HEC MONTRÉAL

École affiliée à l'Université de Montréal

A Study on Genetic Algorithm for CNN Architecture Design : A Variant with Diverse Solution Set

by
Saba Daftari

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science (M.Sc. Data Science and Business Analytics)

August 2022

Abstract

Deep learning technics have noteworthy performance in today's numerous applied science and technology domains. To have a deep learning model working very well, hyperparameters must get tuned which is done either manually, semi-manually or automatically. Additionally, this task is a challenging optimization problem, especially for non-experts who do not have a deep knowledge of this field. Various search algorithms exist to address this problem; however, they suffer from a compromise between intensity and diversity in a full automation. In this research, we exploit genetic algorithms to find a diverse and high-performance set of solutions. Genetic algorithms are derived from natural reproduction based on elite selection and randomness. We will investigate a fixed search space on the image classification benchmark MNIST datasets to serve the purpose of comparison with other works in the literature. The novelty of this algorithm lies in the crossover operators which bring diversity to the final set.

Keywords: *Hyperparameter tuning, genetic algorithm, evolutionary deep learning, diversity, crossover, convolutional neural networks*

Acknowledgements

I would like to thank all the people who encouraged and supported me in preparing this dissertation. Including graduate students of HEC Montreal along with my friends and family, my mother, and my husband. The inspiration they gave me went such a long way in this personal journey. To my supervisors, Prof. Gilles Caporossi, and Prof. Sylvain Perron, who gave me this excellent opportunity, with their faith in my competencies and my ideas regarding this research. I gratefully acknowledge their technical guidance and financial support which I found to be so helpful. Especially during such a long pandemic that has impacted us all. It has been an honor to work with them both, to learn from their expertise and patience toward tackling a research problem.

Moreover, I would like to thank my other professors at HEC who taught me distinct aspects of Artificial Intelligence and Data Science. I would like to thank the library of HEC Montreal for their high-quality research papers in this very field which helped me tremendously.

To my parents…

Table of Contents

ABSTRACT ..... **III**

ACKNOWLEDGEMENTS ..... **V**

TABLE OF CONTENTS ..... **IX**

LIST OF TABLES ..... **XI**

LIST OF FIGURES ..... **XIII**

LIST OF ACRONYMS ..... **XV**

CHAPTER 1 INTRODUCTION **1**

CHAPTER 2 LITERATURE REVIEW ON GENETIC ALGORITHMS FOR HYPERPARAMETER TUNING **7**

2.1. Xiang & Zhining, 2019 11
2.2. Mattioli et al, 2019 16
2.3. Sun et al., 2020 19
2.4. Young et al., 2015 23
2.5. Sun et al., 2019 24

CHAPTER 3 METHODOLOGY **29**

3.1. Encoding 32
3.2. Population Initialization 33
3.3. Elite Selection Mechanism 33
3.4. Crossover Selection Mechanism 34
3.5. Crossover Operation 34
3.5.1 Fully Connected Exchange Crossover (FCE) *35*
3.5.2 A Hybrid of Point Fusion and Mixed Crossover (HPF&M-R) (Xiang & Zhining, 2019) *36*
3.5.3 A Hybrid of Point Fusion and Mixed Crossover – Collected (HPF&M-C) *38*
3.5.4 A Hybrid of Point Fusion and Mixed Crossover – Replacement (HPF&M-R) *39*
3.5.5 Fully Connected Exchange + Single Point Fusion Crossover (FCE+SPF) - firstly proposed *41*
3.5.6 Fully Connected Exchange + Two Points Fusion Crossover (FCE+TPF) - secondly proposed *43*
3.5.7 Fully Connected Exchange + mean value for two important parameters + Single Point Fusion Crossover (FCE+Mean+SPF) - thirdly proposed *45*
3.5.8 Convolutional Fusion & Fully Connected Exchange and Maximization Crossover (CF&FCEM) - fourthly proposed *47*
3.6. Mutation Operations 50
3.7. Experiment Setting 50
3.8. Design Choice 52
3.8.1 Toward a Strategy *52*
3.8.2 Improvements and Fine Tuning *53*
3.9. Challenges 54
3.9.1 Feasibility *54*
3.9.2 Fix the Unwanted Offspring Challenge *54*
3.9.3 Mimicking Human Body Challenge *55*

CHAPTER 4 DISCUSSION & RESULTS..... 57

4.0. INTRODUCTION 57
4.1. FINDINGS 58
4.2. DISCUSSION 65
4.2.1 Tables 65
4.2.2 Gene Representations 67

CHAPTER 5 CONCLUSION..... 71

LIMITATIONS 72
RECOMMENDATION 72

BIBLIOGRAPHY..... 75

APPENDIX 83

List of Tables

3.1 Search Space Restrictions……. 51

4.1 A comparison between six crossover operators……. 58

4.2 Two algorithms versus three selection mechanisms…….. 59

4.3 A comparison between two crossover operators, CF&FCEM and HPF&M……. 60

4.4. A comparison between two crossover operators and two mutation operators……... 62

4.5 Final Algorithm on its Evolution to Generation Number 4…….…….……65

Table A…… 83

Table B……...……. 83

Table C……...……. 84

List of Figures

3.1 Algorithm Flowchart……. 30

3.1.1 Algorithm 1……..……. 31

3.2 Fully Connected Exchange Crossover…….. 35

3.3 A Hybrid of Point Fusion and Mixed Crossover…….. 36

3.4 A Hybrid of Point Fusion and Mixed Crossover – Collected…… 39

3.5 A Hybrid of Point Fusion and Mixed Crossover - replacement…… 40

3.6 Fully Connected Exchange + Single Point Fusion Crossover…….. 42

3.7 Fully Connected Exchange + Two Points Fusion Crossover……. 44

3.8 Fully Connected Exchange + mean value for two important parameters + Single Point Fusion Crossover……... 46

3.9 Convolutional Fusion & Fully Connected Exchange and Maximization Crossover ……..48

4.1 Boxplot of maximum accuracy distribution, a comparison between CF&FCEM crossover and HPF&M crossover both with our mutation operations……. 61

4.2 Boxplot of maximum accuracy distribution, a comparison between CF&FCEM crossover and our mutation operations versus HPF&M crossover with Xiang & Zhining (2019)'s original mutation operations……. 63

List of Acronyms

Adagrad	Adaptive Gradient
Adadelta	Adaptive Delta
Adam	Adaptive Moment Estimation
Adamax	Adaptive Movement Estimation with infinity norm (Max)
ANN	Artificial Neural Network
Block-QNN-A	Block-wise Q-Learning Neural Network type of Block A
CGP-CNN	Cartesian Genetic Programming
CIFAR_10	Canadian Institute For Advanced Research with 10 classes
CIFAR_100	Canadian Institute For Advanced Research with 100 classes
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CS	Convex Sets
DAG	Directed Acyclic Graph
EA	Evolutionary Algorithm
EAS	Evolutionary Algorithm Search
GA	Genetic Algorithm
HPT	Hyper Parameter Tuning
IdMorph	Identity Morphism
Leaky ReLU	Leaky Rectified Linear Unit
MB	MNIST Basic
MBI	MNIST with Background Images
MetaQNN	Meta Q-learning Neural Network

MLP	Multilayer Perceptron
MNIST	Modified National Institute of Standards and Technology
MRB	MNIST with Random Background
MRD	MNIST with Rotated Digits
MRDBI	MNIST with Rotated Digits plus Background Images
Nadam	Nesterov-accelerated Adaptive Moment Estimation
NAS	Neural Architecture Search
NN	Neural Network
RAM	Random-Access Memory
ResNet	Residual Neural Network
RI	Rectangle Images
Rmsprop	Root Mean Squared Propagation
RNN	Recurrent Neural Network
Prelu	A Parametric Rectified Linear Unit
PSO	Particle Swarm Optimization
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
Tanh	Hyperbolic Tangent

Chapter 1
Introduction

Deep learning techniques have noteworthy performances in today's various applied science and technology domains. Medicine (Li et al., 2022), natural language processing, visual recognition (He et al. 2016, Krizhevsky et al. 2012, Redmon et al. 2016), speech recognition, object detection (LeCun et al., 2015), automatic game playing (Paduraru et al., 2021), recommendation systems (Dhelim et al., 2022) and many other problems are solved with the development of deep learning frameworks.

On the other hand, these different deep learning models are most effective once their hyperparameter configuration is optimal (Lentzas et al. 2019, Yang & Shami 2020). The prediction of a learning algorithm depends on the values which we refer to as parameters. One type of parameter is conventionally initialized randomly and later updates throughout the progression of the model. These values are called model parameters. The other type are hyperparameters; values that are set before the training process begins. By tuning these hyperparameters we obtain control over the output (Goodfellow et al., 2016). This task is an important problem, and as such there are companies and cloud platforms that offer tools for it (Turner et al., 2021).

One can tune hyperparameters manually for the user's own benefit if the user starts from a reasonable point or if the model does not have more than a few hyperparameters. This reasonable point or value comes from the suggestions of others who have experience with the same architecture, neural network, or the same task. However, this starting point is not available in most cases (Goodfellow et al., 2016). Moreover, in the case of neural networks the number of hyperparameters is large and the model is complex, so the traditional way of tuning them may not be the best option (Olof, 2018). In case there is a few numbers of hyperparameters, for instance, support vector machines that has one or two hyperparameters, one can easily benefit from manual tuning. Aside from manual tuning, one can use an automatic conventional method such as grid search or a random search to obtain a faster convergence (Goodfellow et al., 2016).

Convolutional neural networks are neural networks with the key that they include at least one layer in which convolution is used instead of a simple matrix multiplication. Convolution is a mathematical operation that is best applied on grid-like data, such as images where you can move over the pixels using a filter as time passes. In fact, a convolution is a weighted average operation at every moment. A dot product multiplication happens between the weight array and the input image. The first argument is called "kernel" and the second argument is the input and the output of this multiplication is called "feature map". The kernel has two dimensions at every moment as it is moving over the input image. This way allows us to detect object edges. After the convolution operator is performed, a nonlinearity e.g., rectified linear function is applied. In the end, a pooling function will execute. Pooling function summarizes the output in a statistical fashion. This function includes max pooling, min pooling and average pooling as three types. Pooling summarizes a rectangular neighborhood pixel with either their maximum, minimum or average values. Therefore, a single convolutional layer consists of a layer of convolutions, a nonlinearity after that, and finally a pooling layer. A convolutional neural network incorporates one or more of convolutional layers followed by one or more fully connected feedforward layers (Goodfellow et al., 2016).

