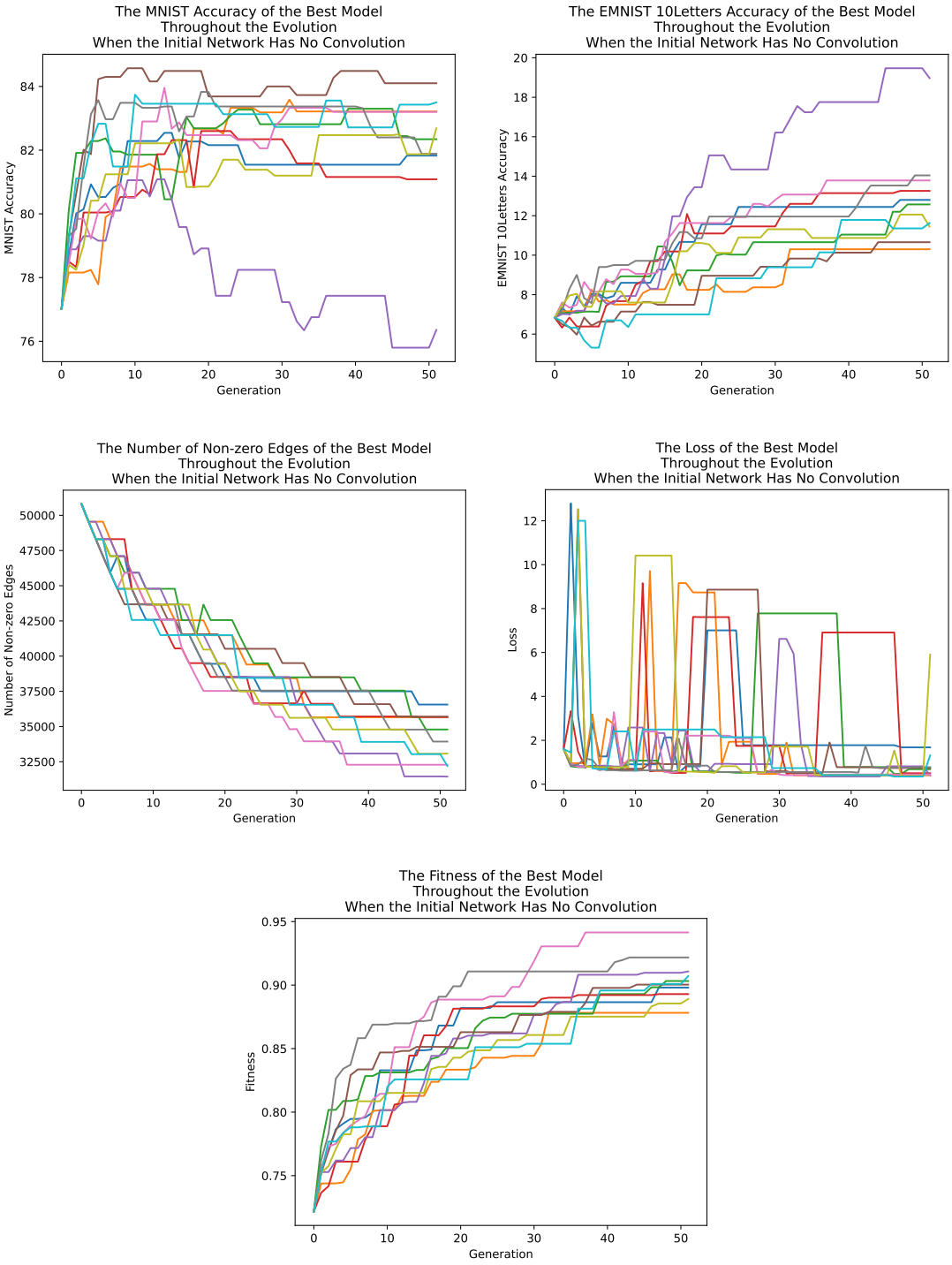


Figure B.5: The evolution of the PADAWANNs' parameters, loss and fitness when evolved from a network without convolutional layers.

