2020-2021

Master thesis: MEMO-F-524

Neural Architecture Search: How to compromise between Time and Accuracy

Defraene Pierre

Advisor : Professor Hugues Bersini, Co-Advisor: Antonio Garcia Diaz

2021-2022

Abstract

With the evolution of the artificial intelligence and of the deep learning, A new field become interesting to explore. The neural architecture search become very interesting. Indeed, find the best architecture for a neural network is quite complex and considering that the search space is very large, it takes quite a long time. What we are working on this master thesis is reduce the time of this search of architecture to make it more reasonable even for a private computer with only 1 GPU. To do so, we combine an cartesian genetic algorithm and a predictor for the accuracy of these network together. Furthermore, we also developed an new fitness function taking into account the number of nodes in the network that we call Accuracy and Size (AAS). With the combination of these methods, we obtain quite interesting results in accuracy and very low time of execution for the algorithm close to only 1 hour of experiment.

0.1 Abbreviations and Acronyms

AAS : Accuracy and Size

AI : Artificial Intelligence

ANN : Artificial Neural Network

ANOVA : Aalysis of Variance

CGP : Cartesian Genetic Programming

CNN : Convolutional Neural Network

FCNN : Fully Connected Neural Network

FFNN : Feedforward Neural Network

GA : Genetic Algorithm

ML : Machine Learning

MLP : Multi-Layer Perceptron

NAS : Neural Architecture Search

NN : Neural Network

ReLU : Rectified Linear Unit

RL : Reinforcement Learning

RNN : Recurrent Neural Network

Contents

1 Introduction

In this world full of new technologies, there is a large expansion of Artificial Intelligence (AI) in our daily life. For example, AI-Based Digital Assistants like Google and Siri [Mae+19], self-driving cars [Boj+16], bots playing games like go in which the champion was beaten by a AI [Sil+16] or chess and shogi [Sil+18], image recognition like in medical image analysis [FK19], etc are examples of the use of AI. AI is the theory and development of a computer system which try to solve some problems or perform some tasks that need normally human intelligence [McC07].

In our case, we are interested by the subset of AI that is Machine Learning (ML). ML is more specific than AI by the fact that it learns to perform specific tasks by learning automatically through experience [JM15]. More precisely, we focus on deep learning [LBH15] and Neural Networks (NNs) [Zup94]. They are interesting considering that with the evolution of computers and algorithms, we are now able to have NNs with a equal or better results than humans for different tasks. For example, in recognition of disease like Alzheimer's disease [KYK19], recognition of writing by hand [HA19], in dental image diagnostics [Sch+19] etc... Considering that we focus on image recognition, we are interested in three different datasets which are CIFAR-10, CIFAR-100 [Kri12] and SVHN[Net+11]. We are mainly interested in Convolutional Neural Networks (CNN) [AAA17] since they are obtaining the best results in image recognition.

However, they are computationally intense and because of that we are always trying to find a good compromise between good results, complexity and computation time. To do so, we are using different algorithms to try to optimize automatically the large number of parameters of the NNs and also the topology of it. This field is called Neural Architecture Search (NAS) [Ren+21]. This field is interesting considering that we can reduce the complexity and the time consumption of the search by prioritizing what we want from the NNs. It is really important considering that the search space is really too large.

To do so, we looked at a lot of heuristic algorithms that could be used to find the best parameters considering that the task is quite complex and it is impossible to find the best solution. In our case, we will focus on evolutionary algorithms [Rea+17] [Mii+17] and more precisely Genetic Algorithms (GA) [XY17] but it exists a lot of algorithms which could be studied to optimize NNs like particle swarm optimization[FY19], microcanonical annealing algorithm[Ayu+16]etc... We will use Cartesian Genetic Programming (CGP) which is a form of GA to find automatically one of the best topology or parameters for CNNs.

In this work, we will be interested in the CGP from Suganuma as base of research [SSN17] [Sug+20]. This algorithm is interesting to go over the search space and to represent easily the CNN that we are working on. We want to try different fitness function in this CGP to be able to make comparisons about their performance (accuracy, complexity, speed). That means, we want to be able to approximate the accuracy of different NN without the need of training completely every NN. To do the prediction of accuracy, we were interested in a predictor called E2EPP [Sun+20a]. We developed also a method called Accuracy and Size(AAS) to prioritize a NN which has a lower size over another NN which has the same accuracy but a bigger size. So, in this work, we combine these 3 methods together (CGP, E2EPP and AAS) to try to obtain automatically some architectures of CNN interesting without a need to run the experiments for too much time.

2 Background

2.1 Deep Learning and Important concepts

The deep learning is a subset of the ML which is specialized in learning with complex data like image,sound,etc... It is used in image recognition, speech recognition, natural language processing, etc... [LBH15]. The deep learning model, the most known, is the Artificial Neural Networks (ANN) but there are a lot of model like the deep Boltzmann machine [Rav+16]. The adjective "deep" refers to the large number of layers used in these models. We will now explain more about NNs and try to show some important concept about these tools.

2.1.1 Artificial Neural Network

An ANN or most of the time called NN is a computational model which is inspired by the human brain and more precisely by the biological neural network [Zup94]. It is composed of nodes structured in different layers and relied by some weighed edges. For the most simplistic structure, it is composed of an input layer, of some hidden layers or intermediary layers and of an output layer. It learns by training with data [ZAS14].

2.1.2 Batches and epochs

The input data in the NN are not sent into the system in one big block, it is split into different chunks of equal size called batch. The batch represent the number of sample to go through before updating the weight. The epoch represent the number of time, we need to go through all batches. All the batches are trained for one iteration in both forward and back propagation and that is an epoch. This means 1 epoch is a single forward and backward pass of the entire input data [Bro18].

2.1.3 Hyper-parameters

Hyper-parameters are the variables that regulate the network structure and the variables which govern the learning process [YZ20]. So, They are what we are interested when we try to optimize NN considering that they are the parameters which change the learning curves. hyper-parameter optimization is a very big field when working with NN. There are a lot of hyper-parameters in NN but even more in CNN. This is what make the optimization very complicated. The learning rate, the number of epochs, the size of the batch , the number of layers are example of hyper-parameters in simple NN [Smi18]. With CNN, there are also the size of the filter, the stride and the padding.

2.1.4 Trainable parameters

The trainable parameters or learnable parameters are the parameters which are adapted during the training [Dee18]. For example, the weights and biases. They are very important considering that they are directly related to the time needed to train a CNN. For a complete network, they are calculated by summing the number of parameters per layer.

2.1.5 Underfitting and Overfitting

In ML, we need to be careful about how we learn in our model considering that we can fall into an underfitting or overfitting problem [Koe18] [Smi18].

Underfitting is the case where the model has not learned enough from the training data, resulting in low generalization and bad predictions. The model has not learned enough features from the data and even if he can have good results with the training set, he will have pretty bad results for the test set.

Overfitting is the case where the model has learned too much. That means, it have also learned the noise in the data. It will learns features which are not really features that we wanted to take into account. It can recognize from his training set very well, but if we take new input, it won't be able to have good results because of the noise which was learned.

Therefore, They are 2 states that we want to avoid at any cost to be able to have good performance.

2.1.6 Cross-Validation

Cross-validation is a very useful technique for assessing the effectiveness of our model, particularly in cases where we need to mitigate overfitting . Obviously, there are more than one validation method possible like the Leave-one-out cross-validation, k-fold cross-validation and Repeated k-fold cross-validation [RTL09]. In the k-fold cross validation, we are starting by shuffling the data. Then, we divide it into k equal parts. For every block, we will use it for the test set and every others for the training set. So, every block will be used in the training set k-1 times and 1 time for the test set. To calculate the result, we will calculate the mean of the model skill scores of the k variations.

2.1.7 Learning rate

The learning rate is a hyper-parameter in the NN that control how much to adjust the model in response to the estimated error each time the model weights are updated [Smi18]. His value needs to be carefully chosen considering that if it is set too low, the model will train very slowly and if it is set too high, this causes undesirable divergent behavior to the loss function due to drastic updates in weights, and it may fail to converge.

2.1.8 Backpropagation

The backpropagation is a mechanism to update the weigh of the transition in the NN by taking into account the result of the previous steps. Indeed, when we create a neural network, we assign random weights and bias values to our nodes. Once we have received the output for a batch of the input data, we can calculate the error. This error is then fed back to the network to update the weights of the network. These weights are then updated so that the errors in the subsequent iterations are reduced.

2.1.9 Activation Function

They are also using some activation function applied at the output of the neurons to allow a neuron to be active or not according to his input. It introduce some non-linearity in the output of the neurons [SSA17]. It is needed considering that without non-linearity, the models wouldn't have the ability to learn and recognize complex mappings from data. We will show some examples of these function:

ReLU

The Rectified Linear Unit (ReLU) activation function has been the most widely used for deep learning applications with state-of-the-art results. It usually achieves better performance and generalization in deep learning compared to the sigmoid activation function. It is a very simple function which transform negative value into 0. It follows the equation 2.1 and the graph 2.1:

$$f(x) = max(0, x) \tag{2.1}$$

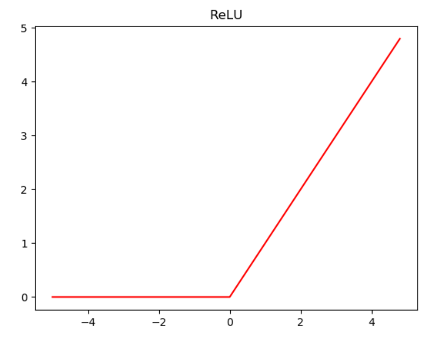


Figure 2.1: ReLU function

Sigmoid

Another common activation function is the sigmoid, the sigmoid transformation generates a smooth range of values between 0 and 1. It follows the equation 2.2 and follow the graph 2.2 :

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.2}$$

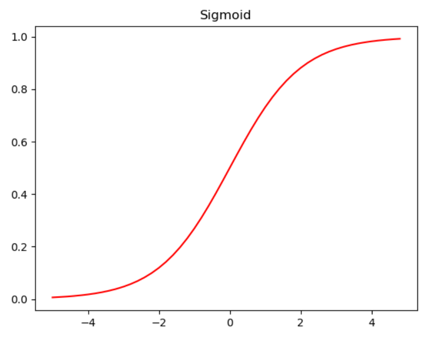


Figure 2.2: Sigmoid function

Tanh

The hyperbolic tangent function (tanh) is another function close to the sigmoid but the output value is between -1 and 1. The advantage of that function is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero. It follows the equation 2.3 and the graph 2.3:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.3}$$

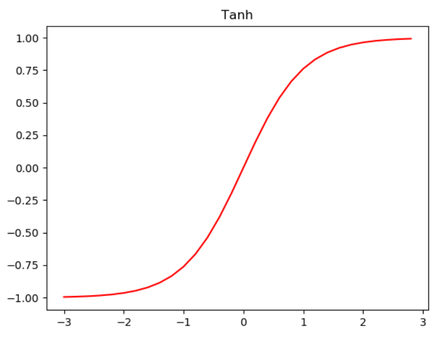


Figure 2.3: Hyperbolic Tangent function

Softmax

There exist also some function which takes as input a vector and which output also a vector like the Softmax function. Every element output is also between 0 and 1 and the sum of these elements is equal to 1. The Softmax function is used for prediction in multi-class models considering that it returns probabilities of each class. The calculated probabilities are then helpful in determining the target class for the given inputs. It follows the equation 2.4:

$$f(x) = \frac{e_i^x}{\sum_{j=1}^{J} e_j^x} \text{ for i = 1,2,3,..,J} \tag{2.4}$$

2.2 Different Type of Neural Networks

There are a lot of different NNs, We will introduce some of these special NNs.

2.2.1 FeedForward Neural Network

A FeedForward Neural Network (FFNN) is a NN which can only go forward, it doesn't have loop in the contrary of Recurrent Neural Network (RNN) [Saz06]. It is the first NN designed,the simplest and the most common. We can see that this structure is quite simple in the figure 2.4:

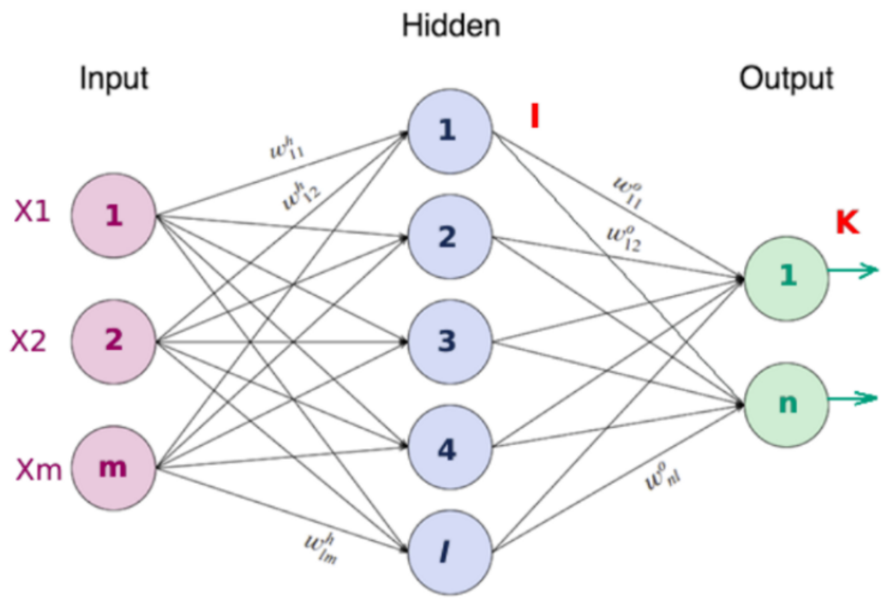


Figure 2.4: FFNN topology from [Ima20]

As we can see in this image, neurons are weighted sums. They are considering the previous inputs and the weight of the edges to be calculated. Furthermore, these weighted sums pass into an activation function to obtain the output of the neurons.

2.2.2 Recurrent Neural Network

A RNN is a NN which can have loops and going backward [Bod02]. They are distinguished by the fact that they use "memory" by taking information from precedent inputs to influence the current input and output. These algorithms are commonly used for ordinal or temporal problems like language translation, speech recognition, language processing and image captioning. We can see that the difference of structure compared to the FFNN in the figure 2.5:

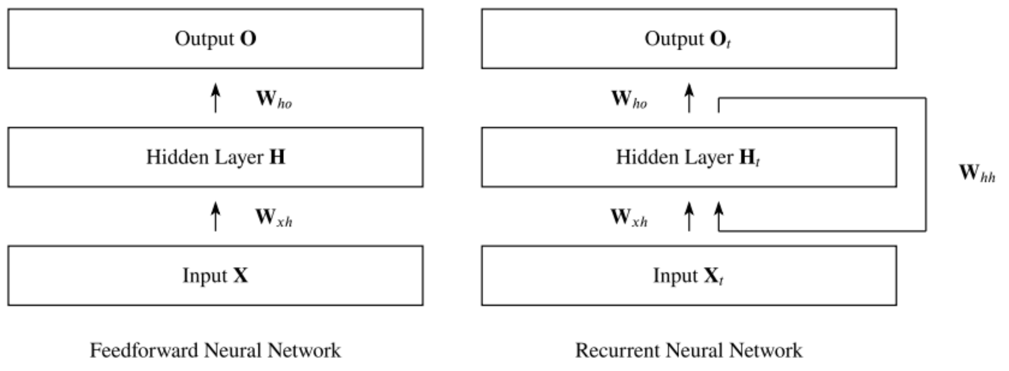


Figure 2.5: FFNN and RNN differences [Sch19]

We can clearly see the differences with the output of the hidden layer used as an input for himself. They can also be more complex as in a deep RNN [Pas+13].

2.2.3 Fully Connected Neural Network

A Fully Connected Neural Network (FCNN) is a FFNN which every layer is fully connected layers that means that every node of this layer is connected to all nodes of the next layer. These types of networks are computationally intense and be induced to overfitting. Because of that, they are not so much used, but we used some fully connected layers in some CNN [Bas+20]. The CNNs are preferred over FCNN because of the numbers of nodes which grow exponentially in FCNN is not adapted in image classification. On the contrary, considering that CNNs use fully connected layers only at the end and after some reduction of size, CNNs are more interesting in this sense.

2.2.4 Convolutional Neural Network

The CNNs are a very interesting structure and lot of people introduced them[AAA17] [ON15] [Li+21]. The CNNs are FFNNs designed to process data from multiple array. Signals, sequences, language, images or audio spectrograms and video or volumetric images are example of these data. We will focus on image recognition in this explication. CNNs are looking at piece of the image called features and by finding close features roughly to the same position in two images, CNNs become better at finding similarity than a total match for the image [Ale20] [Ale21]. Each feature is a part of the image and CNN try to match them independently to find the best match. They are split into 2 big parts. The first part is the feature learning which is composed of a certain number of convolution and pooling layers. The second is the classification which is more classical with the flattening and the fully connected layers. We can see that in the figure 2.6:

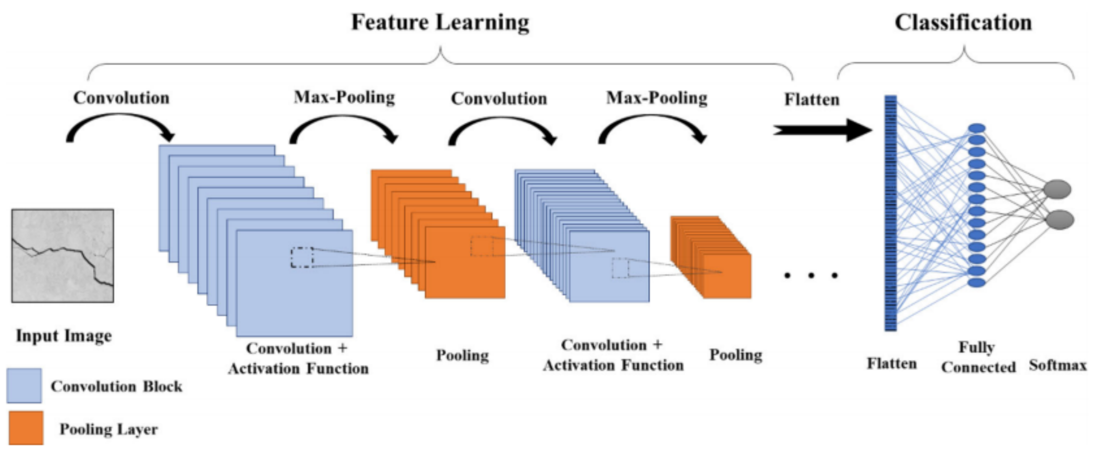


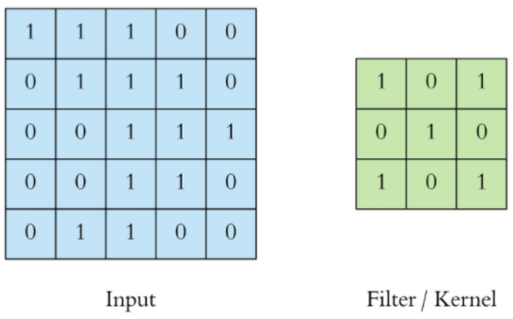
Figure 2.6: CNN topology from [Kim+21]

We can go now into more details for all the different special layers:

Convolution layer

This layer creates a feature map by using the convolution operation to find the feature in another position in the image. The convolution operation is taking a filter which is a feature, it compares the image by using a moving window. With this moving window and the filter, we make an element wise multiplication and write the sum of the result in a feature map. This feature map will show where the feature appears the most in the image. There is two important hyper-parameters in this operation, the filter (F) which denotes the size of the window and the stride (S) which denotes how many steps we are moving in each times. We could also discuss about padding considering that the filter doesn't fit always perfectly the image. There are two big options for the padding, padding the image with zeros or drop the part of the image which doesn't fit. The convolution operation is greatly used in image analysis to perform some operation like edge detection, blur and sharpen by using some special filter.