Convolutional Neural Networks (CNNs) are known as one of the most important research fields due to their application in computer vision (Li et al., 2022), document type classification (Silva et al., 2018), historic and environmental collections (Lin et al., 2019), climate analysis (Chattopadhyay et al. 2020, Rosentreter et al. 2020), agriculture (Kamilaris & Prenafeta-Boldú, 2018), recommendation systems (Li et al. 2022, Mittal et al. 2022, Lee & Kim 2022), modeling artificial organs (The lazy programmer, 2016), etc. Subsequently, successfully tuning hyperparameters of a CNN to obtain the highest accuracy for such a practical architecture is of high interest.

Hyperparameter tuning is an optimization problem where hyperparameters are decision variables, and the validation error is the cost that we want to minimize. To estimate this cost, Bayesian regression is very often used. Model-based optimization indeed consists of a trade-off between exploitation and exploration (Goodfellow et al., 2016). Continuing onto gradient-based models, Nelder & Mead (1965) uses Gaussian method to create a new

optimization algorithm that outperforms Bayesian optimization for hyperparameter tuning. Another type of model-based optimization utilizes surrogate methods instead of gradient optimization (Nelder & Mead, 1965). In case that the objective function is not available or very costly to estimate, surrogate model defines few objective function evaluations to find a global minimum (Gutmann, 2001). To name two of this type of optimizations we have Tree-structured Parzen Estimator (Bergstra et al., 2011) and fully Bayesian treatment for EI (Snoek et al., 2012). Still, the disadvantage that Bayesian models have is that they are sequential, and so there only a small opportunity for parallelization (Loshchilov & Hutter, 2016).

There is another model-based hyperparameter optimization called population-based optimizations. These models can be parallelized easily. For instance, particle swarm optimization (PSO) is a population-based ensemble learning technique inspired by a flock of fish or bird navigation. Each solution in this algorithm is a particle. An initial population of solutions or particles are created randomly and from there, all particles begin to move. This movement is based on the particle's own information (personal best) but also its collective information (global best). All the particles store these two values and will move in a certain amount toward their personal best, then later in the same amount toward the population's global best. This movement or the 'velocity' is calculated based on this information and it will be added to the current position of the particle for acquiring a new position. In the subsequent round, a global best is declared for all, the particles new velocity is calculated based on both their own personal best and global best until now, repeating this procedure until finding the global minimum (Eberhart et al., 1995). Guo et al. (2021) has provided a distributed particle swarm optimization that outperforms the traditional PSO variant. Furthermore, in (Lorenzo et al., 2017) another distributed PSO algorithm is introduced and has proven to bring desirable results within a reasonable timeframe. On the contrary, a downside to PSO is that the convergence rate is slow and there is a high probability that the algorithm falls into a local optimum (Li et al., 2014). This happens when the initial population does not include proper solutions. This occurs when the hyperparameters are discrete (Yang & Shami, 2020).

As described previously, population-based algorithms benefit from the initial steps in genetic algorithms (Nalçakan & Ensari, 2018). Genetic algorithm (He et al., 2016), evolutionary strategy (Srinivas & Patnaik, 1994), genetic programming (Srivastava et al., 2015) are well known evolutionary algorithms from which the GA's (genetic algorithms) are mostly used due to its promising evidence (Davis, 1991). The most important feature of the genetic algorithms besides the fact that they can resolve different optimization problems (Goldberg & Holland 1988, Miller et al. 1995, Zhang et al. 2003, Anderson-Cook 2005, Malik & Wadhwa 2014), is their ability to be parallelized (Xiang & Zhining, 2019). Additionally, they are known for their bio-inspired operators such as selection, crossover, and mutation (Back, 1996). Based on the problem in hand, the operations need to be designed accordingly. This evolution over generations selects the best individuals to pass on their features to the next generation and removes low capability individuals within the same process. In the last generation the global optimum is found and the search for hyperparameters terminates (Yang & Shami, 2020).

Among the population-based metaheuristics available, genetic algorithms do not necessarily require a good initial generation, simply because they will select the best individual within the population to pass onto the next generation (Yang & Shami, 2020). Note that, often, the operators in the algorithm steps makes this automatic search technique very useful. For instance, in several papers (Guo et al. 2021, Nag & Pal 2015, Sun et al. 2020, Xiang & Zhining 2019, Mattioli et al. 2019, Young et al. 2015, Wang et al. 2018), they have their own set of operators to succeed. On the other hand, one widespread problem within these proposed algorithms is the lack of diversity for the final generation (Xiang & Zhining, 2019): meaning that most of the individual architectures are alike each other. Genetic algorithms need diversity since it prevents premature convergence. Premature convergence refers to the situation where crossover and mutation operators can no longer create a better individual which surpasses their parents (Gupta & Ghafir, 2012).

The novelty of our proposed algorithm is achieved in two parts. First one is full automation in hyperparameter tuning, meaning that there is no manual intervention from the beginning of the search until the end where we acquire a set of solutions. Second one

is mimicking the crossover operator exactly as it is in the human body. The contribution of this research is built upon the following:

1. Where most peer competitor algorithms use blocks of architecture, especially for CNNs, which in practice become complex, ineffective and cannot be generalized. We instead deploy each layer's hyperparameters from the search space therefore leading to no blocks of layers.

2. Most of the genetic algorithms focus on mutation operators and leave the crossover operators as simple as possible. In this work, we introduce crossover operations that improves upon diversification.

3. There are some papers that claim tournament selection mechanism outruns truncation (ranked) and random selection. Therefore, we decided to apply a comparative study on this matter from a point of view benefiting diversification. To claim that this algorithm is more diverse, we performed an attempt to compare the state-of-the-art proposed algorithm (Xiang & Zhining, 2019) and our algorithm.

4. Many of the hyperparameter tuning technics are not fully automated, however we designed a search algorithm that needs no manual intervention.

We begin the next chapter with a detailed description of proposals within the field of genetic algorithms for hyperparameter tuning. Chapter 3 documents the method we used to obtain this new algorithm and everything about it. Next, the findings and results of the comparisons are written in Chapter 4. Chapter 5 concludes and further suggests future work.

Chapter 2
Literature Review on Genetic Algorithms for Hyperparameter Tuning

A convolutional neural network architecture typically consists of convolutional layers followed by pooling layers, and finally fully connected layers (Wang et al., 2018). This mix of convolutional layers and pooling layers creates the first section of the architecture, and the follow up section are fully connected layers. In the convolutional layer, utilizing a filter which is a matrix, moving across the image from left to the right and subsequently downward. An elementwise multiplication is applied to this area covered by the filter on the image and the results are summed together in each input channel to create the final feature map. There are two distinct types of convolutions: 'Valid' which defines itself by framing the image with padding, and 'Same' that is an image lacking any paddings.

The designing of convolutional neural networks is divided into two categories (Sun et al., 2020). First is when the designer has knowledge and a little background regarding the CNN designing. This means that the expertise of designer can help manually tune hyperparameters and there will be some automatic tuning involved that is "automatic + manually tuning". The other one is the automatic tuning which does not require any knowledge of the designer and all the process is done automatically (He et al. 2021, Srinivas & Patnaik 1994, Mattioli et al. 2019, Hawkins, 2014). Not to mention that the manually tuning is that leads to higher accuracies (Srivastava et al., 2014). On the other hand, the automatic search is for the ones without any expertise and since many CNN users are not professionals with tuning manually then this category might be of their interest.

This automation sometimes is the automation of every aspect of machine learning workflow and sometimes is limited to hyperparameter tuning and sometimes will be neural architecture design or neural architecture search (NAS) (He et al., 2021). Every NAS is composed of three stages: search space, architecture optimization, and model evaluation methods. Search space examples are entire-structured, cell-based, hierarchical, and morphism-based.

In the entirely structured architectures, they first define the whole architectures possible in the set of search space. So, each member of the search space is in fact an architecture itself. Also, there can be arbitrary skipping in some layers in the architecture which is going to be a little more complex. On the other hand, searching for a cell structure is easier than looking for the whole structure. This introduces cell-based architectures where there are blocks that are being repeated.

Hierarchical search tries to focus on the architecture of cells as well as the architecture within each cell. This way, it is possible to explore hyperparameters of channels, feature maps for each layer and allows more complex and flexible topologies.

Identity morphism transformation is a function that operates between the layers of a network and is classified into two types: by width and by depth. Each IdMorph can be used as in width direction or depth direction. This will result in having an equal model but a deeper or wider version of it. An upgrade version of IdMorph is network morphism. In this method, a child can inherent all the information of the father.

Looking for an architecture optimization that can provide the best architecture in the search space previously defined, the architecture of a network is a static set of hyperparameters, and its optimization proceeds with trial and error. However, there are ways to automatically find the best combination for the specific task and dataset. Paper (He et al., 2021) takes the NAS problem to be a subproblem of hyperparameter optimization. There are different examples of such methods namely evolutionary algorithms, reinforcement learning, gradient descent, Surrogate Model-based Optimization, Grid and Random Search, Hybrid Optimization Method. In this review we only focus on the evolutionary algorithm.

From another point of view, aside from the evolutionary optimization techniques (Huang et al., 2017) such as EAS, there are reinforcement learning technics such as MetaQNN, Block-QNN-A (Liu et al., 2018) that are used to solve the architecture design problem. These algorithms add the feature of reward-punishing from the reinforcement learning. From the paper (E. Real et al., 2017) we know that reinforcement learning technics require more resources compared to the evolutionary algorithms such as Genetic CNN and CGP-

CNN (Suganuma et al., 2017). In these experiments, the authors observe that for example a Genetic CNN used 17 GPUs for one day to converge and a NAS algorithm with reinforcement learning RNN unit called (NASCell) (V. Le & Baret, 2017) consumes 800 GPUs in 28 days to converge the same accuracy.