B.6 MNIST-Only

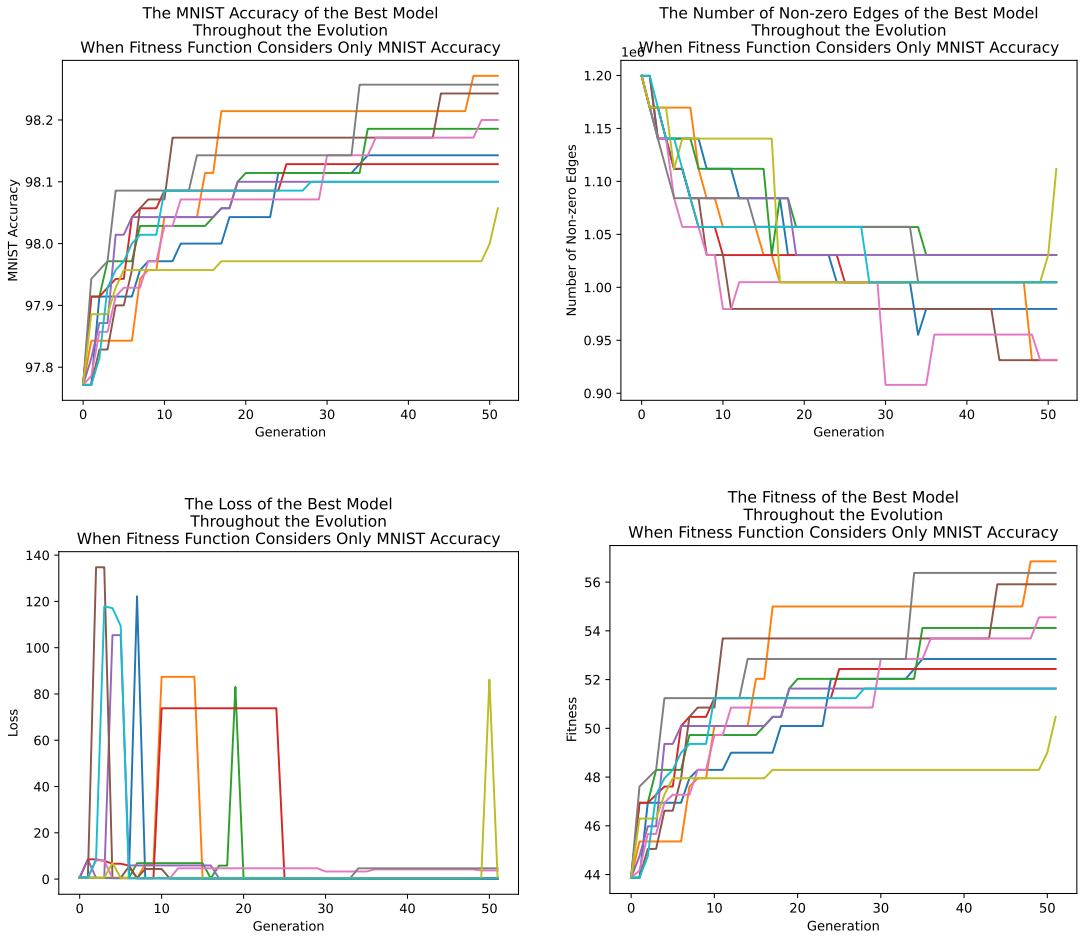


Figure B.6: The evolution of the MNIST-Only network's parameters and accuracy.