We will show a simplified example with a predetermined filter (3x3) and the stride of value 1. Firstly, the input and the filter are following the figure 2.7:



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input

1	0	1
0	1	0
1	0	1

Filter / Kernel

Figure 2.7: Input and filter for convolution from [Der17]

Considering that we have a stride of 1, the window will move of 1 pixel to the right each time and sum the result of his multiplication in the feature map like in the figure 2.8:

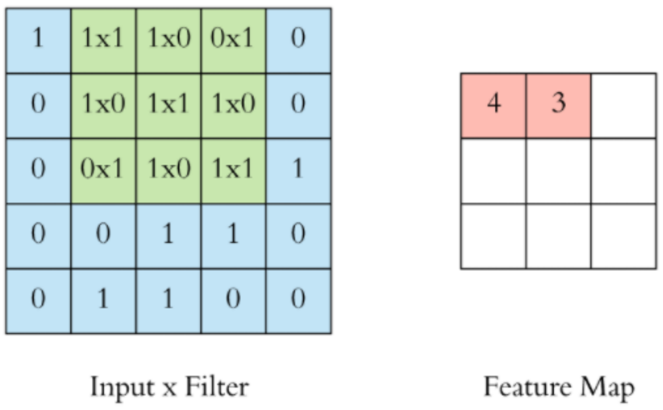


Figure 2.8: Example of Convolution operation from [Der17]

After the convolution operation, Rectified Linear Unit or ReLU is necessary to break up the linearity that we add through the convolution operation and add non-linearity. This unit transforms every negative number of the feature map into 0.

Pooling layer

The pooling layer is shrinking the image while preserving the most information about them. To do so, we use a window which will move in the feature map. In this window, we will take every value and reduce them to one value with different methods like the sum of every values or the average of every values or the maximum value. We need this layer between the convolution layer and the fully connected one to limit the number of nodes needed in the fully connected layer and decrease the computational load. An example with the max-pooling operation can be seen in the figure 2.9:

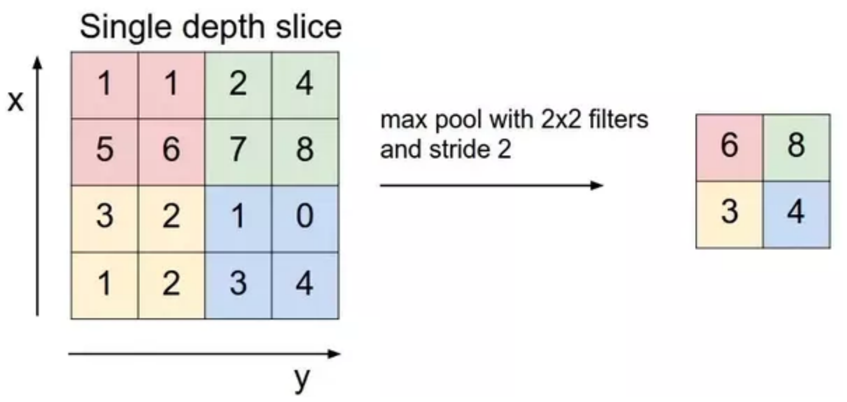


Figure 2.9: Example of max-pooling operation from [Pra19]

Fully connected layer

This layer is taking the high level filtered image and transform them into votes. Every vote must be taken into account so we need a fully connected layer to give it to the output layer and the output layer can do his final choice.

2.3 Neural architecture search

Given the evolution of deep learning, it has been proven that the architecture of the NNs is very impactful on the performance [Ren+21]. However, there is a limit of human knowledge about the vast possibility of architecture. So, reduce the human intervention as much as possible and let the algorithms automatically design the NNs became an very interesting idea. This idea is known as NAS. The optimization strategy of NAS is quite simple as we can see in the figure 2.10.

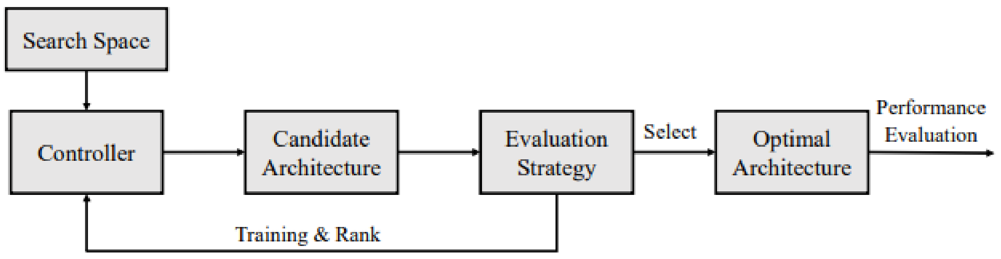


Figure 2.10: General algorithm for NAS [Ren+21]

The basic component of the different algorithms used in NAS are the search space, the search strategy and the evaluation strategy. We will define a bit more these component and how they are important.

First of all, the search space is all the candidate solutions of architecture that can possibility be interesting. It need to be chosen with attention considering that we can't really look at all the possibility especially when the search space is large like in the CNNs.

Secondly, the search strategy is how we are going to go through the search space. It is really important considering that we need to start somewhere in this search space full of possibility and then go through others possibility in a ways different in each experiment to be able to look at a lot of different possibility.

Lastly, the evaluation process of the solution is quite important. Indeed, If we evaluate every NN by training it completely, the time needed to the algorithm become very large. Considering that, we have started to try to evaluate the accuracy of the NN without the need to train them completely.

There is a lot of algorithms used for NAS, in our case, we will be interested more precisely in the subset of evolutionary algorithm which is GA.

2.3.1 Genetic Algorithm

A GA is some algorithm inspired by the biology and more precisely by the natural selection of Charles Darwin [Deb99] [SD08]. The natural selection is the fact that in a population, the individuals which are the more suited to the environment will survive and pass their genes to the next generation. They are used to generate heuristic solutions to optimization and search problems by using some operators from the biology (mutation, crossover and selection). In our case, we will be interested in the use of these algorithms in the optimization of CNNs and more precisely in NAS. To be able to perform, a GA requires a simple representation which permits a proper use of the operators in our case mostly a binary representation of the solution and a fitness function to evaluate each individual of the population. It consists of the constructions of a generation of individuals (possible solution) and using genetic operations to allow them to evolve and moving towards better solutions at each new generation. We can have a better understanding of this algorithm by looking at the figure 2.11.

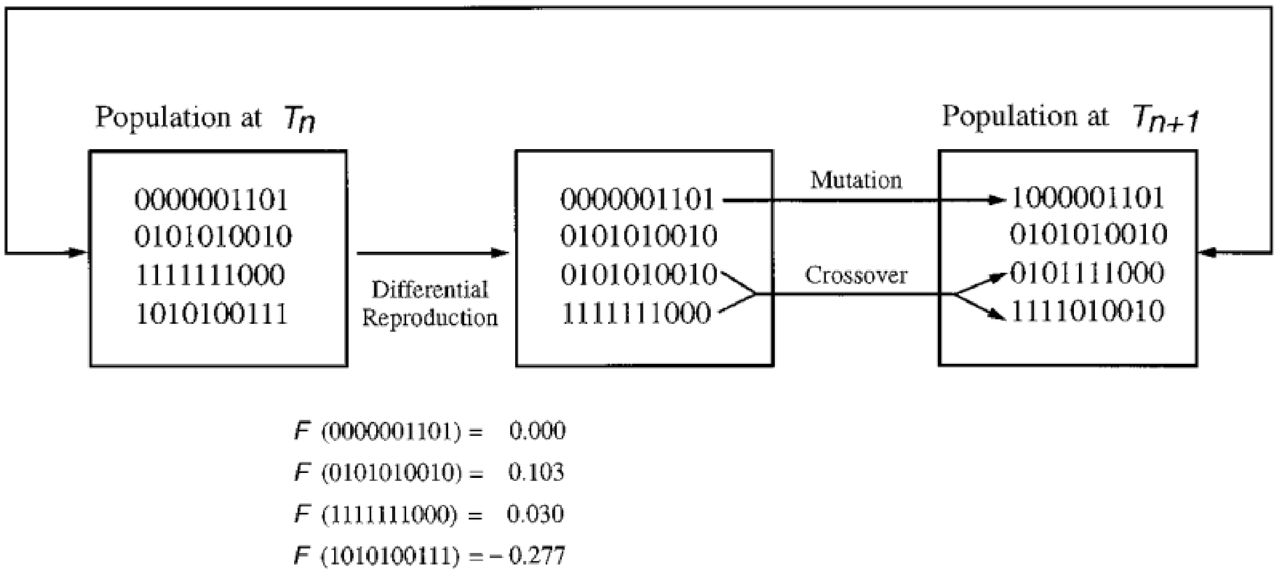


Figure 2.11: Genetic algorithm [For96]

It shows the evolution of a population with binary encoding with some example of mutation, crossover operation and fitness evaluation. We can now enter in details into these operations.

Selection

The selection is a process which is used at the beginning of every new generation, it chooses the best individuals to keep and eliminate the individuals who are too weak. The probability to pick an individual (p_s) is based on his fitness. The higher is the fitness, higher is the probability to be picked. That permit the solution to move towards a stronger population and allow us to delete individuals weaker after a mutation or a crossover.

Mutation

The mutation is a process which consists of flipping each bit individually for each individual with a certain probability p_m. This probability is often small to avoid that a mutation change a individual too much. This action permits to remark if a special change can change the results strongly and keep it if the change is positive.

Crossover

The crossover is a process which consists of mixing 2 individuals. A portion of the binary sequence is exchanged between two individuals. This exchange permits to keep a certain portion of the sequence if this portion is increasing the result. There exist a lot of different options for the crossover considering that we can swap bits in every ways possible and not only contiguous sequence. In the purpose to avoid too much change, We are also taking a small probability (p_c) of doing this operation.

2.3.2 Cartesian Genetic Programming

CGP is an evolution of simple genetic algorithm, it uses directed acyclic graphs to represent the program and the individuals [Mil20]. A directed acyclic graph is a directed graph with no directed cycles. A directed graph is graph with edges which have a direction and the no directed cycles means that we can't cycle into the graph. It is called "Cartesian" considering that these graphs are represented using a two-dimensional grid of computational nodes. Each node in the directed graph represents a particular function and is encoded by a number of integer genes. The nodes are divided into a function gene which is an ID of the computational function of the node in a user-defined look-up table of functions and connection genes which tell what is the input of the node. These genes represent addresses in a data structure like an list. Nodes take their inputs from either the output of a node (in acyclic CGP, from a previous column) or from a program input. To have a better understanding of the interpretation of the genotype, we can look at the figure 2.12 with simple math operation as function gene (0=Plus, 1=minus, 2=multiply, 3=divide, 4=or, 5=xor).

We can see how the genotype is directly related into the phenotype which is the graph.

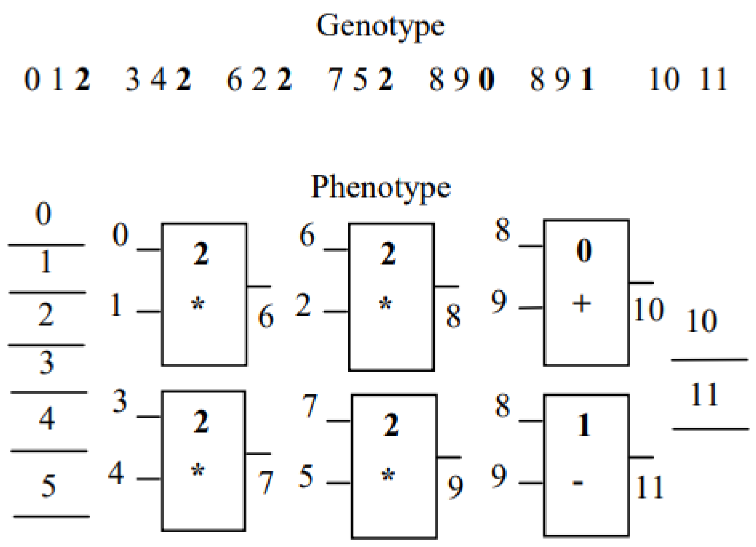


Figure 2.12: Link between genotype and phenotype from [Mil+99]

The genotype is split into group of one function gene and connected genes. In this case, the function gene is the last element of a group of three as most. So, the first triplet say that we need to use the function gene 2 (multiply) with as inputs the gene number 0 and 1. The genes from 0 to 5 are program inputs, the gene higher of 5 are the output of the previous nodes (triplet). So, after the first triplet, now we have the output of this triplet as gene number 6. The next gene is exactly the same principle the genes 3 and 4 are the input into the functional node to multiply and get a new gene number 7 as output. After that, the third triplet use as input the output of the first triplet considering that it is number 6 which is higher than the first input.

In our case, we will focus on the CGP of Suganuma [SSN17], we we will describe the representation of individuals, the different types of nodes, the genetic operators used, the algorithm and the evaluation of the fitness.

Representation of individuals

The individuals are a list of different nodes in a CNN represented in a directed acyclic graph and they are stocked in a genotype. We can now enter into more details at how we are representing them in practice.

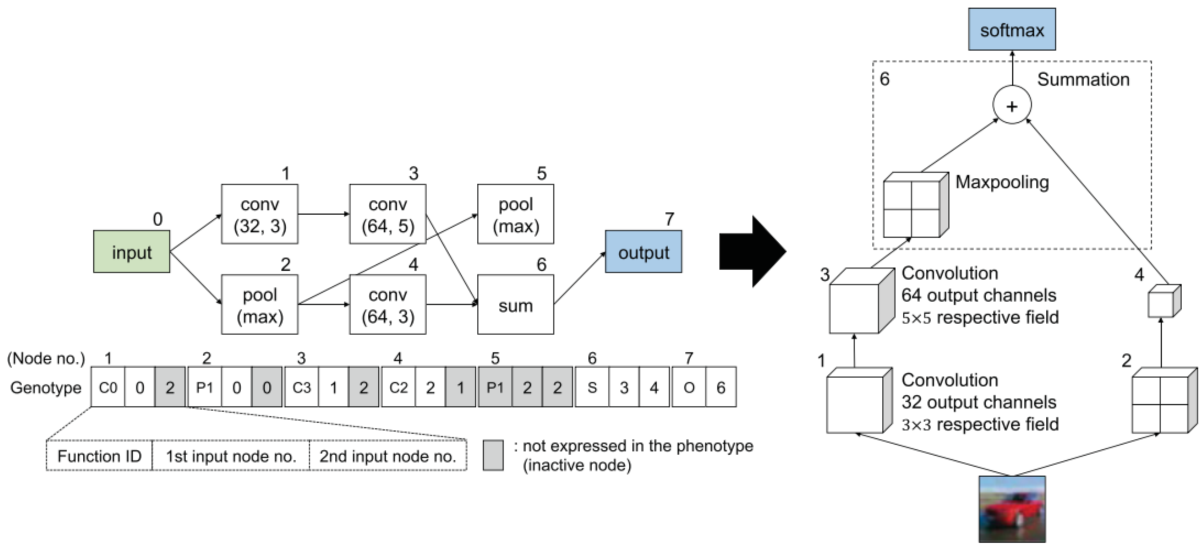


Figure 2.13: Representation of an individual from [Sug+20]

By looking at the figure 2.13, we can have a better understanding how the directed acyclic graph is directly related to the NNs and how it is link together. First of all, The genotype is an list which represent the graph. Every triplet composing the genotype is composed of an ID representing a type of the node in the CNN and followed by two integers representing the input nodes. At first, the only node in input is the number 0, After every triplet, we now have a new input possible which is the output of the node represented by the previous triplets. Every triplet can represent an active node or an inactive node. Indeed, some type of nodes in the CNN need two input and some need only one input. In this case, the first is used and the second become an inactive gene. Furthermore, If the a triplet is not used by another triplet, the node represented by the triplet in the graph become inactive. An inactive node won't be a part of the CNN. On the others hand, an active node is a part of the CNN. The genotype in CGP is a fixed-length representation, the number of nodes in the NN varies because of the inactive nodes, which is a desirable feature because the number of layers can be determined by the evolutionary algorithm. By representing the graph, we can see the active and inactive nodes. To directly transform the graph into our CNN, We can follow the unique path from the output node up to the input node.

Different types of nodes

The different types of nodes we are using are ConvBlock, ResBlock, Max pooling, Average pooling, Summation, Concatenation and an output node at the last position in every individual:

- The ConvBlock correspond to a standard convolution processing with a stride of 1 followed by a batch normalization and a ReLU. Moreover, we pad the outside of the input feature maps with zero values before the convolution operation. For this experiment, We prepare different ConvBlocks with a kernel size of 1, 3 and 5 and a number of output channels equal to 32, 64 and 128.

- The ResBlock correspond to a ConvBlock, a convolution operation, a batch normalization, a tensor summation and a ReLU. The ResBlock performs an identity mapping by shortcut connections. The row and column sizes of the input are preserved in the same way as ConvBlock after convolution. The output feature maps of the ResBlock are calculated by the ReLU activation and the summation with the input feature maps and the processed feature maps. This block is inspired from the ResNet architecture, we can have a better understanding of the layer in the figure 2.14. Like in the ConvBlocks, We prepare different ResBlocks with the same parameters.

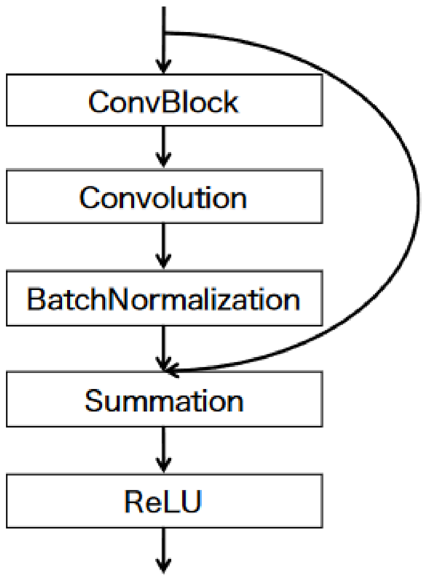


Figure 2.14: ResBlock architecture [SSN17]

- The max and average pooling nodes perform a max and an average pooling operation, respectively, on the feature maps. For this operation, we use a window of size 2×2 and a stride of size 2.