The evolutionary algorithms are inspired by biological evolutions, thus is a population-based paradigm for optimization. For an evolutionary algorithm, the first task to do is defining an encoding scheme. There are two types of direct and indirect encoding schemes: Direct encoding refers to the static genotype construction. In (Xie & Yuille, 2017), the authors define a fixed-length binary string encoded for architecture design. In this paper, each node has a binary value in the string. This means the computational complexity of this encoding can be two to the power of number of nodes. On the other hand, the implementation of such encoding is simple and easy. There are also variable length encodings technics such as DAG (Suganuma et al. 2017, Real et al. 2017, Liu et al. 2017) and Cartesian Genetic Programming (CGP) (Suganuma et al. 2017, Miller & Harding 2008, Milano & Nolfi 2018). Directed Acyclic Graph refers to a list of sub-modules here. Likewise, in the paper from Real et al. (2017), a graph is used to encode the neural architecture. Each node represents a three-dimensional tensor, and each edge represents convolutions or identity connections; Indirect encoding means there is a rule to build a network. It takes architectures that are related to each other and put them in a set of labeled trees in a simple graph grammar. Furthermore, the most usage of this type of encoding is to define an operation for network morphism. This is helpful if one uses morphism-based search spaces.

A conventional evolutionary algorithm has four stages according to the paper He et al. (2021). Selection, Crossover, Mutation, and Update. In the selection stage, the goal is to eliminate the weak individuals, and to maintain and proceed with the stronger individuals. There are different mechanisms when it comes to the selection phase. One is fitness selection, in which the probability of one individual to be selected is proportional to its fitness value. The second one is ranked selections, in which they rank the individuals in the generation based on the fitness values and select the top wanted proportion. The third one is tournament selection, in which k individuals are selected randomly and then sorted

by their fitness value, and then the best individual is selected with the probability of p, the second-based individual has the probability of being selected as p (1-p). This process will be repeated until they reach the maximum number of individuals desired and in addition when it is assured that the best individual of the generation is selected.

In crossover phase, one will cross individuals to get newly built individuals. This newly built offspring has some information from each of its parents. This process imitates the genotype crossover within sex cells in human body. The operation for this crossover highly depends on the encoding scheme. Note that sometimes this operations make the genotypes damaged and taking care of that is more of an engineering and implementation matter. In cellular encoding, a random branch of the tree structured genotype will be selected for parental exchange.

A typical gene mutation is an operation that will flip a bit. We select this bit point randomly and independently. For instance, papers (Suganuma et al., 2017) and (Xie & Yuille, 2017) apply a point mutation. In Real et al. (2017), mutation operations are of two types: one decides if there should be a connection between two layers, the other decides on the skip connections between two nodes or layers. Same paper introduces a set of mutation operations such as removing a skip connection or a change in the learning rate. Equally to the real gene mutation in human beings, this mutation can be either beneficiary to the individual or may cause damage to the crossed individual. What is important is that it creates diversity in the next population. Krizhevsky et al. (2012) provide a new mutation operation that works the best for network planning. Additionally, He et al. (2016) has also proposed a new variety of the genetic algorithm by proposing a new mutation operation.

During update stage, the worst individuals are eliminated. For instance, in Real et al. (2017), two random individuals are selected and from those, the authors eliminate the individual with lower fitness. Another example is in the paper (Real et al., 2019) where the paper eliminates the oldest individuals. Other papers (Suganuma et al. 2017, Xie & Yuille 2017, Miikkulainen et al. 2019) have an interval to eliminate individuals based on it. In (Liu et al., 2017), the authors will not discard any individuals and allow the generation to evolve by itself.

Five papers are more relevant, and we will present them more deeply in the following.

2.1. Xiang & Zhining, 2019

Xiang & Zhining (2019) introduced an improved genetic algorithm from the original version that in 30 rounds of model training generations, it achieves 98.81% accuracy on MNIST data set. Note that the official sample model achieves the accuracy as large as 98.4%. The hyperparameters correspond to the architecture of a fully connected traditional neural network, in fact, a multilayer perceptron (MLP). The final model is called 'super hyperparameter model' and so this algorithm solves super-parameter optimization problem. Using a genetic algorithm to search hyperparameters, there are a few stages of selection, crossover, a mutation and in the end, there will be a fitness function that evaluates the individuals; based on those scores the paper can perform the selection mechanism for the next generation. In each of these stages, the authors store the hyperparameters of a neural network as a set of integers called Integer Coding.

1. **Integer Coding**: here the authors will optimize the hyperparameters of neural networks. However, there are many different architectures for various purposes in deep learning, our focus is on fully connected layers. A fully connected neural network or a multi perception neural network has at least one hidden layer and between the layers all nodes within each layer is connected to those of the previous and the next layers. The input and output layer of this neural network will differ from task to task, thus here the authors code the integers within the hidden layers only. The integer coding includes activation function, batch size, and number of epochs. The restriction sets are as follows: number of layers is between 1-7; number of neurons in each layer is between 1-1024; activation functions are ReLU, Tanh, PReLU, Leaky ReLU; optimization functions are SGD, RMSProp, Adagrad, Adadelta, Adam, Adamax, NAdam; and finally, batch size is between 1-256.

To continue to represent these hyperparameters as a gene sequence the authors can either have a binary coding (one-hot encoding) or to have integer encoding which will be need less resources to store. Therefore, they decided to continue with integer coding.

2. **Initialization Population**: randomly initializing the first generation for iterative evolution can yield to get some individuals with low quality. Moreover, it will affect the convergence rate of the algorithm to the best solution. If one takes the optimization function to be a multimodal function, then the individuals within an initial poor population will be scattered around the local optima and to find a better direction toward the local optima one should carefully define the subsequent genetic operators so that the authors achieve such a direction. Introducing league competition mechanism, they managed to prevent premature individuals in the initial population using the following steps:

A. Randomly initialize *N* individuals,

B. Calculate individual fitness for all the *N* individuals and sort them based on their fitness value,

C. Select the fittest individuals within the sorted list,

D. Repeat the previous steps until there are *N* individuals in the initial population. The value of *N* is around 2-5. If one takes more, then the algorithm will degenerate to random search algorithm.

3. **Truncation Selection Mechanism**: using the traditional roulette selection mechanism, combined with slight differences between the individual fitness in the population, leaves the selection mechanism baffled and inconvenient. Therefore, adding other supplementary methods to create more diversity is necessary. In the truncation selection mechanism, *k* % percent of the fittest individuals are selected, and it is crossbred with other individuals. Each individual has an equal chance to crossbreed with other individuals and both crossover and mutation operations will be done randomly. This is obvious that truncation selection mechanism has a more intense selection that the roulette mechanism so the authors make sure that all the good individuals will be used for the next generation production. In addition, when the gap between the fitness of individuals is exceedingly small, the authors do not use truncation mechanism.

4. **Crossover Operators**:

A. Fusion crossover: In the fusion crossover calculation formula below, the authors observe that one of the gene strings (gene strand) of the crossed offspring will resemble more like the first paternal gene and the second crossed offspring will resemble the second paternal gene. As much as one crossed offspring resembles one parent, it will not resemble the other parent. This amount is randomly generated from a linear combination of an exploratory coefficient and a random number between 0 and 1.

B. Single point crossover: choosing randomly a single point in the gene representation and then there will be head crossing from the chosen point.

C. Mixed crossover: a mixed crossover is a two-point crossover where the authors choose two points randomly and exchange the tail and the head of strings from the two points mentioned. This happens as follows and the new individuals are selected with the probability p and $(1\text{-}p)$. In the context of a CNN architecture, it is not possible to replace a head with a tail. Instead, one can only replace a head with a father's head and replace the tail with a father's tail. This will lead us to point fusion formula.

D. Point fusion crossover: In the above-mentioned crossovers, the genes are only partially exchanged while the value of each coding elements remains the same. In point fusion crossover the value of the two points as crossover points will be calculated and replaced with something other than the previous value from the two parental individuals. In the following you will see the Crosspoint fusion formula.

5. **Mutation Operators**:

A. In the conventional genetic algorithm, the probability of an individual being mutated is fixed in the entire population. The fixed mutation probability comes with the problem of prematurity and poor stability and not good enough results after several experiments for a specific task. A variable probability amount is calculated using the following formula:

$$P_m = k_1 * (f_{max} - f) / (f_{max} - f_{avg}) \text{ if } f \geq f_{avg}$$

$$P_m = k_2 \text{ if } f < f_{avg}$$

Where P_m is the probability to get a mutation. $k_1, k_2 \leq 1.0$, f_{max} is the highest fitness individual and f_{avg} is the average fitness in the population.

B. According to this paper, in the initial stages of evolution, a large-scale variation should be used to keep the population diverse later in the next populations. However, in the later stages of evolution, the local intensity will gradually develop for fine-tuning. Introducing the following formula for mutation operation the authors ensure the fine-tuning and diversity of the population.

Let individual $X = x_1, x_2, x_3, \ldots, x_k, \ldots, x_n$ and $x_k \in [L_k, U_k]$ (lower and upper bounds in the hyperparameter range) be the mutation point. Replacing x_k with a value from below calculated range Ω:

$$\Omega = [x_k - s(t) (x_k - L_k), x_k + s(t) (U_k - x_k)]$$

$$S(t) = 1 - r^{(1-t/T)^c}$$

T is the maximum number of iterations; t is the current iteration. C is a value between 2 and 4 and r belongs to the interval of [0,1].

6. **Evaluation**: An accuracy of classification on the test set is used to evaluate each one of the neural network individuals in the population.

7. **Update**: In this stage, the authors want to keep individuals which have bigger fitness, and to eliminate the non-desired individuals. This process directs the search to achieve converge faster. Moreover, it ensures that the best individuals will be in the next population. However, this is probable that there exist the same identical individuals existing in one generation as a parent and the same individual as the offspring in the next generation. To cope with phenomena, the authors only keep one identical individual per population.