B.7 Without Weight Agnosticism

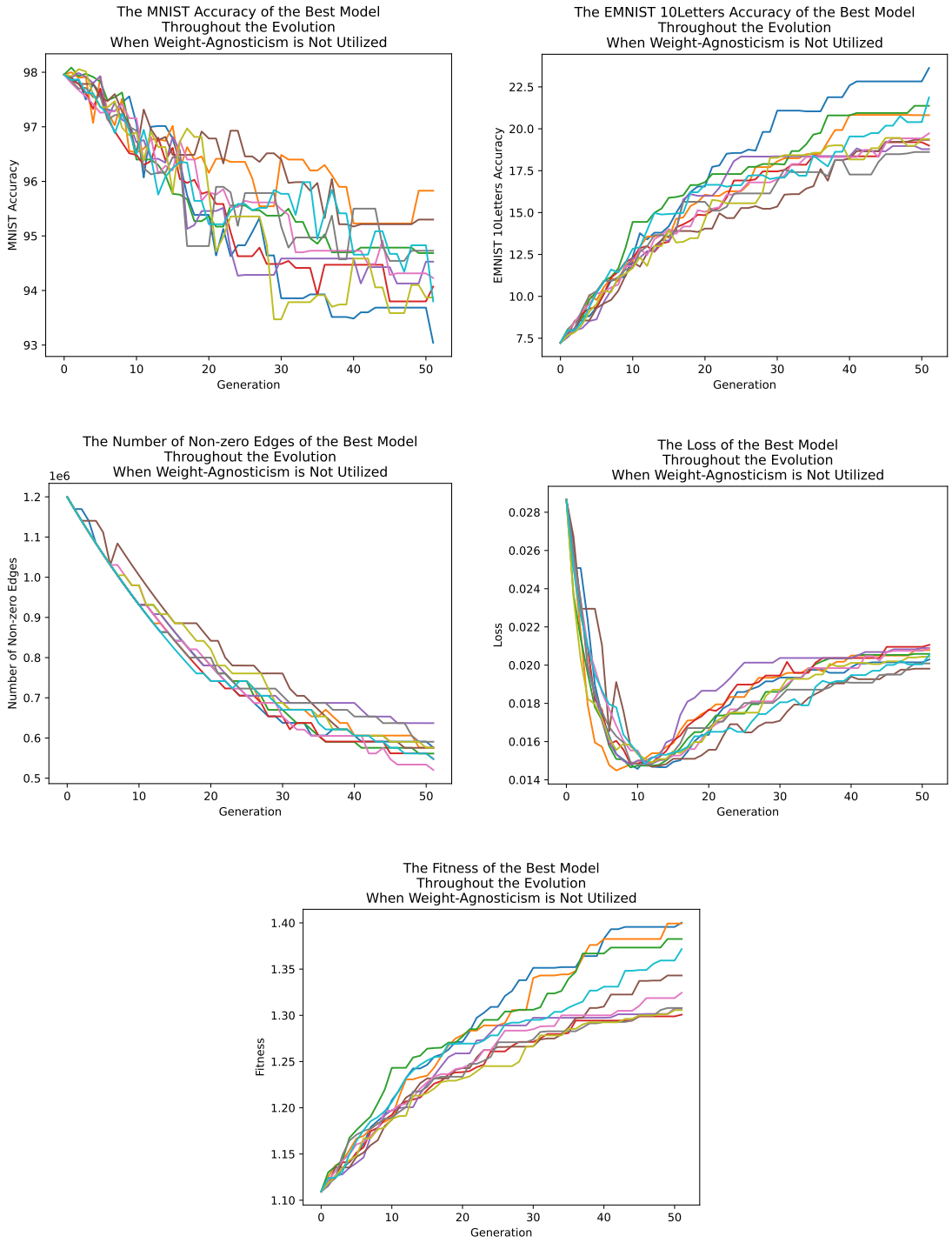


Figure B.7: The evolution of the network obtained when weight agnosticism is not utilized.