- The summation node performs an element-wise addition of two feature maps and The concatenation node concatenates two feature maps, channel by channel. The concatenation and the summation nodes have in common that if the input feature maps to be added have different numbers of rows or columns, we down-sample (reduce the size) the larger feature maps by max pooling so that they become the same sizes of the inputs. That allows us to to perform these operations in every case.

- The output node represents a simple softmax function with the number of classes. It fully connects every output of the last node to the nodes representing the class of predictions.

Genetic operator

The genetic operators used in this work are a forced mutation, a neutral mutation and a simple selection:

- The forced mutation is a series of mutation operation until at least one active node change. This operation is needed considering that the standard CGP mutation has the possibility of affecting only the inactive node. In that case, the CNN architecture does not change with this mutation and we don't need to reevaluate the fitness again. It is very important considering that The fitness evaluation of the CNN architectures is really expensive.
- The neutral mutation is a mutation which works only on inactive parental nodes to contribute potentially new nodes for the next generation. This operation is important considering that it maintain a neutral drift which is effective for the CGP evolution. It also allows us to avoid local optima considering that we modify a parent by the neutral mutation if the fitness of the offsprings doesn't improve.
- The selection operator is simply take the CNN with the best results for the fitness function used. The fitness function is the value of accuracy of the CNN

Considering that we only use a simple selection operation, the type and connections of each node randomly change to valid values according only to the mutation rate that we can change in the experiment.

Algorithm

In our case, We use the modified $(1 + \lambda)$ evolutionary strategy with λ = the number of offsprings ($\lambda = 2$ in our experiments). Our algorithm can be separated in different steps:

1. Generate an initial individual at random as Parent P, train the CNN represented by P and assign the validation accuracy as the fitness.
2. Generate a set of λ Children C by applying the forced mutation to P.
3. Train the λ CNNs represented by Children C and assign their validation accuracies as their fitness.
4. Apply the neutral mutation to the parent P.

5. Select the best individual from the set of P and C which will become the new P.

6. Return to step 2 until a stopping criterion is satisfied (For example, number of generations).

This algorithm can be modified in different way. In our case, we want to look at a change in the calculation of the fitness function to avoid the need to training every individuals produced.

2.4 States of the art

First of all, there are already a lot of research to find the best parameters of CNNs. Over the years, variants of CNN architectures have been developed [Li+21], leading to amazing advances in the field of deep learning. We can see this evolution in competitions such as the ILSVRC ImageNet challenge [Rus+15]. In this competition, we can observe a real decrease of the error rate in the recent years. We can look a bit the evolution of the number of participants and of the results on the dataset used in this challenge in the figure 2.15 and see that the evolution on the error is really decreasing over the years.

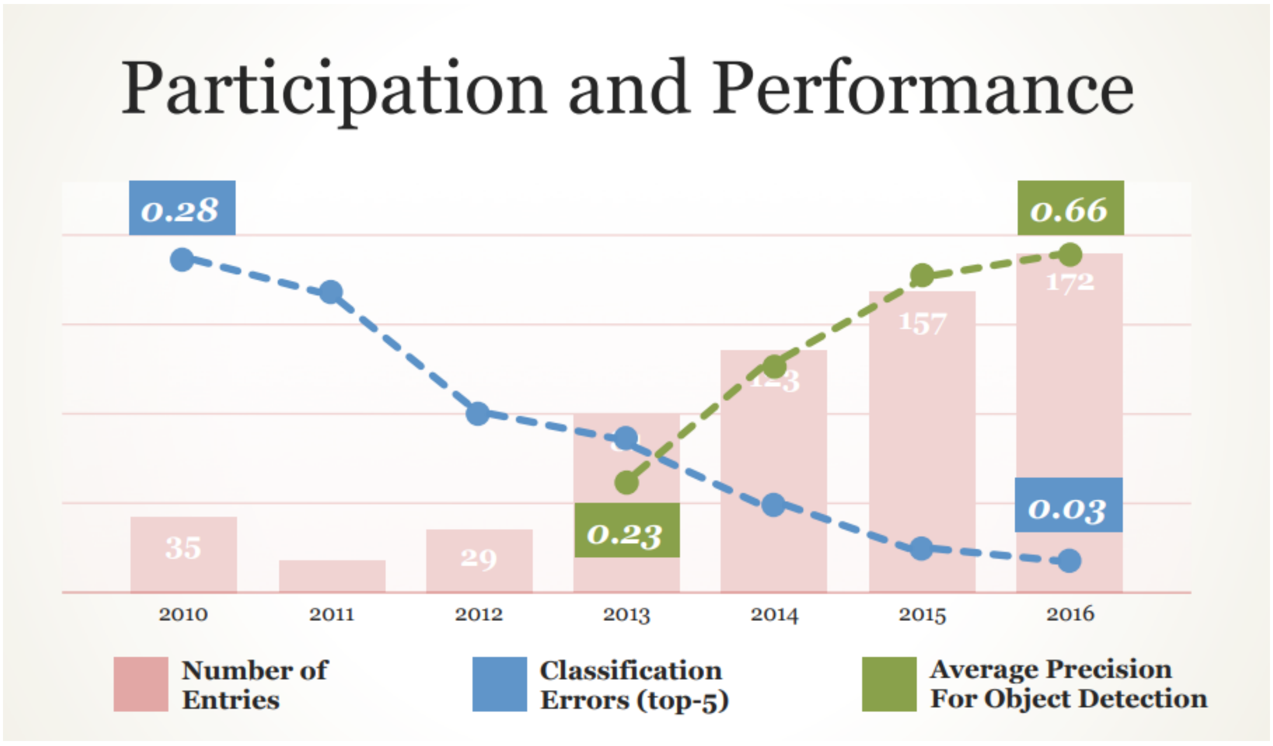


Figure 2.15: ILSVRC Results from [FD17]

2.4.1 A brief history of CNN

One of the biggest pioneers for the CNN is Yann André Le Cun for the creation of the LeNet CNN which is the first CNN created in history. For example, In 1998, he published this famous paper [LeC+98]. In 2012, Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton won the ImageNet Large Scale Visual Recognition Challenge with a test accuracy of 84.6% [KSH17]. Krizhevsky used GPUs to train the AlexNet, which enabled faster training of CNNs models and that started the new interest to study GPU to train CNN. The AlexNet consists of 5 convolutional layers and 3 fully connected layers. They won within a large margin on the runner-up and, because of that, a lot of studies on CNN started to emerge in this period. We won't enter into full details of every architectures of CNN found, but we can cite some like VGG[SZ15], GoogleNet [Sze+14], ResNet [He+15] and DenseNet [Hua+18]. There are also some architectures designed specially for mobile devices with very limited computing power like ShuffleNet [Zha+18] and ENet [Pas+16]. Obviously, there are a lot more in the history of the deep learning and of the CNN [Sch15]. We can look at the table 2.1 to see the results obtained with these architectures.

CNN's Name	top-5 test error rate (%)	Trainable Parameters (($\times 10^6$))
AlexNet	15.3	62
VGG	7.3	138
GoogleNet	6.67	6.4
ResNet	5.71	60
DenseNet	5.29	35
ShuffleNet	7.8	_

Table 2.1: Results of well known CNN in ILSVRC

2.4.2 A brief overview of NAS

NAS is quite a large field considering that the optimization process can be done in a lot of different way [WRP19]. Firstly, We could be interested in Reinforcement Learning (RL), evolutionary algorithms and some optimization techniques. We will only give some example of work considering the size of the states of the art of these method which is quite large.

Reinforcement Learning

They are quite a lot of work with RL for NAS. We will present some of them. First of all, in 2016, Baker et al introduced a a meta-modeling algorithm based on RL called MetaQNN to automatically generate high-performing CNN architectures for a given learning task [Bak+16]. The learning agent is trained to sequentially choose CNN layers using Q-learning with an ϵ-greedy exploration strategy and experience replay. Another interesting works are the works of Zoph et al from Google. In 2016, they proposed a NAS method with RL and recurrent network called NASNet [ZL16]. After that, in 2018, he worked with a larger team to push is idea further in this work [Zop+18]. In 2018, Pham also worked with him on a more efficient method called ENAS which is really interesting [Pha+18], this model main aspect is to force all child models to share weights to eschew training each child model from scratch to convergence.

In 2018, Zhong et al proposed a block-wise network generation method using Q-learning with ϵ-greedy exploration, called BlockQNN [Zho+18]. BlockQNN represents each convolutional block by a directed acyclic graph (DAG) where every node corresponds to a layer/operation and each directed edge expresses the data flow. The optimal block obtained by the learning agent represent the whole CNN.

Another interesting work is the work from Cai et al. In 2018, they proposed EAS, a new framework toward economical and efficient architecture search, where the meta-controller is implemented as a RL agent [Cai+18]. It learns to take actions for network transformation to explore the architecture space. By starting from an existing network and reusing its weights via the class of function-preserving transformation operations, EAS is able to utilize knowledge stored in previously trained networks and take advantage of the existing successful architectures in the target task to explore the architecture space efficiently.

Evolutionary algorithm

Evolutionary algorithm is a large field considering that they are inspired from the nature. We will cite some interesting algorithm in our case. One of the algorithm that we could be interested is particle swarm optimization [FY19]. It is based on the swarm behavior like birds who are looking after food in a flock. This algorithm was introduced in 1995 and has evolved a lot during these years to be able to used now in a lot of different case of optimization [PKB07]. One interesting work with this algorithm is the work from Jing Jiang and his team [Jia+20]. In 2020, they proposed an approach using a multiobjective particle swarm optimization based on decomposition. the computation cost of NAS with their approach has been reduced to an acceptable level, which makes the approach quite interesting.

A main type of evolutionary algorithm is GA. There is quite a large use of GA on CNN considering that it is quite a simple algorithm to look into a large search space. We will look after what was already done with GAs on CNNs. For example, in 2016, Earnest Paul Ijjina and Krishna Mohan Chalavadi have tried to optimize the initial weights of CNN by using GA [IC16]. In 2017, Lingxi Xie and Alan Yuille have tried to use GAs by using a binary string representation to optimize a network structure in a constrained case [XY17].In 2019, Another interesting work is proposed by Nurshazlyn Mohd Aszemi and P.D.D Dominic which is more focused on hyper-parameters and search algorithms to optimize these hyper-parameters [AS19]. Also in 2019, Mattioli et al have made a comparison between different search algorithms like random search, grid search and GAs [Mat+19]. They have found that the GAs have the best accuracy among these search algorithms.

There exist also some GA with quite a different representation of the data not raw data, but more structured data like a tree or a graph. This little evolution of GAs is called genetic programming. Furthermore, Miller proposed the CGP in which programs are represented by linear integer chromosomes in the form of connections and functionalities of a rectangular array of primitive functions[Mil+99]. In 2017, Suganuma et al have tried to use this CGP which encodes the CNN architecture by using directed acyclic graphs with a two-dimensional grid defined on computational nodes to optimize the architecture of CNN. We can look at two papers of them, one which define classically the experiment [SSN17] and the other one with more explanations [Sug+20] and different improvements to speed up the algorithm.

Optimization techniques

There also some technique less used but still interesting like the Hill climbing Optimization [EMH17] and the simulated annealing [Ayu+16]. One interesting work for hill climbing is the work from Elsken et al. In 2017, they proposed a Neural Architecture Search by hill climbing called NASH, a simple iterative approach that, at each step, applies a set of alternative network morphisms to the current network, trains the resulting child networks. It is in idea really close to a a very simple evolutionary algorithm with a population size of the number of neighbors, no cross-over, network morphisms as mutations, and a selection mechanism that only considers the best-performing population member as the parent for the next generation. this algorithm starts with a solution and makes small improvements to it until it has an acceptable result.

Another work is the microcanonical annealing algorithm from Ayumi et al [Ayu+16]. In 2016, they proposed a work on a a variant of simulated annealing called micro-canonical annealing algorithm. The conventional simulated annealing algorithm aims to bring a system to equilibrium by decreasing temperatures to find the best solution. However, the micro-canonical annealing algorithm focus not on the temperatures but on its internal energy. This technique is based on the Creutz algorithm, known as “demon” algorithm or microcanonical Monte Carlo simulation. Without entering into details, the algorithm tolerates attainment of the equilibrium of thermodynamic in an isolated system, where in this condition, total energy of the system is constant. By doing so, they can produce better solution.

2.4.3 Predicting a CNN’s accuracy

There exists a lot of work on the use of different heuristics to estimate the accuracy of a CNN. In 2019, Hahn et al have tried to perform a heuristic to estimate of a candidate architecture of a CNN by dividing the training into two parts, one with the feature selection and one with the fully connected classification. The feature selection part is trained for only two epochs, whereas the fully connected classification can be trained for more epochs considering that it is less costly to train that the first part. That method is used to decrease the computational load. They combine this proposed heuristic with a Bayesian Optimization to find the best accuracy for these CNNs [Hah+19].

Another interesting work is the work of Boyang Deng et al, in 2017, with the development of a new method to test a NN without training it called Peephole [DYL17]. They first encodes each layer into a vector. Then, they apply a RNN to integrate the information of individual layers following the network topology into a structural feature. This structural feature with the epoch index will be finally given to a Multi-Layer Perceptron (MLP) to obtain the prediction of accuracy of the NN given at input at a certain time. They developed also a Block-based Generation, a simple yet effective strategy to generate a diverse set of reasonable network architectures.

Another interesting work from Baker et al is a work on the learning curve to predict the accuracy of a CNN [Bak+17]. In 2017, They introduced a method to parameterize learning curve trajectories with simple features derived from model architectures, training hyperparameters, and early time-series measurements from the learning curve. They use these features to train a set of frequentist regression models that predicts the final validation accuracy of partially trained neural network configurations using a small training set of fully trained curves from both image classification and language modeling domains.

Now we will talk about the method that we will use in this experiment: E2EPP.

2.4.4 E2EPP

Yanan Sun et al have also worked on a new way to predict the accuracy of CNN without training it. In 2020, They have worked on an end-to-end performance predictor (in short named E2EPP) based on random forest [Sun+20a], considering that the random forest construction and prediction is computationally cheaper than the training of every CNN. So, They are using AE-CNN to randomly generate a set of valid CNN architectures to feed to the random forest which will be trained one time to predict every new CNN after. This random forest is not really time-consuming to create. However, collect the data could take a long time, but when the data are collected, it is quite fast to predict the accuracy of new CNN and it obtains pretty good results.

This predictor uses multiple decision trees (random forest) to predict the accuracy of a CNN. A decision tree is a very practical model to predict from data. This model uses a tree and by looking at some part of the input, we take decisions to go in a different direction at every node. So, using multiple decision trees allow us to minimize the error by setting different condition values for each nodes. This algorithm can be separated in 3 operations: the collection and the encoding of the data, the creation and the training of the random forest and the prediction of an input CNN. We can observe how this predictor is used in the figure 2.16. We will now enter into more details into the 3 main operations of this predictor.

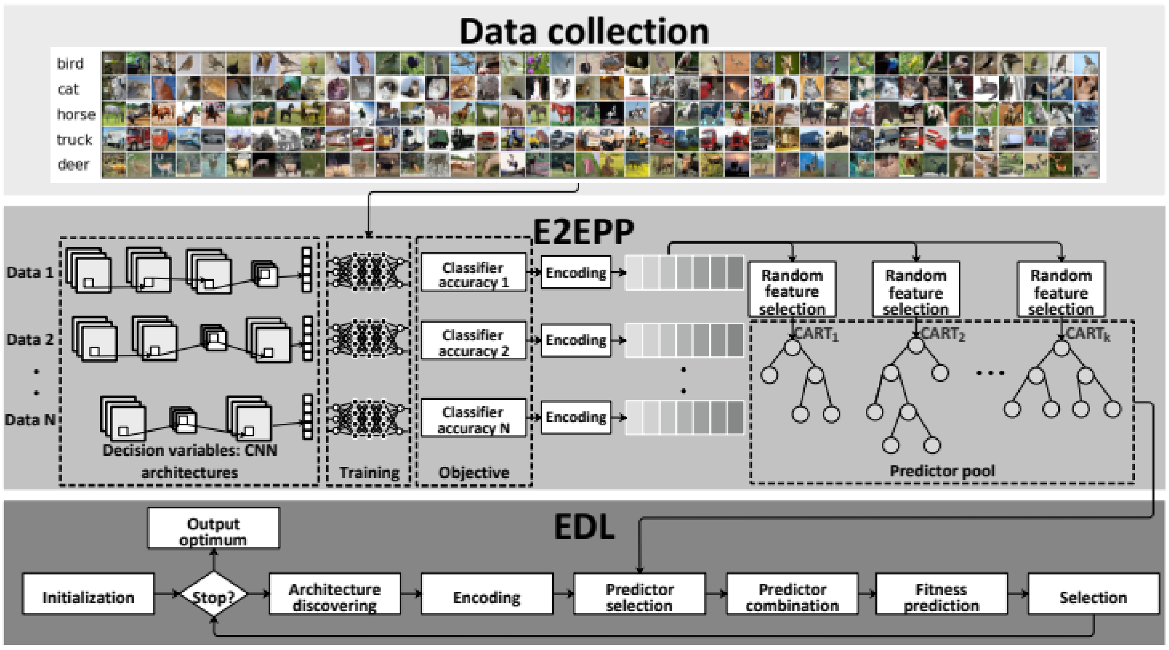


Figure 2.16: Utilization of E2EPP from [Sun+20a]

Collection and the encoding of the data

The first one is the collection and the encoding of the data. Firstly, To collect the data needed to train the decisions tree, they use one of their previous work AE-CNN [Sun+20b] to randomly generate a set of valid CNN architectures. A valid CNN architecture means that this architecture can be trained with a predefined training routine without any exceptions such as the out-of-memory errors causing a zero classification accuracy. The CNN are only composed of DenseNet Blocks (DBs), ResNet Blocks (RBs) and Pooling Blocks (PBs). Secondly, To encode their data, they have chosen to have a maximum number of nodes for the DB (N_b) and RB and another maximal number nodes of PB (N_p).

For the first maximal number, each RB or DB is encoded into a triplet as [type, out, amount], where the block type for RBs is set to 1, and that for DBs are set to 12, 20 and 40 when k is equal to 12, 20 and 40, respectively. For the second maximal number, each pooling layer is encoded into a pair as [pooling type, layer position], the maximal and mean pooling types are presented by 1 and 0, respectively. If a CNN architecture doesn't take all the variables, the rest of variables are set to zeros. Therefore, the performance predictor by using random forest is based on the input data with $3N_b + 2N_p$ discrete decision variables considering the triplet and the doublet, and the output is a continuous value within the range of [0, 1].

We will adapt this part to be able to use it with CGP.