8. **Domain Search**: when the iteration remains still for a while in the algorithm this means that the algorithm is trapped in a local optimum. In particular, this means the individual is not changing over the iterations of population. Addressing this phenomenon, this paper adds 1 and subtracts 1 to the gene encoded and evaluate the newly created two individuals to see if the accuracy level will change. If this newly created individuals have higher accuracy, then they will be replaced with the stagnated gene. New individuals X' and X'' are defined in the following:

For X', when $U_{imin} \leq x_i < U_{imax}$

$$X' = (x_1+1, x_2+1, x_3+1, \ldots, x_i+1, \ldots, x_n+1)$$

When $x_i = U_{imax}$

$$X' = (x_1+1, x_2+1, x_3+1, \ldots, x_i, \ldots, x_n+1)$$

For X'', when $U_{imin} < x_i \leq U_{imax}$

$$X'' = (x_1-1, x_2-1, x_3-1, \ldots, x_i-1, \ldots, x_n-1)$$

When $x_i = U_{imin}$

$$X'' = (x_1-1, x_2-1, x_3-1, \ldots, x_i, \ldots, x_n-1)$$

9. **Experimental Settings**: The target dataset is MNIST, the total number of population generation is 30. The hyperparameters to be tuned are for fully connected neural networks. The size of initial population is 50. The number of epochs is a constant number equal to 20. In each truncation selection phase, the k value is 24%. The total running time of the algorithm is 144 hours. The k_1 in formula for mutation probability is 0.5, k_2 is 0.5, and the c = 3, r = 0.2 in mutation formula. The number of individuals obtained by truncation selection is 12. The optimal individual coding of this generation is 5-2-970-802-417-0-0-0-0-0-121. After decoding, it is a fully connected neural network with three layers (excluding the output layer). The number of neurons in the input layer was 970, the number of neurons in the first hidden layer was 802, and the number of neurons in the second hidden layer was 417. The optimizer function uses Adamax

function, the activation function uses PReLU function, and batch_size is 121. After 20 rounds of training, the accuracy of the model is 98.56%. The fitness of all individuals selected in the first generation is above 0.982, which has a good fitness value. Moreover, the worst individual in the initial population has a fitness of 0.9733, which shows that the way of league competition initialization plays a role in improving the quality of the initial population. Among the 12 individuals selected in this generation, Admax is the dominant choice of optimizer function, followed by Adam function, Adagrad function, RMSProp function and Adadelta function. In terms of activation function, ReLU and PReLU are the two main activation functions. In terms of the number distribution of neurons in each layer, there was no obvious characteristic distribution of the number of neurons in each layer. In terms of network layer information, the number of individuals with one layer is the largest, reaching 3, followed by those with three, four and five layers, reaching 2, and the number of individuals with two, six and seven layers is the smallest, only one. From the above analysis, one can see that the genotype distribution of the selected population is scattered, and there is no obvious pattern characteristics.

2.2. Mattioli et al, 2019

Mattioli et al (2019) originally compares the different methods of tuning hyperparameters such as grid search, random search, and genetic algorithm. We have the following steps defined for the genetic algorithm process:

1. **Chromosomes**: In this study, each chromosome is composed by two main descriptors. One for convolutional layers and the other for fully connected layers, each of these descriptors is a vector of strings and each string is a description of a layer. For convolutional layer, the description refers to activation function, number of filters, kernel size, stride, and pooling size. For fully connected layer there are the number of neurons and the dropout probability.

2. **Selection Mechanism**: a standard roulette wheel is used to select the individuals for crossover phase. This means a selection of an individual with a probability proportional to its fitness value.

3. **Crossover Operators**: A random point is selected, and the operation is defined as swapping layers between parent individuals from that point. Two points are selected in the fully connected part and two points are selected in convolutional parts. The two points mentioned each are selected from one parent individual. Then swapping the tail of the individuals from those selected points separately in convolutional and fully connected parts.

4. **Mutation Operators**: Just as in crossover phase, each part of convolutional part and fully connected layers are mutated independently. There is 50% chance for both mutation operations to be operated on each crossed individual. A random position will get picked and from there a layer will be added. This layer to add is built randomly.

After mutation phase terminates, the authors evaluate the valid individuals, and discard invalid individuals.

5. **Evaluation**: The fitness function is built upon the test accuracy of the individual. After the individual is being evaluated, the individuals will be stored with their fitness values to prevent re-evaluating an identical individual later in the next generations.

6. **Update**: Taking only 10% of the top fittest individuals in the update stage.

7. **Experimental Setting**: The dataset used in this paper is MNIST and CIFAR_10. We have an initial population of size 200. We continue the algorithm for 20 generations. There is a crossover probability of 70% and a mutation probability of 20%, meaning some of the individuals will not have crossover or mutation.

8. **Genetic Algorithm + Fitness Predictor**: There is a strategy with by which the authors can predict if an individual worth evaluating on the test set, since evaluation on the test set is highly computationally expensive. To accomplish this prediction, this paper uses an artificial neural network using the data of the previous generations to predict the accuracy of the current individuals. This ANN is in fact a multi-layer perceptron having a fixed architecture of three dense layers, has 64 neurons per layer with a dropout probability of 0.4. The input of this MLP is a new gene descriptor of the previous generations containing their fitness value. This fitness predictor is trained for 100 epochs

with an early stopping of 10. The dataset is split 80%-20% for train and test datasets. Finally, the decision is made based on two factors: the performance of the classifier and the number of weights which implies the number of parameters. Going stepwise there will be:

1. Randomly generating 10 initial individuals.
2. Dominant solutions to be selected from these 10 individuals.
3. Using the 10 solutions, the authors train the fitness predictor. 8 individuals for training set and 2 individuals for the test set.
4. After repetitive randomly generation of individuals, a new generation of size 200 will be achieved.
5. Predicting the fitness of newly built generation. This fitness predicted combined with the number of parameters of one individual will determine whether to train the neural network corresponding to the individual.
6. If the individual is worth evaluating, then there is a 90% probability that it will be evaluated, otherwise there is only 10% probability of it being evaluated. This ensures that only better solutions have access to the resources, resulting in prevention of unwanted computations.
7. Now updating the predictor with newly evaluated dominant set.
8. Now updating the set previously created in the 2nd stage with the dominant evaluated individuals.

While not reached to twenty generations, continue the following steps:

9. Taking only top twenty individuals, 10% from the last generation.
10. If these individuals have not already been trained, they will.

11. While not having 200 individuals, the authors select individuals using roulette selection mechanism.

12. Crossing the individuals with a probability of 70%

13. Mutation of the individuals with a probability of 20%

14. Replacing the previous individuals with their mutated versions.

As a conclusion from this paper, the genetic algorithm variation presented performs competitive with the grid and random search. The GA + fitness prediction has promising reduction in resources usage in one of the datasets.

2.3. Sun et al., 2020

Sun et al.'s (2020) novelty is the complete automation of the genetic algorithm proposed. A summary of innovation would be firstly this algorithm uses variable-length encoding that represent the individual's architecture features. Secondly, to cope with the vanishing gradient problem with deep networks, this algorithm provides skipping connections. And thirdly, this algorithm uses asynchronized distribution of resources that leads to using less computational resources.

1. **Convolutional Neural Networks**: These networks apply the filters on data so that one can extract features of input. In the convolutional layer, the filter slightly slides horizontally and then it slides vertically. The step by which this filter slides, is called "stride". After this operation, the data becomes smaller in size, so it is easier to store and make use of it later since the images are large and hard to store all its information. This will keep only the most key features of the data compared to keeping all the pixels exists in the original image. These key features or filter outputs are converted as a new matrix namely feature map. In fact, the output of each filter is either the mean or the max value of the projection of the filter on the image.

There is another parameter to set around the image that plays the role of padding around the image, this is to help extracting features of the borders of the image. Thus, for every

convolutional layer there are some hyperparameters; filter size or kernel size, stride, padding and number of features.

After convolutional layer there is a pooling layer. The purpose of a pooling layer is firstly reducing the spatial size of the feature map so less amount of computation required later. There are various kinds of pooling such as max pooling, average pooling, and global pooling which down samples the entire feature map into only one single value. The hyperparameters to set in a pooling layer would be the size of its kernel which is referred to it as pooling size.

2. **Skip Connections**: sometimes we want to skip some layers and to connect some layers that are not adjacent. Originally, this method has been introduced as a method to prevent gradient vanishing or gradient exploding in the deep recurrent neural network units such as long short-term memory (Srivastava et al., 2015). In fact, the superior performance of the ResNets, is again because of the skip connections.

The genetic algorithm steps are descried as follows:

1. **Encoding**: In this architecture there are convolutional blocks consisting of two layers, pooling layers, and skipping layers. For convolutional layer, its descriptor indicates the number of channels respectively for its two layers. For pooling descriptor, there is the type of pooling and for the skipping layer there is the string of the feature maps from the last convolutional layer. The code that represents the total individual is the concatenation of these descriptors mentioned earlier. A block of two convolutional layer in a row in a skip layer is firstly proposed in ResNet. It is proven that this block works very well and has promising result (Huang et al., 2017) (Liu et al., 2018) (E. Real et al., 2017) (He et al., 2016). In Sun et al. (2020), the size of two convolutional layer in the same block can be unequal as opposed to the original version in ResNets. The stride, kernel size and convolutional operation are the same for all the convolutional layers. The attributes of the pooling layers are also the same in all the pooling layers.

2. **Selection Mechanism**: a random number is generated to decide whether the pair selected individuals are going to cross or not. There is a predefined probability of crossing

defined earlier so that one can compare it to this random number generated between (0,1). If the random number is higher than the predefined crossing probability, the algorithm will not proceed with crossing those two individuals. A binary tournament selection is used in this paper to select the individuals for the crossover phase.

3. **Crossover Operators**: the crossover operation is a random split in each parent individual and swapping the tails to generate the offspring. This crossover operation is inspired by Srinivas & Patnaik (1994), with a difference of the ability of the new crossover to cross unequal length individuals.