B.8 PADAWANN Evolved From 10Letters-Net

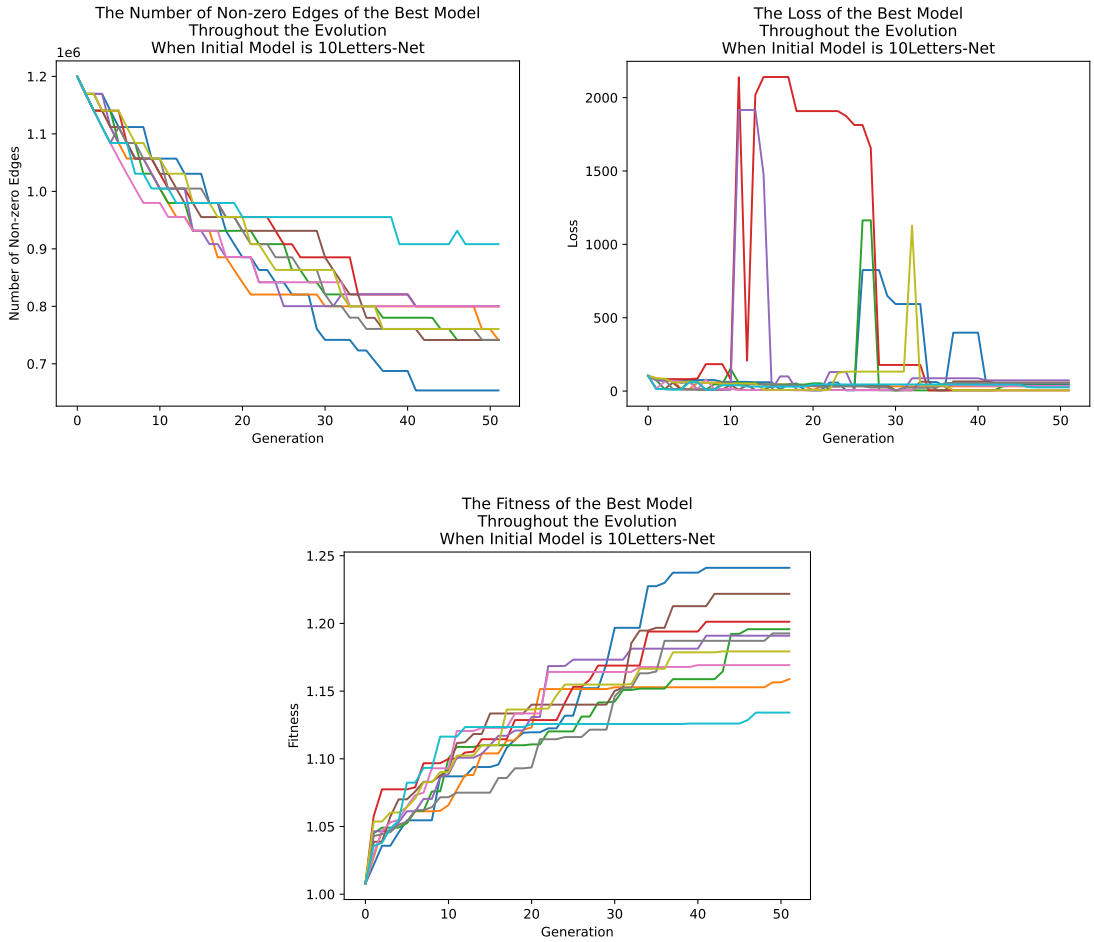


Figure B.8: The evolution of the PADAWANNs' parameters, loss and fitness when evolved from 10Letters-Net.

B.9 Hyperparameter Investigation

All figures produced from our hyperparameter investigation can be found in our GitHub repository.

Bibliography

[1] Chang Wook Ahn and Rudrapatna S Ramakrishna. Elitism-based compact genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 7(4):367–385, 2003.

[2] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):1–18, 2017.

[3] George Bebis and Michael Georgiopoulos. Improving generalization by using genetic algorithms to determine the neural network size. In *Proceedings of Southcon'95*, pages 392–397. IEEE, 1995.

[4] Armando Blanco, Miguel Delgado, and MC Pegalajar. A genetic algorithm to obtain the optimal recurrent neural network. *International Journal of Approximate Reasoning*, 23(1):67–83, 2000.

[5] Miguel A. Carreira-Perpinan and Yerlan Idelbayev. "learning-compression" algorithms for neural net pruning. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8532–8541, 2018.

[6] Richard Caruana. Multitask learning: A knowledge-based source of inductive bias. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 41–48. Morgan Kaufmann, 1993.

[7] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 160–167, New York, NY, USA, 2008. Association for Computing Machinery.

[8] S Deepak and PM Ameer. Brain tumor classification using deep cnn features via transfer learning. *Computers in biology and medicine*, 111:103345, 2019.

[9] Oana Diana Eva and Anca Mihaela Lazar. Comparison of classifiers and statistical analysis for eeg signals used in brain computer interface motor task paradigm. *International Journal of Advanced Research in Artificial Intelligence (IJARAI)*, 1(4):8–12, 2015.

[10] Zhenyu Fang, Jinchang Ren, Stephen Marshall, Huimin Zhao, Song Wang, and Xuelong Li. Topological optimization of the densenet with pretrained-weights inheritance and genetic channel selection. *Pattern Recognition*, 109:107608, 2021.