Creation and training of the random forest

The second one is the creation and the training of the random forest. Each Decision tree is trained on the whole training set composed of the CNNs previously trained. Even if we train all decision tree with all the CNN, we are doing a random feature selection that assure different decision tree. A feature represent only one integer which represent the CNN. So, every Decision tree doesn't take into account all features of the CNN. Indeed, for each feature, we will take it with a probability of 0.5 in order to maximize the diversity of the predictor pool. Moreover, The training set is shuffled for every decision tree to avoid wrong association because of the order. By doing theses operations, we obtain different decision trees that will produce different results for the same input by taking only their selected features as inputs to predict.

Prediction of an input CNN

The Last one is the prediction of an input CNN. This step is simple considering that we take all regression trees and we predict the accuracy with the encoding of the CNN to be evaluated and their selected features. So, the input of the decision trees is only the value of the index of the selected features in the input CNN that is different for each decision tree. Then, we take the mean of these predictions as the result for the accuracy of the CNN.

3 Methods

3.1 Materials

In this thesis, we want to use the CGP of Suganuma et al to study different fitness functions. We will implement it by using Python as programming language and, more precisely, the PyTorch library [Pas+19] which is a well known library for AI and ML created by Facebook. We will start from the implementation of [SSN17] from his GitHub [Sug18]. Furthermore, We want to compare the results of this algorithm in terms of speed, accuracy and complexity of the architecture with different fitness functions that we will re-implement. We will be interested in a predictor like E2EPP [Sun+20a]. We will also implement a new method to combine the accuracy and the size of a CNN as the fitness function. For this experiment, we will use the datasets CIFAR-10, CIFAR-100 [Kri12] and SVHN [Net+11].

CIFAR-10 is a dataset composed of 10 different classes which are airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. CIFAR-100 is an evolution of CIFAR-10 with 100 classes. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs). For example, the superclasses are like different type of animals, different type of vehicles, daily object, etc. So, it it a very complex dataset with a lot a different classes. the SVHN (Street View House Numbers) is composed of the images of printed digits, from 0 to 9, cropped pictures of house number plates. The cropped images are centered in the digit of interest, but nearby digits and other distractions are kept in the image.

Considering the computational load of the neural network algorithms, we will use GPU to implement them to reduce the time of execution needed. In this experiment, the computer used is a MSI Model MS-7994. It has the NVIDIA GeForce GTX 1060 6 GB as GPU, 16 GB RAM and the Intel Core i7-7700 CPU 3.60 GHz as processor. The computer runs on Windows 10. So, we will only use 1 GPU for this experiment. The code, for this experiment, is available in our GitHub [Pie21].

3.2 CGP

We decided to use CGP considering its simplicity and quite direct logic inspired from the nature. This algorithm is really interesting considering that is can search after good solutions in a search space which can be too big to do an exhaustive search. Indeed, it is a heuristic algorithm. So, it searches the search space and try to find good solutions but not really the best solution considering that it is impossible to know the best solution. We can also precise the number of maximum nodes that we want. So, it is also really flexible considering that we use very simple operations. This algorithm is directly usable in practice and can be modify very easily to add a predictor or to just modify the fitness function in general.

3.3 Evaluation of Individual

Evaluate a CNN is the operation the most complex of this algorithm just for training a CNN for 50 epochs, it could take more than 1 hour. Considering that we have quite a lot of individuals to evaluate, it takes quite a long time to execute this algorithm. Indeed, if we use 50 generations, we have to train 101 CNNs. To speed up the algorithm, we are interested in heuristic algorithms to estimate the accuracy of these CNNs. In this case, we will call them predictors algorithms.

First of all, the predictors are heuristics algorithms used to reduce the computational load to evaluate the individuals by approximating their fitness. In our case, it is to avoid a complex training of our CNN. We want to know if they are worth to train and if they still have quite good solutions in accuracy. We are interested in the the predictor called E2EPP considering that it is one of the fastest predictors in the state of the art and that it achieve quite good results in accuracy.

Obviously, just try to reduce the time of the algorithm is not the only possibility, we are also interested by using different equations to take into argument different parameters of the CNN. In our case, we will be interested in taking into account the size of the CNN.

3.3.1 E2EPP

We decided to use this predictor considering that it was one of the fastest in the state of art and that it uses very simple models. Indeed, This predictor uses multiple decision trees (random forest) to predict the accuracy of a CNN. A decision tree is a very practical model to predict from data. This model uses a tree and by looking at some part of the input, we take decisions to go in a different direction at every node. So, by using multiple decision trees that allow us to minimize the error. This E2EPP predictor is very interesting considering that after the data are collected, the approximation of the CNN is very fast. However, collect the data is still a bit problematic. What we have decided to change to be able to use it with CGP is the collection and the encoding of the data.

Collection and the encoding of the data

The first one is the collection and the encoding of our data. Firstly, To collect the data needed to create the decisions tree, we will simply run our CGP algorithm without predictors and save in a file the architectures of the CNNs and their accuracy. Secondly, To encode our data, we have to choose a ways of encoding every node by using integer and give also every argument that differ for all nodes. We have decided to adapt the encoding the from Suganuma to a list composed of ID of the Node, input1, input2, output size and kernel size. The two last elements are there to permit to have an output size and kernel size which can be anything for the ResBlock and ConvBlock. If the node is not a ResBlock and not a ConvBlock, we set the output size and the kernel size to 0 . With these 5 integers, we can represent any nodes in our CNN. Considering that we want the input of the decision tree to always be of the same size, we will add a certain number of zero until we have exactly the number of maximum nodes of our algorithm. The IDs are matched to the name of node following the table 3.1:

Name	ID
input	1
full	2
Max Pooling	3
Average Pooling	4
Concatenation	5
Summation	6
ConvBlock	7
ResBlock	8

Table 3.1: Match Name and ID

The ID are different from the one used for the CGP considering that we wanted to not totally differentiate the ID of two ResBlock with different output size and kernel size considering that they are still the same type of node.

3.3.2 Accuracy and minimum size for the fitness function

First of all, we have though why would we limit the fitness function to just the accuracy. So, what could be interesting to optimize become an interesting question. Considering that we want to optimize the time of the algorithm, we have though to decrease the time needed to train an CNN. To do so, we need to decrease the number of trainable parameters. Considering the high values of trainable parameters and the difficulties to make a clean function which would impact but not too much the choice of the CNN, we decided to use a more simplistic parameter which is the number of nodes in the network. However, in further studies, it could be really interesting to use a function of the numbers of trainable parameters.

Considering that, we have developed a fitness function which take into account the number of nodes of the CNN in addition of the accuracy of the predictions. We will call this method Accuracy and Size (AAS). This allows us to try to minimize the size of the CNNs a bit and not take the larger CNN which risks to take more time to train and to predict. AAS is interesting considering that if we have two CNNs with the same accuracy, the one which is shorter will be picked and could lead to more interesting architectures. To implement this fitness function, we will use a simple equation separated in two parts, one for the accuracy and one for the size. This model follows the equation 3.1:

$$AAS = Acc + \frac{\alpha}{Size} \tag{3.1}$$

The equation 3.1 is composed of Acc which is the percentage of accuracy (between 0 and 100), α which is the parameter of the size and $Size$ which is the number of active nodes in the network. We divide α by the size considering that a lower size is more advantageous in the training and in the prediction. The parameter allows us to change the priority of the size over the accuracy of the network. The higher is alpha, the higher is the proportion of the fitness coming from the size.

3.4 Description of the experiments

we want to experiment the CGP algorithm with different fitness function. Considering that we are interested in 3 methods, we can divide our experiment in 4 different types. The first one is CGP only, the second one is CGP and AAS combined, the third one is CGP and E2EPP combined and the last is CGP, E2EPP and AAS combined. In these experiments, we will be interested to measure the accuracy, the execution time, the number of nodes and lastly the number of trainable parameters. All the algorithms will be launched with 50 generations and 2 children on the 3 datasets (CIFAR-10,CIFAR-100 and SVHN).

3.4.1 CGP

First of all, considering that without predictor, this algorithm is quite long, we will launch it on each dataset only one time to obtain some data to be used with E2EPP. Indeed, We need this step considering that, it allows us to collect the data that we will use as input for the E2EPP predictor. We will save the architectures and the accuracy of the NNs obtained after the training. This will also serve as base of comparison with the others algorithms used to reduce the execution time with a low number of generation. We will start by the classic training for 50 epochs.

3.4.2 CGP-AAS

After that, We will also launch CGP-AAS on each dataset only one time considering the execution time which will be quite high. Furthermore, we will experiment with different values for α that we want to look at before the training. We will launch CGP-AAS with $\alpha = 0.025$ and $\alpha = 0.05$ to compare it with the classic training.

3.4.3 CGP-E2EPP

Considering that a classic training take quite a long time, E2EPP which is is quite fast after the collection of data is very interesting. We have decided to use the CNN trained with the classic training as data for this experiment. We will experiment with this algorithm to predict the accuracy of a CNN. For this experiment, we will launch it 5 times on each dataset considering that it should be quite fast to finish the search and that the randomness of the prediction of the NN will probably impact a lot our prediction. However, we still need to train the best CNN obtained to have the results of the experiment. We will experiment with 50 generations and 250 numbers of generations. Furthermore, we will split the time and the accuracy of the experiment. They will become the evolution time and the training time. The evolution time is the time of the algorithm and the training time is the time to train the best CNN after that. The accuracy is now the predicted accuracy and the training accuracy.

We could ask ourselves what happens if we go for a lot more of generations. So, we will also test it with 1000 generations.

3.4.4 CGP-E2EPP-AAS

We will also try to combine our method AAS with the predictor E2EPP and look if it has an clear impact or it is more hollow. However, we could expect not the best results considering that the utilization of a predictor will make AAS less interesting considering that the comparison will be less precise. We won't be sure that CNN has a better accuracy over another one. We will run the same test as in CGP-E2EPP. However, we will use only $\alpha = 0.05$ in the experiment.

3.4.5 Analysis

After all the experiment, we will analyse a bit our results. First of all, We are interested in comparing them in term of accuracy, time of execution and numbers of parameters. To do so, we will use box-plot to visualize more clearly the difference between each method and realise some One-Way analysis Of variance(ANOVA) tests. These tests allows us to confirm if there is a statistically significant difference between the means of CGP-E2EEP and CGP-E2EPP-AAS in term of of accuracy, time of execution and numbers of parameters by looking at the $p-value$. If the $p-value$ is less than 0.05, we have sufficient evidence to say that the mean values across each group are not equal and that there is a statistically significant difference between the means of the group tested. In the other case, if it is above 0.05, there is not a statistically significant difference between the means of the group tested.

4 Experiments and Results

We can now start experiment the methods and look at the results obtained with them.

4.1 CGP

We can look at the result of this training for the accuracy, the execution time, the number of nodes and trainable parameters for the network on our 3 datasets in the table 4.1:

Dataset	CIFAR-10	CIFAR-100	SVHN
Accuracy (%)	90.56	51.74	93.81
Execution Time (h)	85.7444	45.3022	79.7463
# Nodes	14	12	18
# Trainable Parameters ($\times 10^6$)	1.841098	1.131108	0.863850

Table 4.1: Results obtained from classic training, using network accuracy as the fitness function

By looking at this table, we can clearly see that it takes quite a long time to have correct accuracy and not excellent accuracy. Furthermore, the numbers of nodes of these CNNs are quite large. So, We will be interested in the AAS method to try to decrease the size of the networks without losing too much accuracy. We can also look at the number of training parameters which is not really high. So, they are not too slow to train.

4.2 CGP-AAS

What is interesting in the AAS method is that we can change α to have a certain level of priority on the size of the network. However, we need to be careful that if α is too big, the accuracy will drop a lot. So, we will experiment with different values for α that we want to look at before the training.

4.2.1 Analysis of the variation of the parameter

More precisely, we want to look at the change in the fitness function with different values for α considering that the part of the fitness function represented by the size can be calculated by hand. We can see the difference of impact with different α between different common size of this algorithm in the table 4.2.

α \ size	8	9	10	11	12
0.025	0.3125	0.2777	0.25	0.2272	0.2083
0.05	0.625	0.55	0.5	0.4545	0.41
0.1	1.25	1.111	1	0.9090	0.8333

Table 4.2: Fitness change according to the numbers of nodes

We can see that for $\alpha = 0.025$ the change is close to 0.025 for a difference of 1 in size and as the size go higher the difference become less impacting between close size. As expected, we remark that this function increase linearly regarding to α. So, if we double alpha, the add to the fitness is also doubled.

4.2.2 Experimentation of CGP-AAS

Now, we want to look if AAS has a big impact on the experiment. We can look at the result for one run in the table 4.3.

Dataset	CIFAR-10	CIFAR-100	SVHN
Accuracy (%)	89.76	58.78	93.73
Execution Time (h)	55.7908	40.4533	73.8855
# Nodes	16	9	12
# Trainable Parameters ($\times 10^6$)	1.722730	0.878052	1.715754

Table 4.3: Results obtained from classic training with AAS and $\alpha = 0.025$

We can see that with $\alpha = 0.025$, there is some difference compared to the classic training. Indeed, the size is a bit lower in general. We also see that the execution time is also lower. However, the execution time on different runs changes quite a lot considering that each run is random and that it depends a lot of the CNNs produced during the exploration of the search space. Considering that the parent stays more small, we can expect the algorithm to be a bit faster. We can remark that the accuracy and the number of trainable parameters doesn't change a lot. Now, We want to test with $\alpha = 0.05$ and look if the changes are more noticeable or stay more or less equivalent. We can look at the result in the table 4.4.

Even with AAS, the classic training takes quite a long time. So, it is difficult to launch it a lot of times. We can see that in this case the experiment on CIFAR-10 took a very long time (more than 4 days). That could be explained by the size of the network to train and by the

Dataset	CIFAR-10	CIFAR-100	SVHN
Accuracy (%)	90.38	59.94	94.54
Execution Time (h)	99.8174	67.3597	88.2858
# Nodes	12	14	11
# Trainable Parameters ($\times 10^6$)	2.830826	4.648196	1.565162

Table 4.4: Results obtained from classic training with AAS and $\alpha = 0.05$

number of trainable parameters which is quite high compared to the previous experiment. Indeed, even if we prefer shorter CNNs, the production of the CNN by the mutation operation doesn't change compared to the classic training. So, we can't really conclude anything considering that we would have needed to launch a lot of runs to be able to look if there are some clear differences. However, considering that each run can take 4 days to finish, we won't have enough times to look at these experiments with this limited setup.

4.3 CGP-E2EPP

We can now start the E2EPP experiments. Firstly, we will look with 50 generations.

4.3.1 50 generations

we launched it 5 times for 50 generations on every datasets and look at the results in the tables 4.5, 4.6 and 4.7.

Nb run	1	2	3	4	5
Predicted Accuracy (%)	78.14	77.12	78.63	76.19	78.58
Training Accuracy (%)	89.66	82.72	79.30	86.18	73.24
Evolution Time (s)	25.432	29.9189	27.1865	26.0080	25.5435
Training Time (h)	0.64	0.84	0.21	0.25	0.22
# Nodes	11	12	6	10	7
# Trainable Parameters ($\times 10^6$)	0.841066	2.713418	0.219146	0.318282	0.040298

Table 4.5: Results of E2EPP predictor with 50 generations on CIFAR-10

Nb run	1	2	3	4	5
Predicted Accuracy (%)	48.12	47.98	47.18	48.08	47.71
Training Accuracy (%)	48.64	52.14	57.38	43.26	48.02
Evolution Time (s)	28.6701	29.9189	26.1691	25.8697	25.6681
Training Time (h)	0.50	0.33	0.41	0.62	0.32
# Nodes	7	7	7	12	7
# Trainable Parameters ($\times 10^6$)	0.835626	0.406986	0.377482	0.545994	0.314442

Table 4.6: Results of E2EPP predictor with 50 generations on CIFAR-100

Nb run	1	2	3	4	5
Predicted Accuracy (%)	75.23	76.63	74.20	75.71	76.81
Training Accuracy (%)	92.97	90.76	68.53	89.17	92.56
Evolution Time (s)	26.4978	26.2940	30.7151	25.7450	25.3525
Training Time (h)	0.43	0.69	0.32	0.30	1.01
# Nodes	9	9	6	7	16
# Trainable Parameters ($\times 10^6$)	0.246122	1.146954	0.047402	0.954058	1.145034

Table 4.7: Results of E2EPP predictor with 50 generations on SVHN

We can now observe a bit these results. Firstly, we can clearly see that the evolution time is close to 30 s and is insignificant compared to the the training time which is between 0.5 hours and 1 hours. The total time needed for this experiment is less than 1 hours which is more than 40 times better than the numbers of hours needed with a classic training. It is very interesting in time.

However, we still need to look at the accuracy obtained. We can see that the training accuracy is lower than with the classic training and that it fluctuate a lot between each run even if some runs are close to the results obtained with the classic training, some runs are really low in accuracy. On the other hand, the predicted accuracy is very close for each run. The reason for this observation could be that we don't have enough sample to train correctly our predictor or that the samples are too close in term of architecture. We can also observe that the number of trainable parameters is not very high with this predictor and with CGP in general.

Considering that the time needed for the evolution process is really low, we will also try it with 250 generations.

4.3.2 250 generations

We can now look at the same experiment with 250 generations. The results of this experiment can be found in the tables 4.8, 4.9 and 4.10.

Nb run	1	2	3	4	5
Predicted Accuracy (%)	77.40	78.98	79.00	78.68	78.45
Training Accuracy (%)	87.58	81.22	89.44	83.48	86.84
Evolution Time (s)	131.0145	139.2863	144.7816	145.7498	159.4445
Training Time (h)	0.56	0.40	0.24	0.30	0.43
# Nodes	10	7	7	7	7
# Trainable Parameters ($\times 10^6$)	0.923210	0.437834	0.572394	0.312618	0.570858

Table 4.8: Results of E2EPP predictor with 250 generations on CIFAR-10

As we can see on these results, even with increasing the numbers of generations, we don't really obtained better results in term of accuracy. This could be explained by the lack

Nb run	1	2	3	4	5
Predicted Accuracy (%)	47.42	47.82	47.86	49.34	47.88
Training Accuracy (%)	54.94	53.72	42.36	55.22	51.14
Evolution Time (s)	141.4068	139.6087	144.5409	141.1013	147.6354
Training Time (h)	0.24	0.28	0.55	0.62	0.52
# Nodes	9	7	7	12	7
# Trainable Parameters ($\times 10^6$)	0.581962	0.294314	0.814250	0.517002	1.563146

Table 4.9: Results of E2EPP predictor with 250 generations on CIFAR-100

Nb run	1	2	3	4	5
Predicted Accuracy (%)	75.04	77.77	75.21	77.83	75.70
Training Accuracy (%)	82.39	88.94	92.45	92.94	94.06
Evolution Time (s)	128.4203	141.2099	125.9952	137.3904	143.9185
Training Time (h)	0.33	0.78	0.43	1.1208	0.5722
# Nodes	6	9	8	13	9
# Trainable Parameters ($\times 10^6$)	0.918922	0.931594	0.459594	1.56497	0.981834

Table 4.10: Results of E2EPP predictor with 250 generations on SVHN

of different type of CNNs in the data used to train the predictor.