4. **Mutation Operators**: Again, to decide whether and individual will mutate, a random number is generated in (0,1). If this number is below the predefined mutation probability, then there is the mutation performed on this individual. During the mutation, one point is selected randomly from the individual encoding, and the authors will select one mutation operation from the five different operations with a predefined probability for the mutation operations. The different mutation operations are as follows:

   A. Adding a skip layer with random settings,

   B. Adding a pooling layer with random settings,

   C. Removing a layer at the selected point,

   D. Randomly changing the value of the point selected in the encoded string.

By having higher predefined probability for adding a layer in this phase and having removed a layer as one of the other operations, the authors ensure the achievement of optimal depth for our network individual.

5. **Fitness Evaluation**: here there is also a pool of individuals from the previous generations with their fitness value stored in this pool to prevent re-evaluating the same individual in the later generations. The criteria for calculating the fitness evaluation are the classification accuracy of the CNN individual. Also, allowing the algorithm to store the individual with the highest accuracy of all.

6. **Update**: this stage is the same as the selection phase described earlier, however, the authors also check if the best individual of all time is not in the selected pool, then the authors replace the best individual with the worst individual of the selected pool. This paper believes that only selecting the top best individuals will lead the optimization process to trap in the local optima (Davis 1991, Goldberg & Holland 1988). On balance, a good population will have to have good individuals as well as the bad individuals to ensure the diversity (Anderson-Cook 2005, Malik & Wadhwa 2014). Therefore, the authors use the binary tournament selection (Miller & Goldberg 1995, Zhang et al. 2003). This paper also claims that if they do not explicitly add the best individual to the population if it is not there by nature, then the algorithm will not converge.

7. **Experiment Setting**: The dataset used in this paper is CIFAR_10 and CIFAR_100. The predefined probability of crossover mentioned before is 0.9 and the predefined probability of the mutation phase is 0.2 borrowed from (Thomas Bäck, 1996). The optimization in backpropagation is the stochastic gradient descent with a learning rate of 0.1 with a learning rate decay of 0.1 in three chosen epochs: the 1^{st}, the 149^{th}, 249^{th}. The momentum value used is 0.9 and the number of epochs is 350. Just like the state-of-the-art versions of CNNs, the number of feature maps is selected to be {64, 128, 256}. The mutation probabilities for different mentioned mutation operations are 0.7, 0.1, 0.1, 0.1.

Some papers believe that it is good to for the neural network architecture to not deploy any fully connected layers (Sun et al., 2020). They argue that the fully connected layers easily result in overfitting (Hawkins, 2004). This happens mostly because of all the edges existing between the nodes of each layer (Srivastava et al., 2014). To overcome this problem, a dropout technic has been introduced. This dropout probability is in fact a parameter added to the number of parameters which needs to be tuned. Additionally, there is number of neurons and the number of fully connected layers as hyperparameters to tune. Resulting in a larger search space which makes tuning harder.

On the other hand, there are other papers (Basha et al., 2020) that has been carefully designed and performed experiments to see the impact of fully connected layers on the CNN efficiency and they are in fact very optimist about this kind of layers. If the CNN is

shallow, then the fully connected layers should have more neurons and more layers. If the CNN is deep, then the network requires few neurons in the fully connected layers notwithstanding what dataset is used. Overall, deep CNNs have better performance with deeper datasets, and once there is a small dataset, a shallow CNN performs better than a deep one.

2.4. Young et al., 2015

Young et al. (2015) apply genetic algorithm for neural network topology design. As a result, the genetic algorithms enhance optimization for hyperparameter tuning also provides a directed search when using the previous solutions. Now, the followings are the details of this algorithm:

1. **Encoding**: there is a limited search space defined for each hyperparameter and there is a gene representation to describe an individual. Using a unified distribution, the authors randomly initiate the first generation just like a random search.

2. **Selection Mechanism**: selecting only the individuals that have fitness value higher than the average of their generation will determine which individuals will be passed to the crossover and mutation phases.

3. **Crossover Operators**: for each two consecutive individuals, the authors generate a random number. If this random number is higher than 0.6 then they cross the two consecutive individuals, otherwise, the paper will proceed to the next two consecutive individuals. The crossing point is uniformly and randomly selected between [0,6] and a simple tail swap will be operated as the crossover operation.

4. **Mutation Operators**: again, the authors pick a random number between [0,1] and if this number is higher than 0.05, then they pull out randomly and uniformly from the search space and replace the corresponding gene encoding of the individual with our pick.

5. **Evaluation**: a simple classification test accuracy is used to evaluate an individual.

6. **Experimental Settings**: There are 6 hyperparameters to tune. The algorithm will run for 35 generations. There are 500 individuals per generation. The probabilities of crossover and mutation are 0.6 and 0.05, respectively. There are some fixed parameters in all the individuals described as: base leaning rate equal to 0.001, momentum value of 0.9, weight decay of 0.004, the learning policy is fixed, maximum number of iterations equal to 4000. There is CIFAR_10 as the dataset used. The architecture consists of convolutional layers as well as pooling layers so the hyperparameters to tune are the kernel size with a range [1,8] and number of filters [16,126].

2.5. Sun et al., 2019

Sun et al. (2019) takes a close look at evolutionary algorithm on CNNs which is called EvoCNN. The details of the algorithm steps are as follows:

1. **Encoding**: The first part of the gene representator describes the convolutional layer with hyperparameters of the filter width, the filter height, the number of feature maps, the stride width, the stride height, the convolutional type, the standard deviation, and the mean value of filter elements and then there is the pooling layers descriptor presenting the kernel width, the kernel height, the stride width, the stride height, and the pooling type (average pooling, max pooling) and the last is for the fully connection layers with hyperparameters of the number of neurons, the standard deviation of connection weights, and the mean value of connection weights. A variable-length encoding is applied here because it adds flexibility toward creating deeper networks which is desirable.

2. **Population Initialization**: starting from the first part with one convolutional layer followed by the second part which is one fully connected layer. Two random numbers will decide upon the length of each of these two parts. Note that the convolutional part also includes pooling layers in it. Deciding on whether to add a pooling or a convolutional layer after the first convolution is obtained by binary coin random flip. The hyperparameters of each layer is chosen randomly.

3. **Slack Binary Tournament Selection Mechanism**: there are two different thresholds when it comes to decide upon the selection between two randomly chosen individuals; one for the mean value and the other for the number of parameters. If the mean value difference between two individuals is higher than a threshold, then the bigger one is selected. Otherwise, if the gap between the number of parameters of the two is higher than a threshold then the individual with smaller number of parameters is selected. Otherwise, an individual with smaller standard deviation is selected. If nothing worked, then they select one of them with the probability of 50%.

4. **Crossover Operators**: in crossover operation the chromosome with the smaller length, will be fully exchanged with the same length from the longer chromosome. This process will be operated separately for each of the three convolutional layers, fully connected layers, and the pooling layers, then combining them to get the crossed offspring.

5. **Mutation Operators**: there are three operations; adding, deleting, or modifying. First, a mutation point will be chosen from the chromosome length. In adding, a layer will be selected with a probability of one third from each of the three parts in a chromosome and will be added to the chromosome. In case of modifying, based on the selected point the authors will modify all the hyperparameters within that layer. Finally, for the deletion, the selected point will remove the layer.

6. **Evaluation Criteria**: by these criteria there will be a quantitative measure to determine which individual qualifies as a serving parent. Fitness function is in fact the classification error as well as the number of connection weights. Training an entire population with each individual that has around 100 epochs to train, is very resource demanding. To handle this, the authors train each network with a small size of 5 to ten epochs, then a mean value and a standard deviation is calculated for on each batch. The bigger the mean value the better the individual, if they are the same in mean value then the bigger the standard deviation the better the individual.

7. **Update**: in this stage, the authors select a fraction of top best individuals within the mutated pool. After this the authors will continue to the binary tournament selection phase.

8. **Experimental Settings**: there are nine benchmark datasets used in these set of experiments for EvoCNN. They are the Fashion (Lin et al., 2019), the Rectangle (Chattopadhyay et al., 2020), the Rectangle Images (RI) (Chattopadhyay et al., 2020), the Convex Sets (CS) (Chattopadhyay et al., 2020), the MNIST Basic (MB) (Chattopadhyay et al., 2020), the MNIST with Background Images (MBI) (Chattopadhyay et al., 2020), Random Background (MRB) (Chattopadhyay et al., 2020), Rotated Digits (MRD) (Chattopadhyay et al., 2020), and with RD plus Background Images (MRDBI) (Chattopadhyay et al., 2020) benchmarks. According to the experiments by Young et al. (2015), there are 100 generations where each generation has 100 individuals within. The elitism rate is 20% for the top individuals. Crossover probability is equal to 0.9 and the probability of mutation is 0.1. Layers from each three distinct types, can be as deep as 5 layers each at most. The train-test split in the datasets are different in each dataset. The filter width and heights are set to be equal for kernel, pooling filter, and stride. Therefore, there is only square filters. The value for kernel size and pooling size followed by it is the same. The algorithm is implemented using TensorFlow library in python. For each dataset, there are 30 different runs and the mean value for them are estimated to report as the results.

All things considered, Young et al (2015) suggests using an asynchronous evolutionary algorithm for a better usage of all the resource available. Furthermore, the authors suggest adapting an early stop framework to spot the architectures that has deficient performance before evaluating them.

Mattioli et al (2019) suggests putting effort on evaluation of the method proposed on, for instance, image regression problems. Additionally, enhancing the fitness predictor proposed in the paper.

From authors Sun et al (2019), we know that the bigger the dataset, the more reliable the results but one needs a more efficient fitness evaluation since it is computationally costly. Also, they consider using GA to tune hyperparameters of RNNs.

In paper Sun et al (2020), it has been said that although there are two components proposed and deployed to reduce computation, more methods should develop to solve computationally expensive optimization problems.

According to Xiang & Zhining (2019), with truncation selection or ranked selection comes the problem of diversity and the benefit of intensity. This phenomenon may root in distinct stages of a genetic algorithm besides selection mechanism (Gupta1 & Ghafir2, 2012). In the next chapter we will continue with a new method to address this very gap.

Chapter 3
Methodology

As earlier mentioned in the literature review, there are factors contributing to a sacrifice of diversity for further intensity within optimization issues. This problem expressed itself specifically for metaheuristic algorithms. The following body of work will establish the introduction of a new version of genetic algorithm that addresses diversity problems and analyze with greater accuracy.