[11] Adam Gaier and David Ha. Weight agnostic neural networks. *arXiv preprint arXiv:1906.04358*, 2019.

[12] Rosa Maria Garcia-Gimeno, Cesar Hervas-Martinez, and Maria Isabel de Siloniz. Improving artificial neural networks with a pruning methodology and genetic algorithms for their application in microbial growth prediction in food. *International Journal of Food Microbiology*, 72(1-2):19–30, 2002.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[14] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. *arXiv preprint arXiv:1608.04493*, 2016.

[15] Dongmei Han, Qigang Liu, and Weiguo Fan. A new image classification method using cnn transfer learning and web data augmentation. *Expert Systems with Applications*, 95:43–56, 2018.

[16] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[17] 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.

[18] Peter JB Hancock. Pruning neural nets by genetic algorithm. In *Artificial Neural Networks*, pages 991–994. Elsevier, 1992.

[19] Douglas M Hawkins. The problem of overfitting. *Journal of chemical information and computer sciences*, 44(1):1–12, 2004.

[20] Cristian Ivan and Razvan Florian. Training highly effective connectivities within neural networks with randomly initialized, fixed weights. *arXiv preprint arXiv:2006.16627*, 2020.

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

[22] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

[23] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[24] 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.

[25] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019.

[26] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[27] Arun Mallya, Dillon Davis, and Svetlana Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 67–82, 2018.

[28] Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

[29] Masaya Misaki, Youn Kim, Peter A Bandettini, and Nikolaus Kriegeskorte. Comparison of multivariate classifiers and response normalizations for pattern-information fmri. *Neuroimage*, 53(1):103–118, 2010.

[30] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[31] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911. PMLR, 2017.

[32] R. Reed. Pruning algorithms-a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.

[33] Fabian Ruehle. Evolving neural networks with genetic algorithms to study the string landscape. *Journal of High Energy Physics*, 2017(8):1–20, 2017.

[34] Manali Shaha and Meenakshi Pawar. Transfer learning for image classification. In *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, pages 656–660. IEEE, 2018.

[35] Zhenghao Shi, Huan Hao, Minghua Zhao, Yaning Feng, Lifeng He, Yinghui Wang, and Kenji Suzuki. A deep cnn based transfer learning method for false positive reduction. *Multimedia Tools and Applications*, 78(1):1017–1033, 2019.

[36] Hoo-Chang Shin, Holger R. Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura, and Ronald M. Summers. Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE Transactions on Medical Imaging*, 35(5):1285–1298, 2016.

[37] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[38] SN Sivanandam and SN Deepa. Genetic algorithms. In *Introduction to genetic algorithms*, pages 15–37. Springer, 2008.

[39] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[40] Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the genetic and evolutionary computation conference*, pages 497–504, 2017.

[41] Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G Yen. Completely automated cnn architecture design based on blocks. *IEEE transactions on neural networks and learning systems*, 31(4):1242–1254, 2019.

[42] Hans Henrik Thodberg. Improving generalization of neural networks through pruning. *International Journal of Neural Systems*, 1(04):317–326, 1991.

[43] Darrell Whitley et al. Genetic algorithms and neural networks. *Genetic algorithms in engineering and computer science*, 3:191–201, 1995.

[44] Jiansheng Wu, Jin Long, and Mingzhe Liu. Evolving rbf neural networks for rainfall prediction using hybrid particle swarm optimization and genetic algorithm. *Neurocomputing*, 148:136–142, 2015.

[45] Lingxi Xie and Alan Yuille. Genetic cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1379–1388, 2017.

[46] Chuanguang Yang, Zhulin An, Chao Li, Boyu Diao, and Yongjun Xu. Multi-objective pruning for cnns using genetic algorithm. In *International Conference on Artificial Neural Networks*, pages 299–305. Springer, 2019.

[47] Wenzhu Yang, Lilei Jin, Sile Wang, Zhenchao Cu, Xiangyang Chen, and Liping Chen. Thinning of convolutional neural network with mixed pruning. *IET Image Processing*, 13(5):779–784, 2019.

[48] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014.

[49] Ruizhe Zhao and Wayne Luk. Efficient structured pruning and architecture searching for group convolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[50] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(6):1452–1464, 2018.

[51] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. *Advances in neural information processing systems*, 32, 2019.