4.3.3 1000 generations

We can now look at the experiment with 1000 generations. we can observe the results in the figure 4.11, 4.12 and 4.13.

Nb run	1	2	3	4	5
Predicted Accuracy (%)	78.72	78.66	78.89	78.61	78.95
Training Accuracy (%)	85.70	82.18	82.26	76.46	86.16
Evolution Time (h)	0.15	0.15	0.15	0.16	0.15
Training Time (h)	0.30	0.22	0.23	0.28	0.23
# Nodes	6	7	6	6	6
# Trainable Parameters ($\times 10^6$)	0.407754	0.415722	0.243882	0.328458	0.290090

Table 4.11: Results of E2EPP predictor with 1000 generations on CIFAR-10

In term of accuracy, we can see that results are not higher in the CIFAR-10 case but it is higher in the two other cases. We can also see that the evolution time is indeed less negligible but still very low.

Nb run	1	2	3	4	5
Predicted Accuracy (%)	50.65	47.93	50.29	50.24	50.26
Training Accuracy (%)	54.58	46.34	50.12	56.22	52.12
Evolution Time (h)	0.15	0.17	0.16	0.15	0.15
Training Time (h)	0.27	0.25	0.27	0.58	0.47
# Nodes	11	7	9	13	9
# Trainable Parameters ($\times 10^6$)	0.239466	0.273610	0.344266	0.540746	0.650858

Table 4.12: Results of E2EPP predictor with 1000 generations on CIFAR-100

Nb run	1	2	3	4	5
Predicted Accuracy (%)	80.42	78.50	76.99	77.64	76.37
Training Accuracy (%)	93.27	79.41	92.86	88.20	93.49
Evolution Time (h)	0.15	0.14	0.17	0.15	0.15
Training Time (h)	0.97	0.56	0.61	0.56	0.62
# Nodes	14	9	9	10	15
# Trainable Parameters ($\times 10^6$)	0.896650	0.518314	1.440778	1.130602	0.691946

Table 4.13: Results of E2EPP predictor with 1000 generations on SVHN

4.4 CGP-E2EPP-AAS

We can now try to combine our method AAS with the predictor E2EPP and look if it has an clear impact and if they can be merge together correctly. We will also try with 50 and 250 generations.

4.4.1 50 generations

We can observe the results of this experiment with 50 generations in the table 4.14, 4.15 and 4.16.

Nb run	1	2	3	4	5
Predicted Accuracy (%)	78.71	78.58	77.81	79.58	79.57
Training Accuracy (%)	82.10	86.50	9.94	87.76	83.40
Evolution Time (s)	25.5663	25.7492	28.5668	26.6565	27.3558
Training Time (h)	0.47	0.46	1.01	0.22	0.23
# Nodes	6	7	11	7	6
# Trainable Parameters ($\times 10^6$)	0.762858	0.472330	1.686442	0.27697	0.600874

Table 4.14: Results of E2EPP predictor with 50 generations with AAS and $\alpha = 0.05$ on CIFAR-10

As we can expect, the numbers of nodes is not very high considering that AAS and E2EPP individually are a low numbers of nodes on average. However, the accuracy doesn't really increase. It could be explained by the fact that this method is not really practical considering that it use a predicted accuracy, we can't really be certain that a CNN is better over another

Nb run	1	2	3	4	5
Predicted Accuracy (%)	48.66	48.46	47.84	48.07	48.46
Training Accuracy (%)	49.62	50.18	40.88	50.40	19.74
Evolution Time (s)	29.6916	32.6490	29.2576	25.4748	26.4801
Training Time (h)	0.27	0.58	0.59	0.36	1.60
# Nodes	7	14	5	6	14
# Trainable Parameters ($\times 10^6$)	0.348362	0.864234	1.857770	0.250698	2.979114

Table 4.15: Results of E2EPP predictor with 50 generations with AAS and $\alpha = 0.05$ on CIFAR-100

Nb run	1	2	3	4	5
Predicted Accuracy (%)	74.24	76.15	75.74	73.29	75.62
Training Accuracy (%)	90.29	85.83	92.04	92.86	78.87
Evolution Time (s)	31.2864	29.9539	30.0622	31.2426	28.6733
Training Time (h)	0.44	0.60	0.32	0.38	0.33
# Nodes	6	8	6	9	6
# Trainable Parameters ($\times 10^6$)	0.541962	0.491082	0.313674	0.905130	0.210698

Table 4.16: Results of E2EPP predictor with 50 generations with AAS and $\alpha = 0.05$ on SVHN

one in accuracy, so take into account the numbers of nodes don't really prioritize the accuracy. So, even if it obtain correct results like E2EPP alone, it is not totally interesting to get them together considering that the prediction miss a bit of precision. However, we could look at it more with a better sample of NNs to train our predictor.

4.4.2 250 generations

We can now observe the results of this experiment with 250 generations in the table 4.17, 4.18 and 4.19.

Nb run	1	2	3	4	5
Predicted Accuracy (%)	79.68	79.43	79.39	78.84	79.25
Training Accuracy (%)	83.68	86.62	84.10	89.86	88.84
Evolution Time (s)	149.5616	145.6418	129.9221	133.0064	128.8927
Training Time (h)	0.49	0.22	0.22	0.43	0.31
# Nodes	6	7	7	12	7
# Trainable Parameters ($\times 10^6$)	0.857162	0.160010	0.244202	1.515594	0.390410

Table 4.17: Results of E2EPP predictor with 250 generations with AAS and $\alpha = 0.05$ on CIFAR-10

Nb run	1	2	3	4	5
Predicted Accuracy (%)	48.68	48.59	48.56	50.16	48.45
Training Accuracy (%)	51.46	47.28	48.02	50.92	51.22
Evolution Time (s)	149.0132	134.9744	140.7407	145.8461	157.4696
Training Time (h)	0.46	0.24	0.54	0.47	0.46
# Nodes	7	7	7	9	7
# Trainable Parameters ($\times 10^6$)	0.514762	0.236074	0.698122	0.651178	0.636714

Table 4.18: Results of E2EPP predictor with 250 generations with AAS and $\alpha = 0.05$ on CIFAR-100

Nb run	1	2	3	4	5
Predicted Accuracy (%)	79.00	75.80	75.78	78.42	77.28
Training Accuracy (%)	94.36	92.68	81.90	90.55	91.14
Evolution Time (s)	127.2747	136.9440	134.0478	136.5994	155.1930
Training Time (h)	0.88	0.63	0.32	0.73	0.64
# Nodes	14	12	6	9	9
# Trainable Parameters ($\times 10^6$)	1.336458	1.182282	0.303946	0.631050	0.524298

Table 4.19: Results of E2EPP predictor with 250 generations with AAS and $\alpha = 0.05$ on SVHN

4.4.3 1000 generations

Now, we can also observe the result with E2EPP and AAS combined together in the figure 4.20, 4.21 and 4.22.

Nb run	1	2	3	4	5
Predicted Accuracy (%)	79.47	79.65	79.04	79.50	79.66
Training Accuracy (%)	79.72	83.24	80.86	82.32	79.24
Evolution Time (h)	0.15	0.15	0.16	0.15	0.15
Training Time (h)	0.32	0.49	0.22	0.40	0.46
# Nodes	6	6	7	6	6
# Trainable Parameters ($\times 10^6$)	0.685002	0.952298	0.234442	0.545802	0. 632618

Table 4.20: Results of E2EPP predictor with 1000 generations with AAS and $\alpha = 0.05$ on CIFAR-10

Even with increasing the the number of generations, we can see that we obtained the same type of results that with E2EPP. So, we think that could be caused by the sample used to train the predictor on CIFAR-10. They could not be the most adapted for this predictor. We can now look a bit more to the results that we obtained and analyse them further.

4.5 Analysis

Now that we have all these results, we are interested in comparing them in accuracy, time of execution and numbers of nodes. To do so, we will use box-plot to visualize more

Nb run	1	2	3	4	5
Predicted Accuracy (%)	48.75	48.48	48.32	49.93	48.29
Training Accuracy (%)	49.06	51.24	48.22	52.84	50.56
Evolution Time (h)	0.14	0.14	0.14	0.15	0.14
Training Time (h)	0.20	0.26	0.28	0.58	0.34
# Nodes	7	7	7	13	7
# Trainable Parameters ($\times 10^6$)	0.254506	0.311242	0.511658	0.653674	0.360778

Table 4.21: Results of E2EPP predictor with 1000 generations with AAS and $\alpha = 0.05$ on CIFAR-100

Nb run	1	2	3	4	5
Predicted Accuracy (%)	77.29	79.32	78.49	77.36	76.69
Training Accuracy (%)	92.96	94.09	94.35	93.37	80.81
Evolution Time (h)	0.16	0.14	0.16	0.15	0.15
Training Time (h)	0.75	0.66	1.35	0.37	0.32
# Nodes	9	14	14	10	7
# Trainable Parameters ($\times 10^6$)	0.813322	1.051050	2.998698	1.262730	0.194346

Table 4.22: Results of E2EPP predictor with 1000 generations with AAS and $\alpha = 0.05$ on SVHN

clearly the difference between each method and use a One-Way ANOVA to look if there is a statistically significant difference between the corresponding means of these methods.

4.5.1 Accuracy

Box-plot

First of all, For the accuracy, we will plot a box-plot with the result for every E2EPP experiments, the best accuracy of the data used to train E2EPP and the best accuracy obtained overall with a classic training.

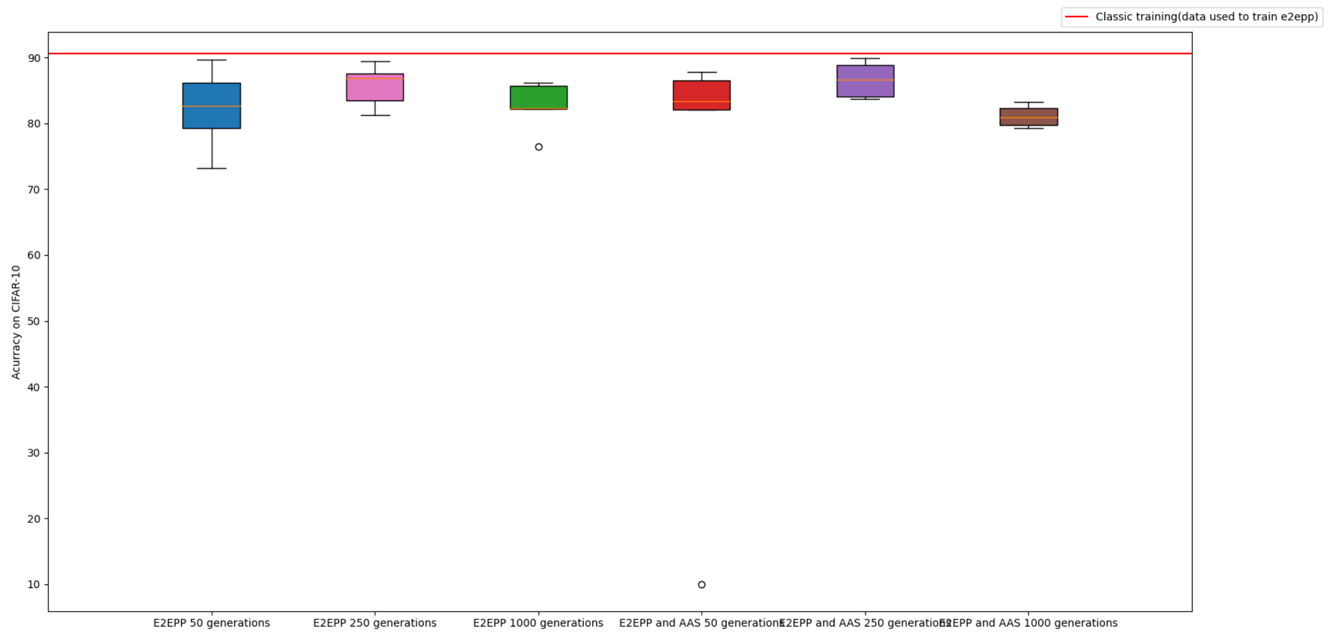


Figure 4.1: Box-plot of the accuracy on CIFAR-10

First of all, we can look at the figure 4.1 for the dataset CIFAR-10. In this plot, we have only one line considering that the best accuracy was obtained with the data used for training E2EPP. On the two other, we have the two different line. We can see that the best results is the classic training. However, the best run of E2EPP with 50 and 250 generations is still really close. We can clearly see that the results the most interesting are for 250 generations and that the accuracy drop when doing the experiment for 1000 generations. This is probably cause by a problem of sample used to train the predictor that cause a higher prediction than the exact value.

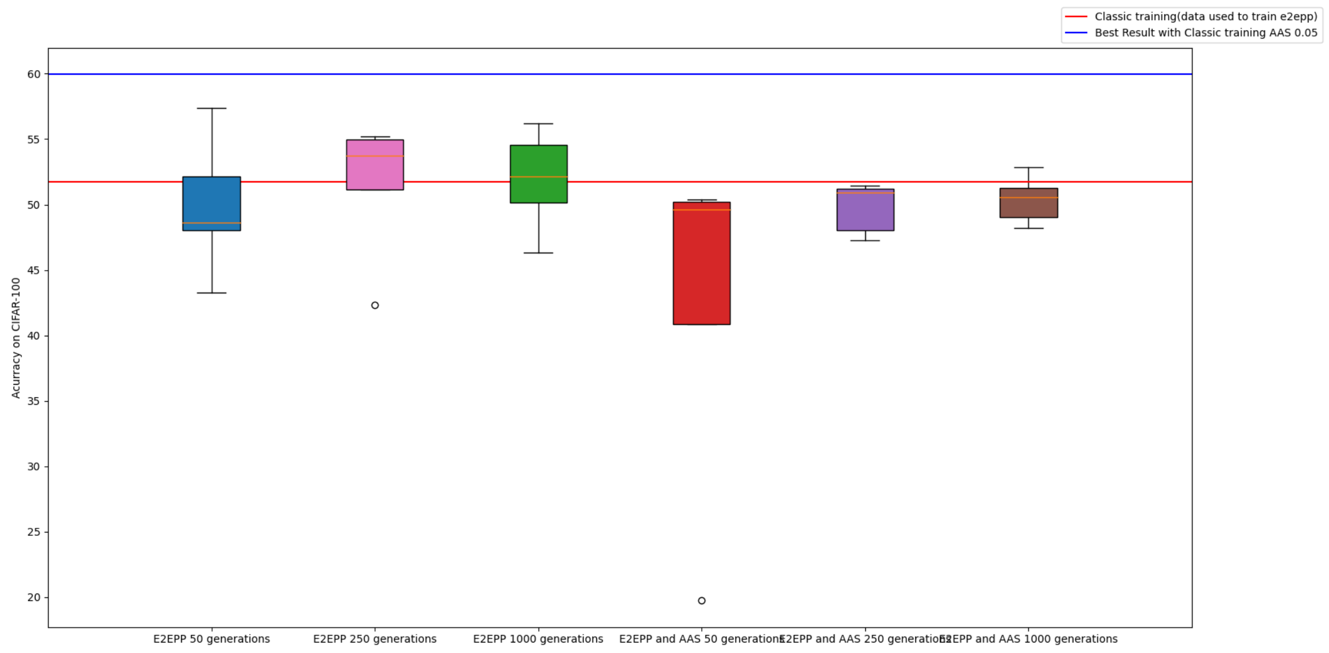


Figure 4.2: Box-plot of the accuracy on CIFAR-100

Then, we can look at the figure 4.2 for the dataset CIFAR-100. This plot is already more what we were expecting. Indeed, even if we are less close to the best accuracy obtained with a classic training and AAS with $\alpha = 0.05$. We can see that even if we use a predictor, we can exceed the accuracy of the data used to train the predictor. We can also observe quite a better results for the algorithms with 1000 generations. So, in this case, increase the number of generations produce quite an encouraging effect.

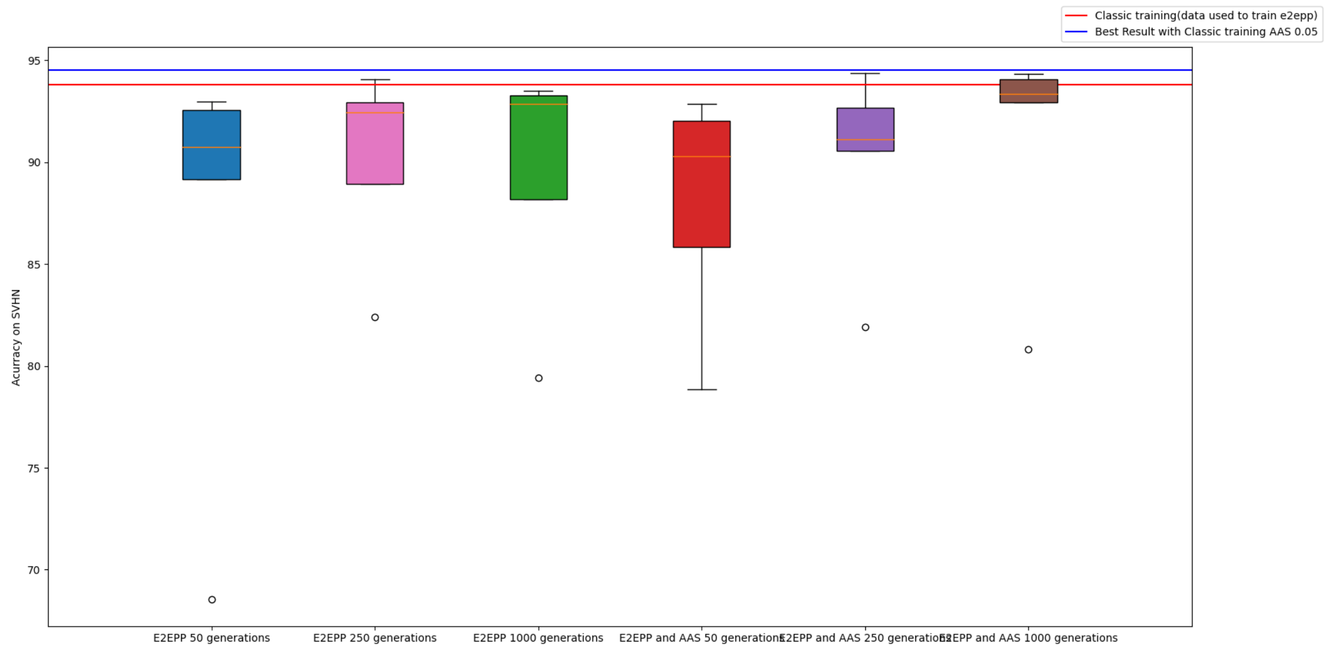


Figure 4.3: Box-plot of the accuracy on SVHN

Lastly, we can look at the figure 4.3 for the dataset SVHN. We can also see that the median of the accuracy increases when we increase the number of generations and that the predictor exceed the accuracy of the data used to train the predictor. Even, if there are still some runs with mediocre accuracy. The accuracy on this dataset is high and really close to the classic training.