We observed that previous works used varied selection mechanisms to cope with lack of diversity. In this work, on the other hand, we focus on crossover operators, and we think that to create diversity we need to modify the crossover operation by which we can create individuals that are different from each other and with their parents while using the solutions from previous generations to ensure a direct search. We also compare various selection mechanisms to see which one works better in terms of diversity.

We use some classic mutation operators as well as some new ones. In each stage of an evolutionary algorithm, we made changes to create diversity. These changes altogether make a new algorithm that yields set of solutions that is demonstratively more diverse. This retains fitter individuals when compared to the previous algorithms. We extensively test our algorithm until we reach a better combination of phases of selection, crossover, and mutation operations which advances both accuracy and diversity. Having mutation operation and selection mechanism fixed, we then explore the difference between crossover operation results. To this end, we use the MNIST dataset, to compare with other papers. Note that this algorithm is semi-automatic, and it does not require human intervention for the most parts.

The diagram below shows the process of a genetic algorithm. First, we initialize a population randomly and we evaluate the individuals within that population.

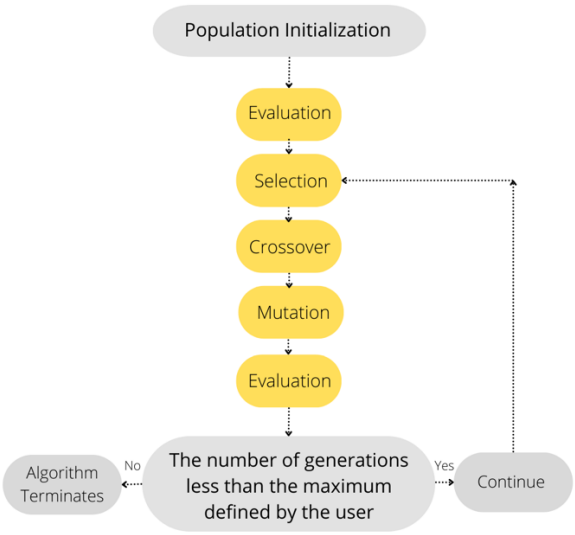

Figure 3.1: *Algorithm Flowchart*

We then loop over selection operator, crossover operator, mutation operator, and the evaluation of the individuals obtained in this iteration until a certain number of generations defined by the user is explored.

Algorithm 1 shows our algorithm below which has two main inputs: the input dataset and the number of generations or the number of algorithm iterations stored in a parameter called *N*. To start, we randomly generate the initial population called G_0 and make a copy of it and store is as *GlobalPool*. The *GlobalPool* helps us not to evaluate the same individual repeatedly later. We then start a loop for *N-1* times. In this loop, we first evaluate the individuals of the previous generation only if the individual is distinct (meaning that it is not found in the *GlobalPool*). Otherwise, instead of evaluating the same individual another time, we simply take its fitness (accuracy) from the *GlobalPool.* Second, we select the individuals that we want to pass to the next generation. Third, we select the exact two individuals for which we want to perform the crossover operation. Fourth, we cross the selected individuals. Fifth, we mutate the individuals that are out of crossover phase and put them in a set called G_{n+1}. We then pass G_{n+1} to Elite Selection for

the next iteration. Sixth, we add valid mutated individuals to the *GlobalPool*. Once this loop is over, we return the individual with the highest fitness as our final designed neural network architecture for this supervised task.

Algorithm 1 Convolutional Fusion & Fully Connected Exchange and Maximization Crossover Algorithm Framework

Input: A dataset, the maximum number of iterations.
$N \leftarrow$the maximum number of iterations
$G_0 \leftarrow$Random initialization and evaluation of the first generation and store individuals in an array where each element is an encoded individual
$GlobalPool \leftarrow G_0$
for n in N-1 **do**
  if individual I in G_n is not in $GlobalPool$ **then**
    Evaluate I
    Add I to the $GlobalPool$
  end if
  Elite Selection of the individuals that pass down to the next generation
  Crossover Selection, a selection of two parents to crossover
  Crossover Operation on the two parents selected in crossover selection
  $G_{n+1} \leftarrow$Mutation Operation on the individuals
end for
Return the individual with the highest accuracy in G_n

Figure 3.1.1: *Algorithm 1*

In the following, we first explain how each of these steps of our proposed algorithm works in detail. Consequently, we explain the experiment settings and finally we will explain our strategy to develop all these operators within this algorithm obtained over time through careful sets of experiments.

As we begin to write the first step of this algorithm, we must first think about how to describe individuals within a population. Each of these solutions is referred to as 'individual' for this work retains a variety of distinctive features. These new features describe an architecture of the neural network we wish to build, accordingly, the name for that feature vector is encoding.

3.1. Encoding

A convolutional neural network has three main layers: convolutions, pooling, and fully connected layers. Each of these layers has its own hyperparameters and the corresponding value of each hyperparameter is stored in an element of a list in the following order that makes the individual's representation. The data structure used to restore this representation is a list of the following hyperparameters in order:

1. Number of convolutional layers,
2. First convolutional layer's kernel size,
3. First convolutional layer's strid,
4. Pooling kernel size,
5. Activation Function,
6. Second convolutional layer's kernel size,
7. Second convolutional layer's stride,
8. Third convolutional layer's kernel size,
9. Third convolutional layer's strid,
10. Number of fully connected layers,
11. Number of neurons, dropout probability,
12. Fourth convolutional layer's kernel size,
13. Fourth convolutional layer's strid,
14. Fifth convolutional layer's kernel size,
15. Fifth convolutional layer's strid,

16. Number of epochs,

17. Accuracy or fitness.

Each of these hyperparameters has its own range. The Searching Space Restrictions is defined later in the experiment setting chapter.

3.2. Population Initialization

The first population is created randomly from the hyperparameter ranges. This way, a population with ten individuals is created. The first generation is devoid of any parental information for the individuals. They are distinguished as a unique orphan generation among all others. The next generations have individuals that hold data regarding both the parents and the individual itself. The value of accuracy or fitness remains a value of none, until we train and test the individual to store its fitness through its representation encoding.

Some of the experiments are designed to have the same first initial population while others do not. We report both for fair comparison of the algorithms. The reason behind fixing exact values of hyperparameters for our first population when running disparate algorithms is to strictly evaluate performance. When randomly initiating the first population, we will sometimes start with highly accurate individuals, so it helps the algorithm converge faster on its conclusive results. However, if the other algorithm begins with individuals possessing low accuracy, then it is highly probable that the algorithm converges at a much slower pace.

3.3. Elite Selection Mechanism

There are three varied selection mechanisms used in this work and we intend to compare these selection mechanisms' performance.

- First one is "binary tournament selection" borrowed from Sun et al. (2020), where two individuals are randomly selected and the one with higher accuracy is taken.

- Second one is “ranked selection” that selects the top half of the generation within fitness ranking.

- Last one is a “random selection”. A selection in where an individual is selected one at a time, until we have the same number of individuals that we possess inside the previous pool.

3.4. Crossover Selection Mechanism

For the crossover phase, we randomly select two parents from our so-called “good pool” we compiled through the elite selection phase. In this random selection, there is a randomness in which of these individuals will later crossover. Moreover, there is yet another mean for crossover selection, that is taking every two individuals in a row and subsequently progressing onto the pair alongside it. This way, we make sure that the chance of selection for a crossover among the individuals in the “good pool” remains equal. Therefore, each individual has the chance for selection at least once. However, for our intended algorithm to work, we choose to allow randomness to dictate this process, since we only wish for enhanced diversity. One downside to this process is that we may neglect the best individual in the pool, but after further experimentation we obtained our highest diversity yet through this random selection process.

3.5. Crossover Operation

Crossover is a phase in human body sex cells where the chromosomes cross to exchange genetic information between the parents, and it is a stage in genetic reproduction. In genetic algorithm is it normally defined as an operator that has two inputs as parents and has two or more outputs as crossed individuals. One type of crossover treats the encoding as an altogether single part, but some others break the encoding into two pieces according to the architecture of CNNs. In this chapter we will explain eight distinct types of crossovers. The goal is to find the crossover that yields into more diversity. Section 3.5.2 is an adapted crossover from Xiang & Zhining (2019), and the rest of the crossovers are proposed to find the best. The winner of this comparisons explained in Section 3.5.8, begins with switching the fully connected parts between two parents. This then creates

two new individuals. From these two, we switch their respective tails for the convolutional parts from a certain random point. The ensuing results declare four new individuals.

3.5.1 Fully Connected Exchange Crossover (FCE)

In this operation the encodings are of two parts: convolutional and fully connected. Convolutional part consists of storage for the hyperparameters corresponding to the layers of convolutions followed by pooling. Fully connected part contains storing the hyperparameters of the last part of any convolutional neural network where we have one or more feed forward layers. After this layer, an activation function is used to obtain the probabilities of the image belonging to one of the classes. A Fully Connected Exchange simply means two individuals will exchange their fully connected layers with each other. An illustration of this operation is the following.

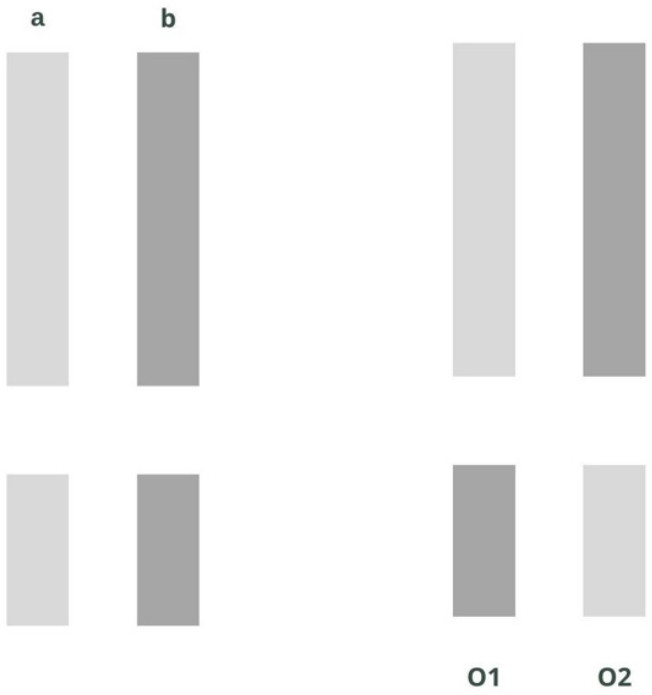

Figure 3.2: *Fully Connected Exchange Crossover. Let a and b be the parental individuals, we will have O_1, O_2 as our crossed offspring.*

From Figure 3.2, on the left side, we have *a* and *b* as parents encodings and graphically one of them is darker than the other. On the right side you see O_1 and O_2 are the outputs

of this crossover operation. In fact, O_1 gets its convolutional part from a and its fully connected part from b which has the darker color. Moreover, O_2 gets its convolutional part from b and its fully connected part from a with lighter color in the figure.