Overall, on every dataset, The accuracy obtained is quite encouraging considering that it stays very close to the classical training of the CNNs. However, we can also see that the accuracy varying a lot in these different run. We can also observe that, on the CIFAR-10 dataset, the results were not as expected and that could be caused by the training samples used to train the predictor. So, we still need to be careful at how we collect these samples.

ANOVA

Now, we can realise an One-Way ANOVA test to check if the mean of the accuracy change really with the different methods. We can see the result in the figure 4.4.

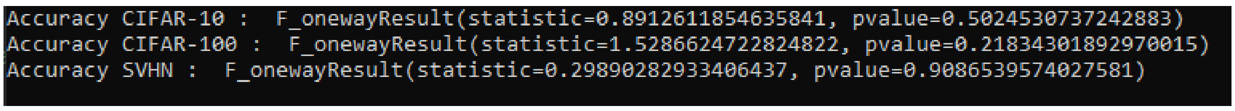

```
Accuracy CIFAR-10 :  F_onewayResult(statistic=0.8912611854635841, pvalue=0.5024530737242883)
Accuracy CIFAR-100 :  F_onewayResult(statistic=1.5286624722824822, pvalue=0.21834301892970015)
Accuracy SVHN :  F_onewayResult(statistic=0.29890282933406437, pvalue=0.9086539574027581)
```

Figure 4.4: One-Way ANOVA test on the accuracy from CGP-E2EPP and CGP-E2EPP-AAS

We can see that the p-value is too high. So there is not a statistically significant difference between the means of CGP-E2EPP and CGP-E2EPP-AAS with the different numbers of generations in term of accuracy.

4.5.2 Execution Time

We also need to look at the execution time considering that it is very important considering that a lot of approaches in the state of the art of NAS don't really care about the computational load of these algorithm that couldn't be launched on a private computer. We will start by comparing all the methods in execution time.

Box-plot

Considering that we expect a big difference in time, we will zoom on the plot after to be able to look if AAS make the algorithm faster.

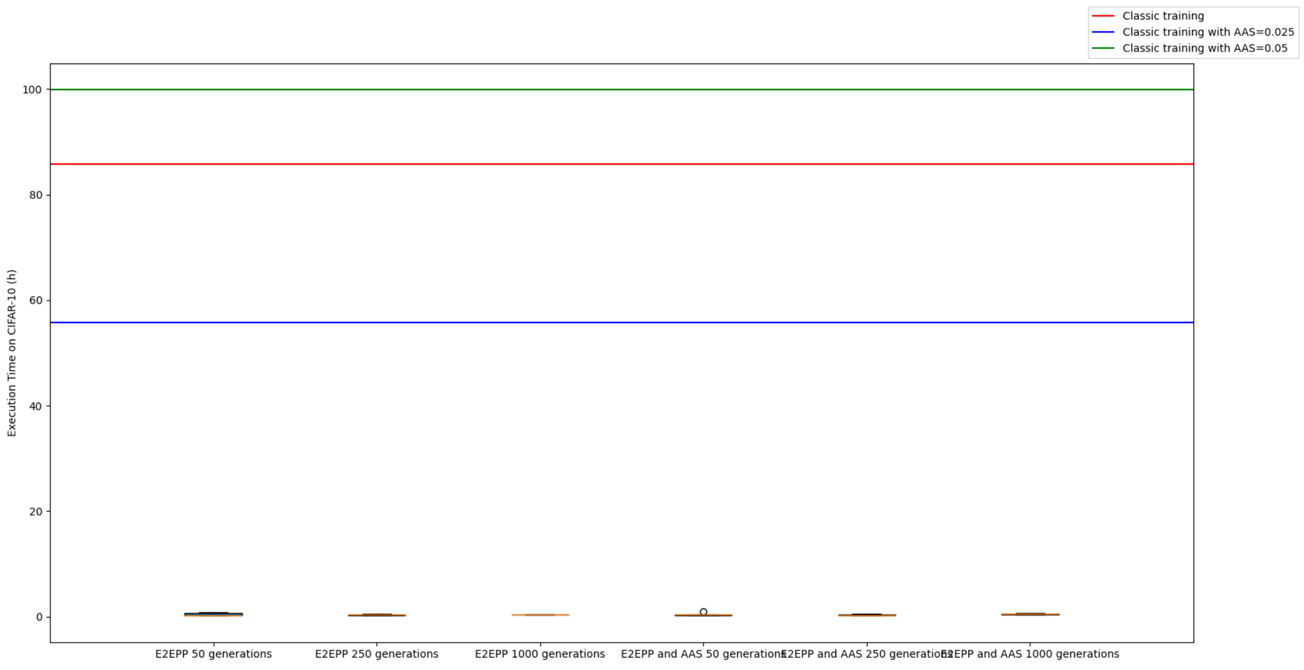


Figure 4.5: Box-plot of the execution Time on CIFAR-10

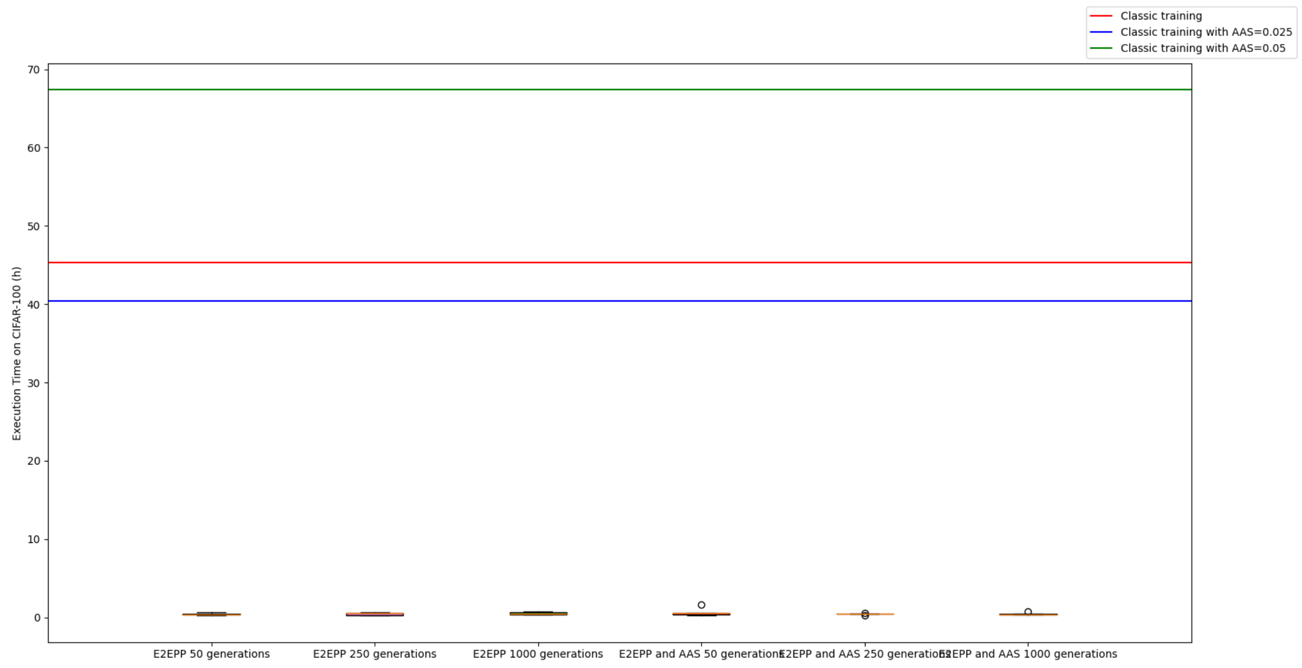


Figure 4.6: Box-plot of the execution Time on CIFAR-100

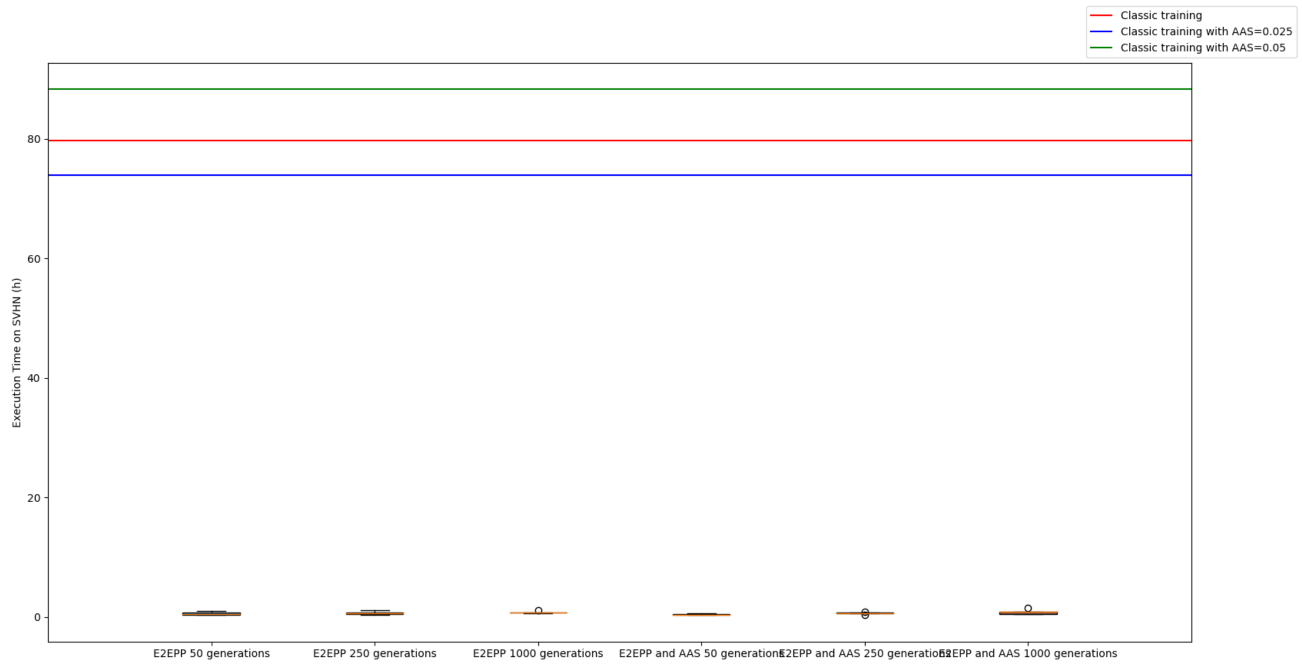


Figure 4.7: Box-plot of the execution Time on SVHN

As expected in figure 4.5, 4.6 and 4.7, we can clearly see that E2EPP is at least 40 times faster than a classic training. Indeed, it depends of the size of the CNN explored and trained during the algorithm. So, it can be really long. We can also see that AAS can be lower in time but it is not enough evidence that it really reduce the time of the algorithm. However, we can't really look precisely at the time needed for E2EPP. So, we will zoom into the plot.

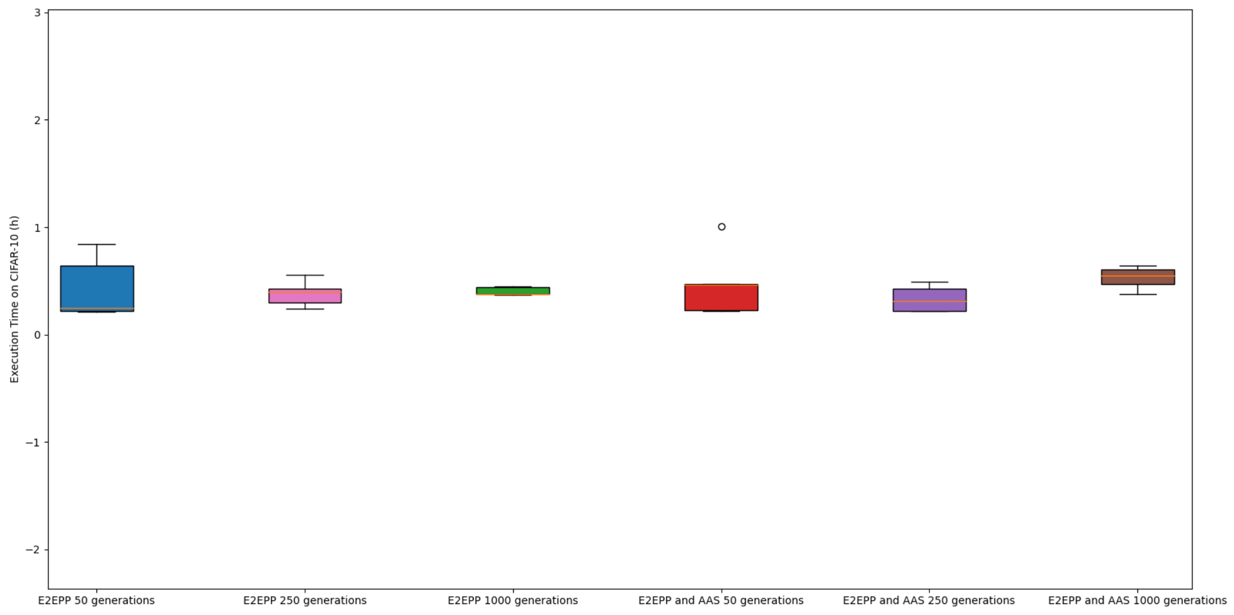


Figure 4.8: Box-plot of the execution Time on CIFAR-10 Zoom on E2EPP

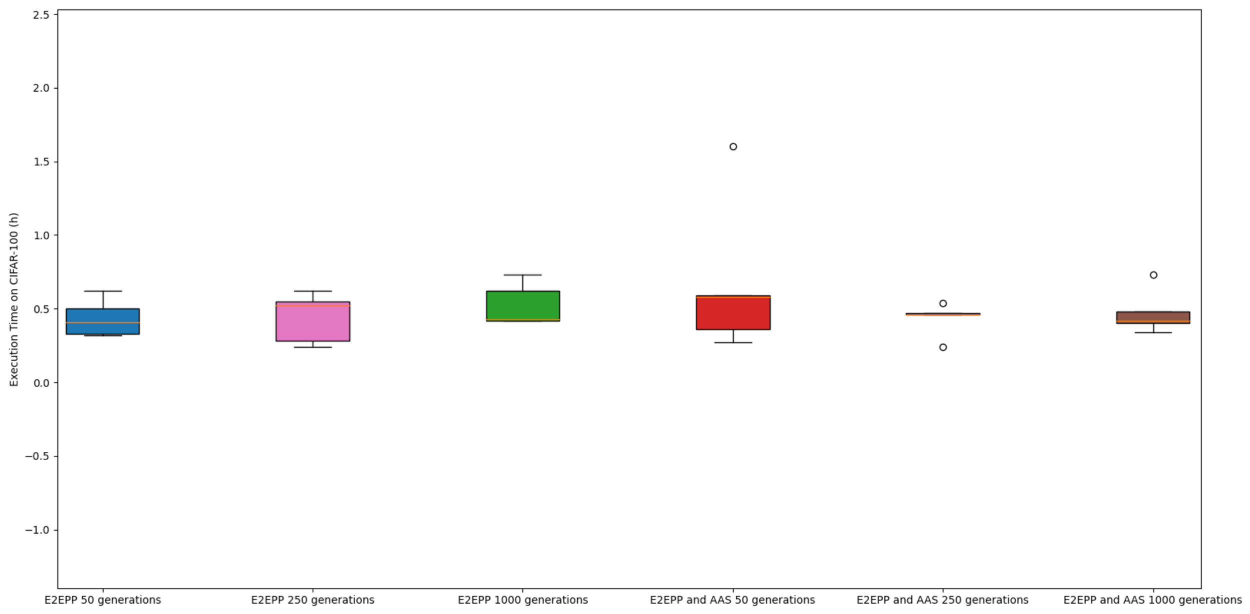


Figure 4.9: Box-plot of the execution Time on CIFAR-100 Zoom on E2EPP

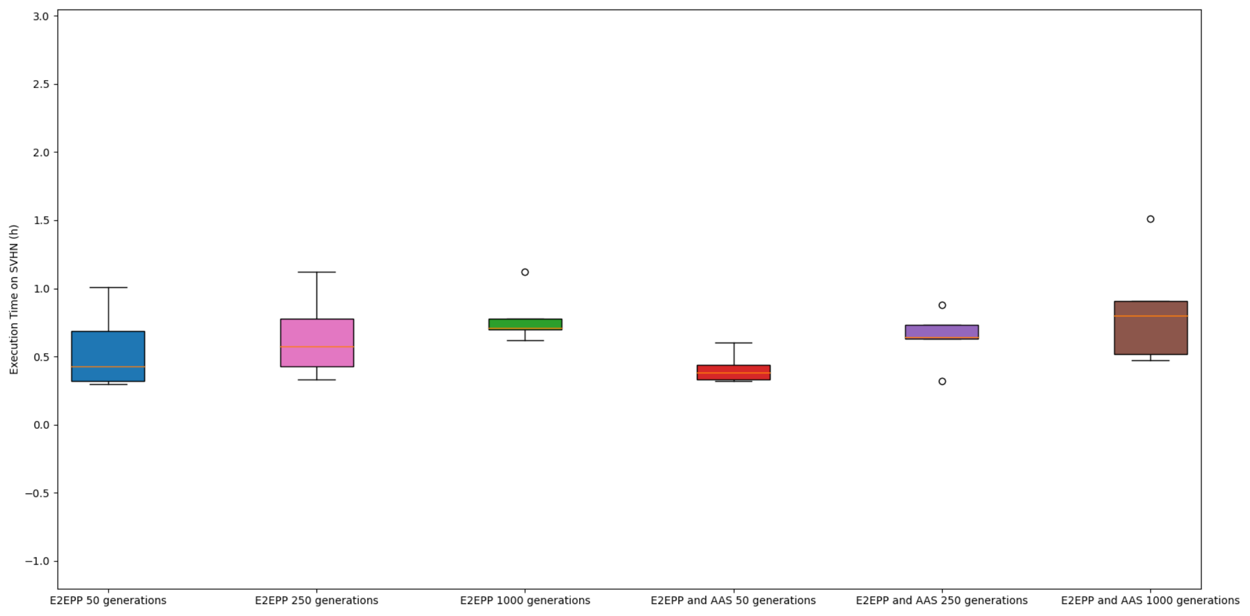


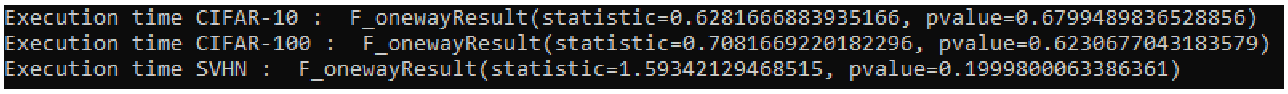
Figure 4.10: Box-plot of the execution Time on CIFAR-100 Zoom on SVHN

By looking at the the result in the figure 4.8, 4.9 and 4.10. We can see that the median is always close to 0.5 hour. So, 30 minutes of execution is really low compared to the 40 or

80 hours needed to only use CGP with a classic training. However, as expected, AAS doesn't really decrease the time needed considering that we don't need to train the CNNs.

ANOVA

Now, we can realise an One-Way ANOVA test to check the mean of the execution time of the different methods. We can see the result in the figure 4.11.



```
Execution time CIFAR-10 :  F_onewayResult(statistic=0.6281666883935166, pvalue=0.6799489836528856)
Execution time CIFAR-100 :  F_onewayResult(statistic=0.7081669220182296, pvalue=0.6230677043183579)
Execution time SVHN :  F_onewayResult(statistic=1.59342129468515, pvalue=0.1999800063386361)
```

Figure 4.11: One-Way ANOVA test on the execution time from CGP-E2EPP and CGP-E2EPP-AAS

We can see that the p-value is too high. So there is not a statistically significant difference between the means of CGP-E2EPP and CGP-E2EPP-AAS with the different numbers of generations in term of execution time.

4.5.3 Number of trainable parameters

Considering that we use AAS, we are also interested in looking if the number of parameters of our best CNN decrease or not.

Box-plot

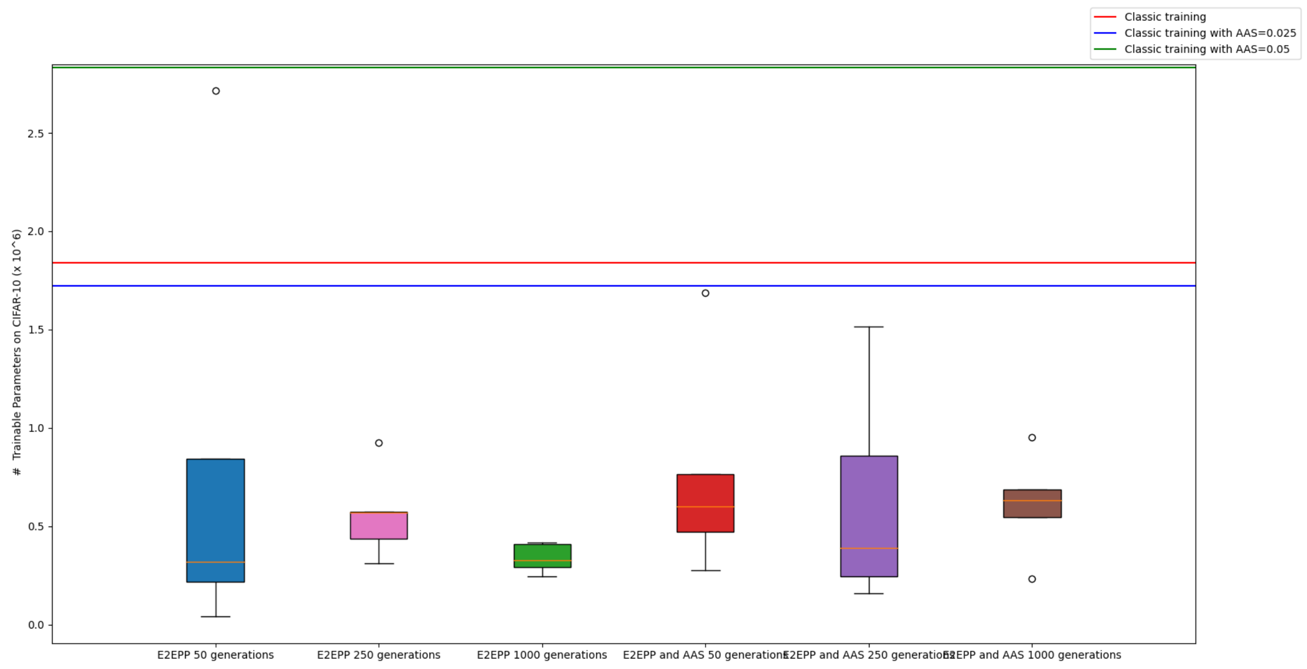


Figure 4.12: Box-plot of the number of parameters on CIFAR-10

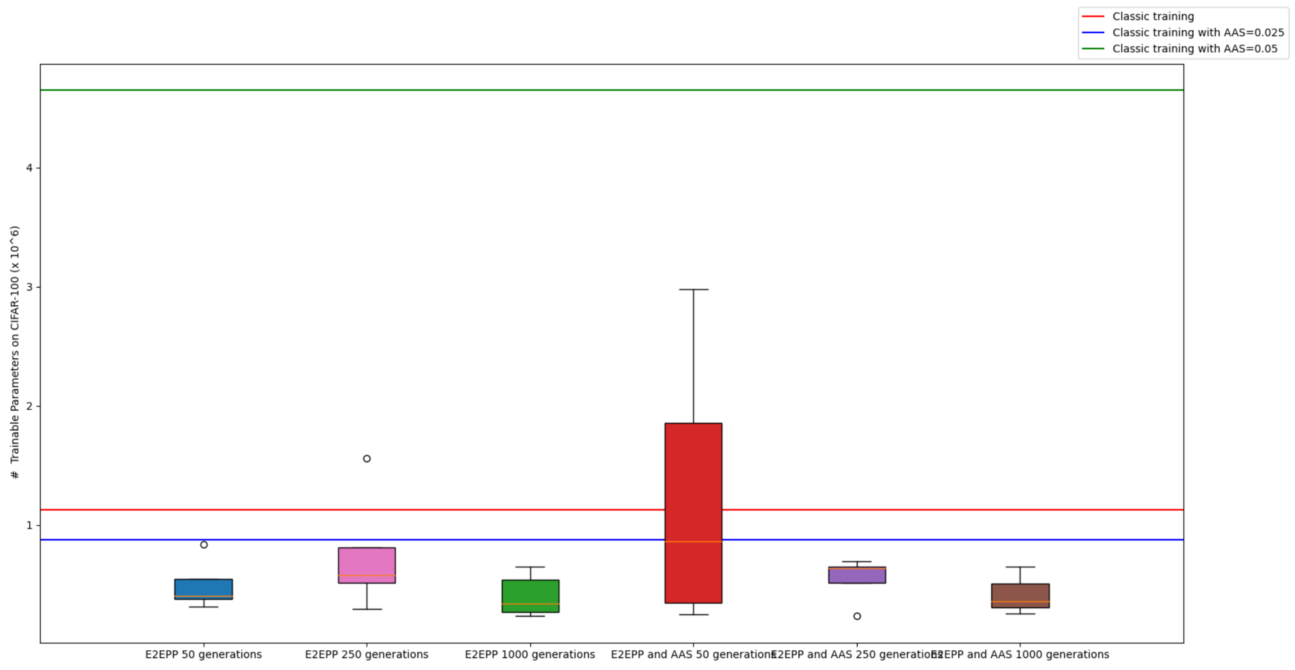


Figure 4.13: Box-plot of the number of parameters on CIFAR-100

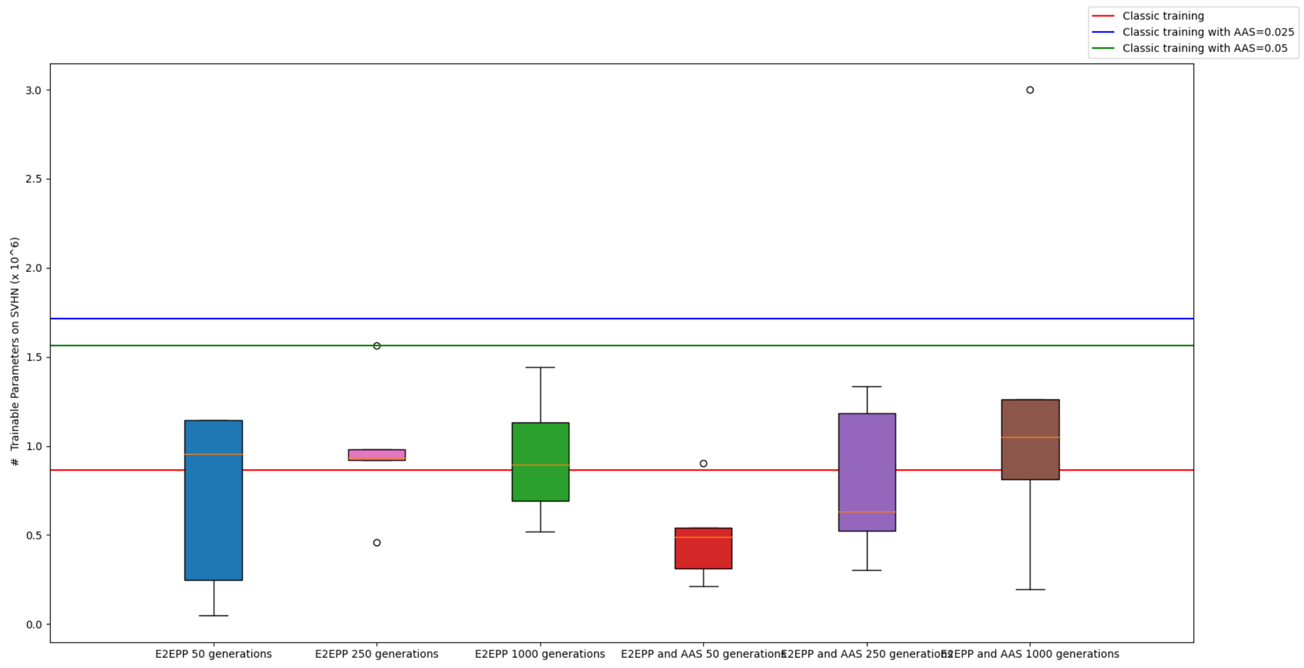
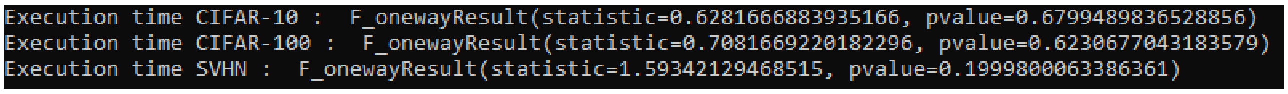


Figure 4.14: Box-plot of the number of parameters on SVHN

By looking at the figure 4.12, 4.13 and 4.14, we can see that the median of E2EPP with AAS is always equal or higher than the median of only E2EPP. We can also look at the result for the classic training with AAS or not and say that it is looking like AAS is not decreasing the number of parameters considering that at least one AAS run is always above the classic training. In the figure 4.14, the two of them are above than the classic training. However, we doesn't have enough run to really confirm this supposition. So, we can't really say that AAS is really interesting in terms of numbers of parameters considering that it doesn't lower the numbers of parameters in the CNN in general.

ANOVA

Now, we can realise an One-Way ANOVA test to check the mean of the number of trainable parameters in the different methods. We can see the result in the figure 4.15.



```
Execution time CIFAR-10 :  F_onewayResult(statistic=0.6281666883935166, pvalue=0.6799489836528856)
Execution time CIFAR-100 :  F_onewayResult(statistic=0.7081669220182296, pvalue=0.6230677043183579)
Execution time SVHN :  F_onewayResult(statistic=1.59342129468515, pvalue=0.1999800063386361)
```

Figure 4.15: One-Way ANOVA test on the number of trainable parameters from CGP-E2EPP and CGP-E2EPP-AAS

We can see that the p-value is too high. So there is not a statistically significant difference between the means of CGP-E2EPP and CGP-E2EPP-AAS with the different numbers of generations in term of execution time.

4.5.4 Comparison

We can now compare a bit ours methods with states of the art an look at the difference in terms of accuracy, GPU days and number of trainable parameters. The GPU days are the execution time divided by the number of GPU used during the experiment. So, it is an evaluation of the execution time.

CIFAR-10

We can compare our method with the states of the art on this field for the CIFAR-10 dataset and look at the result in the table 4.23. We can clearly see that ours algorithms are very fast. However, we can also see that our algorithm is really behind the others algorithms in term of accuracy.

Model	Accuracy (%)	GPU Days	# Trainable Parameters ($\times 10^6$)
ViT-H/14 ([Dos+20])	99.50	2500 (on TPU)	632
BlockQNN [Zho+18]	96.46	96	39.8
Neural Architecture Search ([ZL16])	96.35	16800–22400	37.4
Large-Scale Evolution ([Rea+17])	94.6	2750	5.4
MetaQNN [Bak+16]	93.08	80–100	3.7
Genetic CNN [XY17]	92.9	17	_
CGP-CNN ([Sug+20] ConvSet)	94.08	31	1.50
CGP-CNN ([Sug+20] ResSet)	94.99	30	2.01
CGP-E2EPP (ours)	89.66	0.026667	0.841066
CGP-E2EPP-AAS (ours)	89.86	0.017917	1.515594

Table 4.23: Comparison with the state of the art on CIFAR-10

CIFAR-100

We can also look at some results for the CIFAR-100 dataset in the table 4.24. However, there are not really a lot of information on the execution time (GPU days). So, We will not write it for the other method. However, it is in the same scale that on the CIFAR-10 dataset. We can observe that our accuracy is quite low on this very complex dataset.

SVHN

The SVHN dataset is less used. So, it is quite difficult to compare it with the same type of method. We can still compare it with the best method and with MetaQNN in the table 4.25.

Model	Accuracy (%)	GPU Days	# Trainable Parameters ($\times 10^6$)
EffNet-L2 (SAM [For+20])	96.08	_	64
CGP-CNN ([Sug+20] ConvSet)	73.3	_	2.01
CGP-CNN ([Sug+20] ResSet)	74.9	_	4.60
Large-Scale Evolution ([Rea+17])	77	_	40.4
Genetic CNN [XY17]	70.7	_	_
MetaQNN [Bak+16]	72.86	_	3.7
BlockQNN [Zho+18]	81.94	_	39.8
CGP-E2EPP (ours)	57.38	0.017083	0.377482
CGP-E2EPP-AAS (ours)	52.84	0.030417	0.653674

Table 4.24: Comparison with the state of the art on CIFAR-100

We can see that with this simple dataset, the accuracy are not really far and that the time is really lower than any other method.

Model	Accuracy (%)	GPU Days	# Trainable Parameters ($\times 10^6$)
WRN28-10 (SAM [For+20] [ZK16])	99.01	_	36.5
MetaQNN [Bak+16]	97.04	_	3.7
CGP-E2EPP (ours)	94.06	0.023842	0.981834
CGP-E2EPP-AAS (ours)	94.36	0.036667	1.336458

Table 4.25: Comparison with the state of the art on SVHN

5 Conclusion

To conclude, We were interested in NAS and how to automatically design different architecture of CNN without too much human interaction. To do so, we used a CGP algorithm as base to work with and 3 datasets (CIFAR-10, CIFAR-100 and SVHN). Furthermore, we created an new method to rate a CNN by taking into account theirs numbers of nodes in addition to the accuracy and called it AAS. This method was developed to allow us to keep the CNN with the lowest numbers of nodes compared to another CNN with more nodes and the same accuracy. It should allow a better evolution considering that it avoid the larger CNNs which are not too interesting over the smaller CNN with the same accuracy. However, it doesn't show direct results with E2EPP and couldn't be tested in depth on only CGP considering the time needed to train the CNN. This method used on top of the CGP still gave us correct results but still a long time of execution up to 4 days. So, it is not clear if it has a real impact on only CGP.

Considering the high times needed to train a CNN, it is quite logic that the execution take a long time. So to try to reduce the time needed for this method, we have also implemented, on top of CGP, a predictor called E2EPP which doesn't train every CNN in every generations of our algorithm. It allows us to reduce clearly the time of execution by more than 40 times. However, it also decrease a bit the average accuracy obtained in the experiment. This could be caused by the lack of diversity in the data given to train our predictor or by a lack of enough sample. However, collect the sample is not easy considering that we need to train every CNN to obtain their accuracy.

Lastly, we decided to try to combine our method AAS and the E2EPP predictor together and look if it works well. However, the results were not a lot different of E2EPP alone considering that it has already not a high number of nodes in the results. They were not really working well together but that could be explained by the fact that a prediction is not really exact. So, we can have close predicted accuracy but not really close training accuracy and that could corrupt a bit the evolution. Indeed, if two predicted accuracy are close and that we select the one with less nodes but his training accuracy is lower than the other CNN, this method doesn't achieve his purpose.

Our work was interesting and shows the difficulties to compromise between time and accuracy. Indeed, to have very high accuracy, we could need to let an experiment run for more than 1 months and at the end not be sure to obtain the bests results. Considering that, we were more interested in decreasing the time needed to run the experiment. By doing that, we allow people to work with normal computer and still obtain correct results after only one hour of search which is really low compared to all the states of arts which are all focusing on accuracy only. However, considering the well known architecture already found by the human, it doesn't give enough accuracy.

5.1 Future work

After this work, we could be interested in a lot of different improvement to do to our algorithm and also at some other option of work. First of all, we can look at some possible improvement on what we have already work on:

- for the predictors E2EPP, we could be interested in increasing by stage the number of sample in the data used to train this predictors and look how it impact directly the accuracy of this predictor. Furthermore, we could try to collect the data more randomly to produce different type of samples to have larger view and be able to detect bad architectures more effectively.
- We could also be interested in the use of different pattern for the nodes of the NNs and try to use the different patterns of nodes of CNNs well known for their accuracy in the CGP like VGG [SZ15] or ViT-H/14 [Dos+20] not just the 6 six simple layers that we have used in our experiment.
- Another option could be to try to add different operation into the CGP algorithm like the crossover operation and a selection with probability. However, considering that the group of parents individual need to be higher in this case, we would need to use a stronger machine and it would increase the time needed compared to our simpler algorithm.
- Evolve AAS by using the numbers of trainable parameters which is more directly related to the training time needed for our algorithm over only the size of the network.
- We could also try to use others datasets like the MNIST [Den12], the Flowers102 [NZ08] and the Oxford-IIIT-Pet [Par+12].

Furthermore, we could also try different predictors and not only E2EPP with AAS on the same evolutionary algorithm to look if AAS could be used correctly with other predictors. :

- Peephole could be one interesting option [DYL17]
- the minimal training of the convolution part from Hahn et al could also be very interesting [Hah+19].
- The most interesting option could be probably the work from Baker et al with the early termination that could really fit well with AAS [Bak+17]

Lastly, we could observe more our results to push the analyse further and use an ensemble learning algorithm [ASA+20]. Indeed, we could look at a way to use multiple CNNs produced by our algorithm, not particularly the best, and use a function of the prediction of all CNNs to choose the prediction and look if it increase clearly the accuracy. However, we should choice the CNN carefully considering that we would need to train all of them to predict and if they predict exactly the same result in every case, it would not be really interesting.

References

[ASA+20] Alam, K. M., N. Siddique, H. Adeli, et al. (2020). “A dynamic ensemble learning algorithm for neural networks”. In: *Neural Computing and Applications* 32.12, pp. 8675–8690.