Considering m be the length of encoding, $a = (a_1, a_2, \ldots, a_{m-1}, a_m)$ and $b = (b_1, b_2, \ldots, b_{m-1}, b_m)$ are the parent individuals. Let c show convolutional part and f show fully connected part of these two individuals. Therefore, a_c and b_c are the convolutional parts and the a_f and b_f are the fully connected parts and O_1 and O_2 are the offspring of this crossover then:

$$O_1 = a_c + b_f$$

$$O_2 = b_c + a_f$$

3.5.2 A Hybrid of Point Fusion and Mixed Crossover (HPF&M-R) (Xiang & Zhining, 2019)

In the paper (Xiang & Zhining, 2019), having two individuals, first two random points are picked in the information chain or encoding list. Then the middle part is exchanged. This operation is summarized in the illustration below.

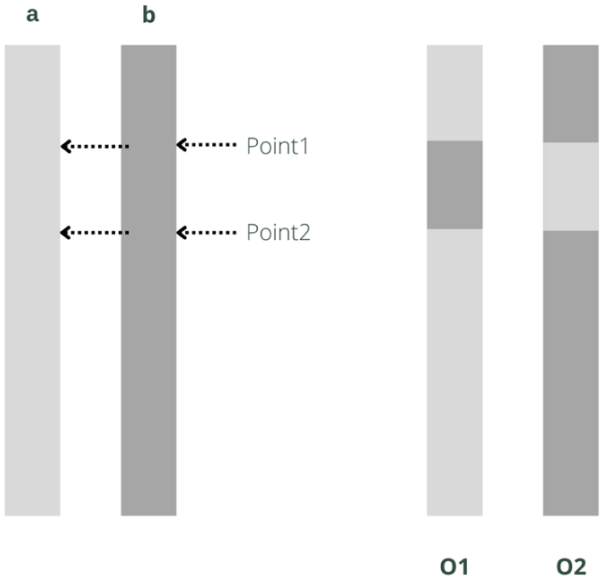

Figure 3.3: *A Hybrid of Point Fusion and Mixed Crossover. Let a and b be the parental individuals, we will have O_1 and O_2 as our crossed offspring.*

As you see in Figure 3.3, on the left side, *a* and *b* are the parents encodings and graphically one of them is darker than the other. On the right side you see O_1 and O_2 are the outputs of this crossover, and it shows that from the *Point1* and *Point2* their colors are different. That means they are exchanged. O_1 gets its encoding from the top until *Point1* from *a* and from *Point1* to *Point2* from parent *b*. Again, from *Point2* until the end of the encoding comes from parent *a*. On the other hand, O_2 obtains its encoding from the top until *Point1* from parent *b*, from *Point1* to *Point2* from parent *a* and finally the rest of its encoding from parent *b*.

The intersection points hyperparameter value is calculated from the following formula. Let *m* be the length of an encoding, $a = (a_1, a_2, \ldots, a_{m-1}, a_m)$ and $b = (b_1, b_2, \ldots, b_{m-1}, b_m)$ be the parent individuals. Considering that r_1 and r_2 are random values between 0 and 1 while *p* and *t* are the random points for crossover selected from the range [2, *t*-2] and [*p*+2, *m*-1] respectively, a' and b' be the p^{th} and t^{th} value, if the crossed offspring at the crossover intersection does not exceed the search space:

$$a' = r_1 * (a_p - b_p) + r_2 * a_p + (1 - r_2) * b_p$$

$$b' = r1 * (a_p - b_p) + r_2 * b_p + (1 - r_2) * a_p$$

$$c' = r_1 * (a_t - b_t) + r_2 * a_t + (1 - r_2) * b_t$$

$$d' = r1 * (a_t - b_t) + r_2 * b_t + (1 - r_2) * a_t$$

otherwise:

$$a' = r_1 * a_p + (1 - r_1) * b_p$$

$$b' = r_1 * b_p + (1 - r_1) * a_p$$

$$c' = r_1 * a_t + (1 - r_1) * b_t$$

$$d' = r1 * a_t + (1 - r_1) * a_t$$

Now the offspring will be as follows:

$$O_1 = (a_1, \ldots, a_{p-1}, a', b_{p+1}, \ldots, b_{t-1}, c', a_{t+1}, \ldots, a_{m-1}, a_m)$$

$$O_2 = (b_1, \ldots, b_{p-1}, b', a_{p+1}, \ldots, a_{t-1}, d', b_{t+1}, \ldots, b_{m-1}, b_m)$$

Note that these calculations and the choice of random r_1 and r_2 are borrowed from the paper (Xiang & Zhining, 2019) as a solution to the problem of using a coefficient which again requires tuning. In Section 3.7, we can find that the elements of the individuals are integers.

3.5.3 A Hybrid of Point Fusion and Mixed Crossover – Collected (HPF&M-C)

Keeping the hybrid of point fusion and mixed crossover operations borrowed from the paper Xiang & Zhining (2019) the as follows,

$$O_1 = (a_1, \ldots, a_{p-1}, a', b_{p+1}, \ldots, b_{t-1}, b', a_{t+1}, \ldots, a_{m-1}, a_m)$$

$$O_2 = (b_1, \ldots, b_{p-1}, a', a_{p+1}, \ldots, a_{t-1}, b', b_{t+1}, \ldots, b_{m-1}, b_m)$$

We add two more simple operators referred to as fully connected exchange crossover, so instead of two offspring O_1 and O_2 we get two more crossovers O_3, and O_4 as follows:

$$O_3 = a_c + b_f$$

$$O_4 = b_c + a_f$$

Note that in O_1 and O_2 the encoding is **one single part**, however, in creating O_3 and O_4 we are separating the fully connected part from the convolutional part in the encoding. The coding implementation will be affected this way.

Figure 3.4 shows that, on the left side, *a* and *b* are the parents encodings and graphically one of them is darker than the other. On the right side you see O_1, O_2, O_3, and O_4 are the outputs of this crossover. For the first two outputs O_1, O_2 we are not separating the convolutional part from fully connected part. O_1, O_2 are obtained through the exchange of parents' encodings from *Point1* and *Point2*. Note that these points are randomly picked and can be anywhere in the encoding. In the illustration we picked *Point1* and *Point2* far

from each other, but they may be close based on randomness. Next, O_3 and O_4 are obtained by first separating the fully connected part from the convolutional part and then exchanging them from the parents *a* and *b*. In fact, O_1 and O_2 are obtained through the Hybrid of Point Fusion and Mixed Crossover while O_3 and O_4 are earned through Fully Connected Exchange Crossover. Therefore, it is a collection of two crossovers. That is why we named this crossover A Hybrid of Point Fusion and Mixed Crossover – Collected.

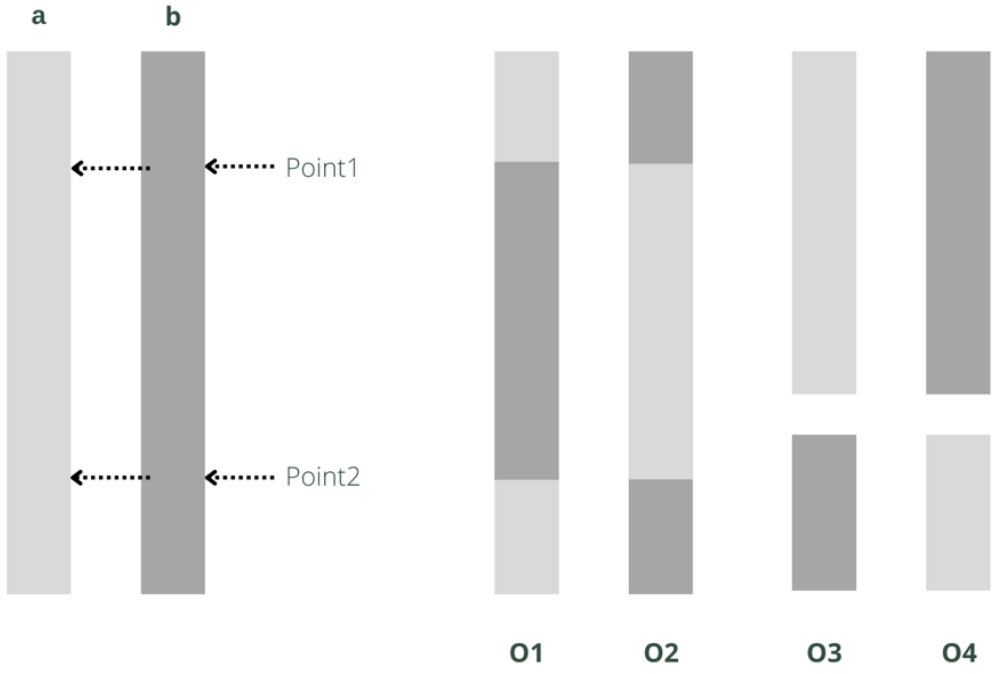

Figure 3.4: *A Hybrid of Point Fusion and Mixed Crossover - Collected. Let a and b be the parental individuals, we will have O_1, O_2, O_3, and O_4 as our crossed offspring.*

3.5.4 A Hybrid of Point Fusion and Mixed Crossover – Replacement (HPF&M-R)

Based on the paper Xiang & Zhining (2019), sometimes the value of crossover intersection *a'*, *b'*, *c'*, and *d'*, obtained through its formula, does not fall in the search space restrictions. In this case, we replace the crossover operation with the fully exchange crossover. If *a'*, *b'*, *c'*, and *d'*, does not exceed the search space, we will have:

$$a' = r_1 * (a_p - b_p) + r_2 * a_p + (1-r_2) * b_p$$

$$b' = r_1 * (a_p - b_p) + r_2 * b_p + (1-r_2) * a_p$$

$$c' = r_1 * (a_t - b_t) + r_2 * a_t + (1 - r_2) * b_t$$

$$d' = r_1 * (a_t - b_t) + r_2 * b_t + (1 - r_2) * a_t$$

$$O_1 = (a_1, \ldots, a_{p-1}, a', b_{p+1}, \ldots, b_{t-1}, c', a_{t+1}, \ldots, a_{m-1}, a_m)$$

$$O_2 = (b_1, \ldots, b_{p-1}, b', a_{p+1}, \ldots, a_{t-1}, d', b_{t+1}, \ldots, b_{m-1}, b_m)$$

Otherwise, we replace O_1 and O_2 with the following formula:

$$O_1 = a_c + b_f$$

$$O_2 = b_c + a_f$$

Where a_c is the convolutional part from parent a and a_f is the fully connected part from parent a. b_c is the convolutional part from parent b and b_f is the fully connected part from parent b. The illustration below shows this crossover. Note that r_1 and r_2 are random numbers in [0,1].