[AAA17] Albawi, S., T. Abed Mohammed, and S. ALZAWI (Aug. 2017). *Understanding of a Convolutional Neural Network*. en. DOI: `10.1109/ICEngTechnol.2017.8308186`.

[Ale20] Alexander Amini (2020). *MIT 6.S191 (2020): Convolutional Neural Networks*. URL: `https://www.youtube.com/watch?v=iaSUYvmCekI`.

[Ale21] — (2021). *MIT 6.S191: Convolutional Neural Networks*. URL: `https://www.youtube.com/watch?v=AjtX1N_VT9E&t=1660s`.

[AS19] Aszemi, N. M. and D. D. D. P. Selvam (2019). “Hyperparameter Optimization in Convolutional Neural Network using Genetic Algorithms”. en. In: *International Journal of Advanced Computer Science and Applications* 10.6. ISSN: 21565570, 2158107X. DOI: `10.14569/IJACSA.2019.0100638`. URL: `http://thesai.org/Publications/ViewPaper?Volume=10&Issue=6&Code=IJACSA&SerialNo=38`.

[Ayu+16] Ayumi, V., L. M. R. Rere, M. I. Fanany, and A. M. Arymurthy (Oct. 2016). “Optimization of convolutional neural network using microcanonical annealing algorithm”. en. In: *2016 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*. Malang, Indonesia: IEEE, pp. 506–511. ISBN: 978-1-5090-4629-4. DOI: `10.1109/ICACSIS.2016.7872787`. URL: `http://ieeexplore.ieee.org/document/7872787/`.

[Bak+16] Baker, B., O. Gupta, N. Naik, and R. Raskar (2016). “Designing neural network architectures using reinforcement learning”. In: *arXiv preprint arXiv:1611.02167*.

[Bak+17] Baker, B., O. Gupta, R. Raskar, and N. Naik (2017). “Accelerating neural architecture search using performance prediction”. In: *arXiv preprint arXiv:1705.10823*.

[Bas+20] Basha, S. S., S. R. Dubey, V. Pulabaigari, and S. Mukherjee (2020). “Impact of fully connected layers on performance of convolutional neural networks for image classification”. In: *Neurocomputing* 378, pp. 112–119.

[Bod02] Boden, M. (2002). “A guide to recurrent neural networks and backpropagation”. In: *the Dallas project* 2.2, pp. 1–10.

[Boj+16] Bojarski, M., D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. (2016). “End to end learning for self-driving cars”. In: *arXiv preprint arXiv:1604.07316*.

[Bro18] Brownlee, J. (2018). “What is the Difference Between a Batch and an Epoch in a Neural Network”. In: *Machine Learning Mastery* 20.

[Cai+18] Cai, H., T. Chen, W. Zhang, Y. Yu, and J. Wang (2018). “Efficient architecture search by network transformation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1.

[Deb99] Deb, K. (1999). “An introduction to genetic algorithms”. In: *Sadhana* 24.4, pp. 293–315.

[Dee18] DeepLizard (2018). *Learnable Parameters in a Convolutional Neural Network (CNN) explained*. en. URL: `https://deeplizard.com/learn/video/gmBfb6LNnZs` (visited on 05/11/2022).

[DYL17] Deng, B., J. Yan, and D. Lin (Dec. 2017). “Peephole: Predicting Network Performance Before Training”. en. In: *arXiv:1712.03351 [cs, stat]*. arXiv: 1712.03351. URL: `http://arxiv.org/abs/1712.03351`.

[Den12] Deng, L. (2012). “The mnist database of handwritten digit images for machine learning research”. In: *IEEE Signal Processing Magazine* 29.6, pp. 141–142.

[Der17] Dertat, A. (Nov. 2017). *Applied Deep Learning - Part 4: Convolutional Neural Networks*. en. URL: `https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2`.

[Dos+20] Dosovitskiy, A., L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. (2020). “An image is worth 16x16 words: Transformers for image recognition at scale”. In: *arXiv preprint arXiv:2010.11929*.

[EMH17] Elsken, T., J.-H. Metzen, and F. Hutter (2017). “Simple and efficient architecture search for convolutional neural networks”. In: *arXiv preprint arXiv:1711.04528*.

[FD17] Fei-Fei, L. and J. Deng (2017). *ImageNet: Where have we been? Where are we going?* `https://www.image-net.org/static_files/files/imagenet_ilsvrc2017_v1.0.pdf`.

[FY19] Fernandes Junior, F. E. and G. G. Yen (Sept. 2019). “Particle swarm optimization of deep neural networks architectures for image classification”. en. In: *Swarm and Evolutionary Computation* 49, pp. 62–74. ISSN: 22106502. DOI: `10.1016/j.swevo.2019.05.010`. URL: `https://linkinghub.elsevier.com/retrieve/pii/S2210650218309246`.

[For+20] Foret, P., A. Kleiner, H. Mobahi, and B. Neyshabur (2020). “Sharpness-aware minimization for efficiently improving generalization”. In: *arXiv preprint arXiv:2010.01412.*

[For96] Forrest, S. (1996). “Genetic algorithms”. In: *ACM Computing Surveys (CSUR)* 28.1, pp. 77–80.

[FK19] Fourcade, A. and R. Khonsari (2019). “Deep learning in medical image analysis: A third eye for doctors”. In: *Journal of stomatology, oral and maxillofacial surgery* 120.4, pp. 279–288.

[Hah+19] Hahn, L., L. Roese-Koerner, K. Friedrichs, and A. Kummert (2019). “Fast and Reliable Architecture Selection for Convolutional Neural Networks”. en. In: *Computational Intelligence*, p. 6.

[He+15] He, K., X. Zhang, S. Ren, and J. Sun (Dec. 2015). “Deep Residual Learning for Image Recognition”. In: *arXiv:1512.03385 [cs]*. arXiv: 1512.03385. URL: `http://arxiv.org/abs/1512.03385`.

[HA19] Hossain, M. A. and M. M. Ali (2019). “Recognition of handwritten digit using convolutional neural network (CNN)”. In: *Global Journal of Computer Science and Technology.*

[Hua+18] Huang, G., Z. Liu, L. van der Maaten, and K. Q. Weinberger (Jan. 2018). “Densely Connected Convolutional Networks”. In: *arXiv:1608.06993 [cs]*. arXiv: 1608.06993. URL: `http://arxiv.org/abs/1608.06993`.

[IC16] Ijjina, E. P. and K. M. Chalavadi (Nov. 2016). “Human action recognition using genetic algorithms and convolutional neural networks”. en. In: *Pattern Recognition* 59, pp. 199–212. ISSN: 00313203. DOI: `10.1016/j.patcog.2016.01.012`. URL: `https://linkinghub.elsevier.com/retrieve/pii/S0031320316000169`.

[Ima20] Imam, A. (June 2020). *Neural Networks.* en. URL: `https://medium.com/swlh/neural-networks-4b6f719f9d75`.

[Jia+20] Jiang, J., F. Han, Q. Ling, J. Wang, T. Li, and H. Han (2020). “Efficient network architecture search via multiobjective particle swarm optimization based on decomposition”. In: *Neural Networks* 123, pp. 305–316.

[JM15] Jordan, M. I. and T. M. Mitchell (2015). “Machine learning: Trends, perspectives, and prospects”. In: *Science* 349.6245, pp. 255–260.

[KYK19] Kavitha, M., N. Yudistira, and T. Kurita (2019). “Multi instance learning via deep CNN for multi-class recognition of Alzheimer’s disease”. In: *2019 IEEE 11th International Workshop on Computational Intelligence and Applications (IWCIA).* IEEE, pp. 89–94.

[Kim+21] Kim, B., N. Yuvaraj, K. R. Sri Preethaa, and R. Arun Pandian (Jan. 2021). “Surface crack detection using deep learning with shallow CNN architecture for enhanced computation”. en. In: *Neural Computing and Applications.* ISSN: 0941-0643, 1433-3058. DOI: `10.1007/s00521-021-05690-8`. URL: `http://link.springer.com/10.1007/s00521-021-05690-8`.

[Koe18] Koehrsen, W. (2018). "Overfitting vs. underfitting: A complete example". In: *Towards Data Science.*

[Kri12] Krizhevsky, A. (May 2012). "Learning Multiple Layers of Features from Tiny Images". In: *University of Toronto.*

[KSH17] Krizhevsky, A., I. Sutskever, and G. E. Hinton (May 2017). "ImageNet classification with deep convolutional neural networks". en. In: *Communications of the ACM* 60.6, pp. 84–90. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/3065386. URL: https://dl.acm.org/doi/10.1145/3065386.

[LBH15] LeCun, Y., Y. Bengio, and G. Hinton (May 2015). "Deep learning". en. In: *Nature* 521.7553, pp. 436–444. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature14539. URL: http://www.nature.com/articles/nature14539.

[LeC+98] LeCun, Y., L. Bottou, Y. Bengio, and P. Ha (1998). "Gradient-Based Learning Applied to Document Recognition". en. In: p. 46.

[Li+21] Li, Z., F. Liu, W. Yang, S. Peng, and J. Zhou (2021). "A survey of convolutional neural networks: analysis, applications, and prospects". In: *IEEE transactions on neural networks and learning systems.*

[Mae+19] Maedche, A., C. Legner, A. Benlian, B. Berger, H. Gimpel, T. Hess, O. Hinz, S. Morana, and M. Söllner (2019). "AI-based digital assistants". In: *Business & Information Systems Engineering* 61.4, pp. 535–544.

[Mat+19] Mattioli, F., D. Caetano, A. Cardoso, E. Naves, and E. Lamounier (Jan. 2019). "An Experiment on the Use of Genetic Algorithms for Topology Selection in Deep Learning". en. In: *Journal of Electrical and Computer Engineering* 2019, pp. 1–12. ISSN: 2090-0147, 2090-0155. DOI: 10.1155/2019/3217542. URL: https://www.hindawi.com/journals/jece/2019/3217542/.

[McC07] McCarthy, J. (2007). "What is artificial intelligence?" In:

[Mii+17] Miikkulainen, R., J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat (Mar. 2017). "Evolving Deep Neural Networks". In: *arXiv:1703.00548 [cs].* arXiv: 1703.00548. URL: http://arxiv.org/abs/1703.00548.

[Mil+99] Miller, J. F. et al. (1999). "An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach". In: *Proceedings of the genetic and evolutionary computation conference.* Vol. 2, pp. 1135–1142.

[Mil20] Miller, J. F. (2020). "Cartesian genetic programming: its status and future". In: *Genetic Programming and Evolvable Machines* 21.1, pp. 129–168.

[Net+11] Netzer, Y., T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng (2011). "Reading Digits in Natural Images with Unsupervised Feature Learning". In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011.* URL: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf.

[NZ08] Nilsback, M.-E. and A. Zisserman (Dec. 2008). “Automated Flower Classification over a Large Number of Classes”. In: *Indian Conference on Computer Vision, Graphics and Image Processing.*

[ON15] O’Shea, K. and R. Nash (Dec. 2015). “An Introduction to Convolutional Neural Networks”. In: *arXiv:1511.08458 [cs].* arXiv: 1511.08458. URL: `http://arxiv.org/abs/1511.08458`.

[Par+12] Parkhi, O. M., A. Vedaldi, A. Zisserman, and C. V. Jawahar (2012). “Cats and Dogs”. In: *IEEE Conference on Computer Vision and Pattern Recognition.*

[Pas+13] Pascanu, R., C. Gulcehre, K. Cho, and Y. Bengio (2013). “How to construct deep recurrent neural networks”. In: *arXiv preprint arXiv:1312.6026.*

[Pas+16] Paszke, A., A. Chaurasia, S. Kim, and E. Culurciello (June 2016). “ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation”. In: *arXiv:1606.02147 [cs].* arXiv: 1606.02147. URL: `http://arxiv.org/abs/1606.02147`.

[Pas+19] Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. (2019). “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems* 32.

[Pha+18] Pham, H., M. Guan, B. Zoph, Q. Le, and J. Dean (2018). “Efficient neural architecture search via parameters sharing”. In: *International conference on machine learning.* PMLR, pp. 4095–4104.

[Pie21] Pierre, D. (2021). *cgp_cnn_predictors.* URL: `https://github.com/pdefraene/cgp_cnn_predictors`.

[PKB07] Poli, R., J. Kennedy, and T. Blackwell (2007). “Particle swarm optimization”. In: *Swarm intelligence* 1.1, pp. 33–57.

[Pra19] Prabhu (Nov. 2019). *Understanding of Convolutional Neural Network (CNN) — Deep Learning.* en. URL: `https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148`.

[Rav+16] Ravi, D., C. Wong, F. Deligianni, M. Berthelot, J. Andreu-Perez, B. Lo, and G.-Z. Yang (2016). “Deep learning for health informatics”. In: *IEEE journal of biomedical and health informatics* 21.1, pp. 4–21.

[Rea+17] Real, E., S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin (June 2017). “Large-Scale Evolution of Image Classifiers”. en. In: *arXiv:1703.01041 [cs].* arXiv: 1703.01041. URL: `http://arxiv.org/abs/1703.01041`.

[RTL09] Refaeilzadeh, P., L. Tang, and H. Liu (2009). “Cross-validation.” In: *Encyclopedia of database systems* 5, pp. 532–538.

[Ren+21] Ren, P., Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang (2021). “A comprehensive survey of neural architecture search: Challenges and solutions”. In: *ACM Computing Surveys (CSUR)* 54.4, pp. 1–34.

[Rus+15] Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei (2015). “ImageNet Large

Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3, pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[Saz06] Sazli, M. H. (2006). “A brief review of feed-forward neural networks”. In: *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering* 50.01.

[Sch15] Schmidhuber, J. (Jan. 2015). “Deep Learning in Neural Networks: An Overview”. en. In: *Neural Networks* 61. arXiv: 1404.7828, pp. 85–117. ISSN: 08936080. DOI: 10.1016/j.neunet.2014.09.003. URL: http://arxiv.org/abs/1404.7828.

[Sch19] Schmidt, R. M. (2019). “Recurrent neural networks (rnns): A gentle introduction and overview”. In: *arXiv preprint arXiv:1912.05911*.

[Sch+19] Schwendicke, F., T. Golla, M. Dreher, and J. Krois (2019). “Convolutional neural networks for dental image diagnostics: A scoping review”. In: *Journal of dentistry* 91, p. 103226.

[SSA17] Sharma, S., S. Sharma, and A. Athaiya (2017). “Activation functions in neural networks”. In: *towards data science* 6.12, pp. 310–316.

[Sil+16] Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. (2016). “Mastering the game of Go with deep neural networks and tree search”. In: *nature* 529.7587, pp. 484–489.

[Sil+18] Silver, D., T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. (2018). “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: *Science* 362.6419, pp. 1140–1144.

[SZ15] Simonyan, K. and A. Zisserman (Apr. 2015). “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *arXiv:1409.1556 [cs]*. arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

[SD08] Sivanandam, S. and S. Deepa (2008). “Genetic algorithms”. In: *Introduction to genetic algorithms*. Springer, pp. 15–37.

[Smi18] Smith, L. N. (2018). “A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay”. In: *arXiv preprint arXiv:1803.09820*.

[Sug18] Suganuma, M. (2018). *cgp-cnn-PyTorch*. URL: https://github.com/sg-nm/cgp-cnn-PyTorch.

[Sug+20] Suganuma, M., M. Kobayashi, S. Shirakawa, and T. Nagao (Mar. 2020). “Evolution of Deep Convolutional Neural Networks Using Cartesian Genetic Programming”. en. In: *Evolutionary Computation* 28.1, pp. 141–163. ISSN: 1063-6560, 1530-9304. DOI: 10.1162/evco_a_00253. URL: https://direct.mit.edu/evco/article/28/1/141-163/94990.

[SSN17] Suganuma, M., S. Shirakawa, and T. Nagao (Aug. 2017). “A Genetic Programming Approach to Designing Convolutional Neural Network Architectures”. en. In: *arXiv:1704.00764 [cs]*. arXiv: 1704.00764. URL: `http://arxiv.org/abs/1704.00764`.

[Sun+20a] Sun, Y., H. Wang, B. Xue, Y. Jin, G. G. Yen, and M. Zhang (Apr. 2020a). “Surrogate-Assisted Evolutionary Deep Learning Using an End-to-End Random Forest-Based Performance Predictor”. en. In: *IEEE Transactions on Evolutionary Computation* 24.2, pp. 350–364. ISSN: 1089-778X, 1089-778X, 1941-0026. DOI: 10.1109/TEVC.2019.2924461. URL: `https://ieeexplore.ieee.org/document/8744404/`.

[Sun+20b] Sun, Y., B. Xue, M. Zhang, G. G. Yen, and J. Lv (2020b). “Automatically designing CNN architectures using the genetic algorithm for image classification”. In: *IEEE transactions on cybernetics* 50.9, pp. 3840–3854.

[Sze+14] Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (Sept. 2014). “Going Deeper with Convolutions”. In: *arXiv:1409.4842 [cs]*. arXiv: 1409.4842. URL: `http://arxiv.org/abs/1409.4842`.

[WRP19] Wistuba, M., A. Rawat, and T. Pedapati (2019). “A survey on neural architecture search”. In: *arXiv preprint arXiv:1905.01392*.

[XY17] Xie, L. and A. Yuille (Mar. 2017). “Genetic CNN”. en. In: *arXiv:1703.01513 [cs]*. arXiv: 1703.01513. URL: `http://arxiv.org/abs/1703.01513`.

[YZ20] Yu, T. and H. Zhu (2020). “Hyper-parameter optimization: A review of algorithms and applications”. In: *arXiv preprint arXiv:2003.05689*.

[ZK16] Zagoruyko, S. and N. Komodakis (2016). “Wide residual networks”. In: *arXiv preprint arXiv:1605.07146*.

[ZAS14] Zakaria, M., M. Al-Shebany, and S. Sarhan (2014). “Artificial neural network: a brief overview”. In: *International Journal of Engineering Research and Applications* 4.2, pp. 7–12.

[Zha+18] Zhang, X., X. Zhou, M. Lin, and J. Sun (June 2018). “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices”. en. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Salt Lake City, UT: IEEE, pp. 6848–6856. ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00716. URL: `https://ieeexplore.ieee.org/document/8578814/`.

[Zho+18] Zhong, Z., J. Yan, W. Wu, J. Shao, and C.-L. Liu (2018). “Practical block-wise neural network architecture generation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2423–2432.

[ZL16] Zoph, B. and Q. V. Le (2016). “Neural architecture search with reinforcement learning”. In: *arXiv preprint arXiv:1611.01578*.

[Zop+18] Zoph, B., V. Vasudevan, J. Shlens, and Q. V. Le (2018). “Learning transferable architectures for scalable image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710.

[Zup94] Zupan, J. (1994). “Introduction to artificial neural network (ANN) methods: what they are and how to use them”. In: *Acta Chimica Slovenica* 41, pp. 327–327.