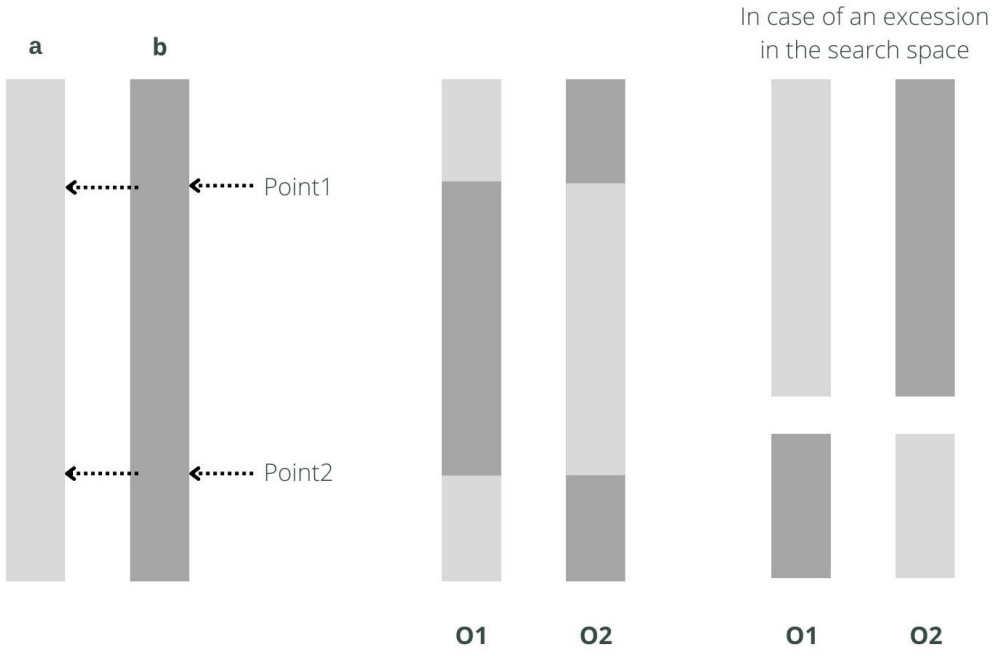

Figure 3.5: *A Hybrid of Point Fusion and Mixed Crossover - replacement. Let a and b be the parental individuals, we will have O_1 and O_2 as our crossed offspring.*

Figure 3.5 displays that, on the left side, *a* and *b* are the parents encodings and graphically one of them is darker than the other. On the right side you see O_1, O_2, O_3, and O_4 are the outputs of this crossover. For the first two outputs O_1, O_2 we are not separating the convolutional part from fully connected part. O_1, O_2 are obtained through the exchange of parents' encodings from *Point1* and *Point2*. Note that these points are randomly picked and can be anywhere in the encoding. In the illustration we picked *Point1* and *Point2* far from each other, but they may be close based on randomness. In case that either of the crossing points obtained from the formulas does not fall into the search space restrictions, we replace O_1, O_2. These two new outputs are obtained by first separating the fully connected part from the convolutional part and then exchanging them from the parents *a* and *b*. In fact, O_1 and O_2 are obtained through the Hybrid of Point Fusion and Mixed Crossover while O_3 and O_4 are earned through Fully Connected Exchange Crossover. Therefore, it is a replacement of two crossovers. That is why we named this crossover A Hybrid of Point Fusion and Mixed Crossover – Replacement.

3.5.5 Fully Connected Exchange + Single Point Fusion Crossover (FCE+SPF) - firstly proposed

There are four offspring out from this proposed operation. First, we separate the convolutional and fully connected parts from the encoding and then exchange the fully connected parts with each other. This creates two offspring. Next, we will have a single point fusion crossover on these two offspring to have the next two offspring.

Figure 3.6 shows that, on the left side, *a* and *b* are the parents encodings and graphically one of them is darker than the other. On the right side you see O_1, O_2, O_3, and O_4 are the outputs of this crossover. First, O_1, O_2 are acquired by first separating the fully connected part from the convolutional part and then exchanging the fully connected parts. Next, O_3 and O_4 are obtained through the exchange of parents' convolutional part from *Point1*. Note that this point is randomly selected and can be anywhere in the convolutional part of the encoding. In fact, O_1 and O_2 are earned through Fully Exchange Crossover while O_3 and O_4 obtained through the single Point Fusion and Mixed Crossover. Therefore, it is a mixture of two crossovers. That is why we named this crossover *Fully Connected Exchange + Single Point Fusion Crossover*.

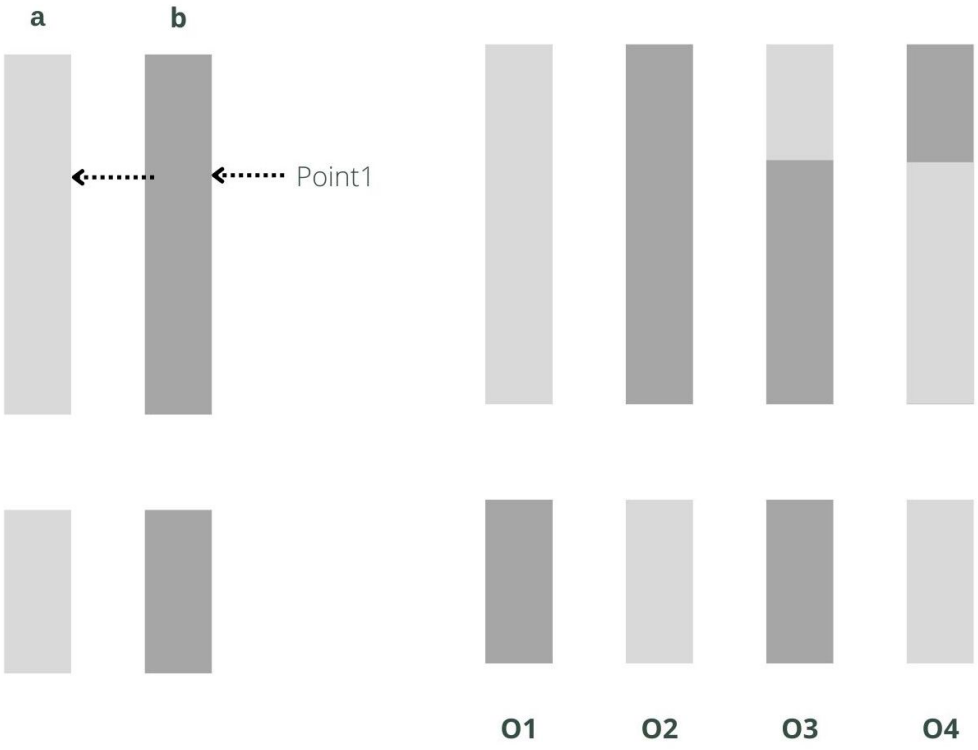

Figure 3.6: *Fully Connected Exchange + Single Point Fusion Crossover. Let a and b be the parental individuals, we will have O_1, O_2, O_3, and O_4 as our crossed offspring.*

In details, O_1 gains its convolutional encoding from *a* and its fully exchange encoding from *b*. O_2 gains its convolutional encoding from *b* and its fully exchange encoding from *a*. O_3 obtains its fully connected encoding from O_1 and its convolutional part from the top until *Point1* from O_1 and from *Point1* to the end of convolutional encoding from convolutional encoding of O_2. O_4 acquires its fully connected encoding from O_2 and its convolutional part from the top until *Point1* from O_2 and from *Point1* to the end of convolutional encoding from convolutional encoding of O_1.

Let *n* be the length of convolutional part in individual encoding. Also, $O_{1c} = (g_1, g_2, ..., g_n)$ and $O_{2c} = (h_1, h_2, ..., h_n)$ be the convolutional parts and O_{1f} and O_{2f} are the fully connected parts, then we will have the following operation on O_{1c} and O_{2c} to obtain the O_3 and O_4:

$$O_1 = a_c + b_f$$

$$O_2 = b_c + a_f$$

$$O_3 = (g_1, \ldots, g_{p-1}, g', h_{p+1}, \ldots, h_n)$$

$$O_4 = (h_1, \ldots, h_{p-1}, h', g_{p+1}, \ldots, g_n)$$

$$g' = r_1 * (g_p - h_p) + r_2 * g_p + (1 - r_2) * h_p$$

$$h' = r_1 * (g_p - h_p) + r_2 * h_p + (1 - r_2) * g_p$$

3.5.6 Fully Connected Exchange + Two Points Fusion Crossover (FCE+TPF) - secondly proposed

This operation is the same as the first proposed however instead of only one point to cross the convolutional parts and crossing the tails, we have two points with crossing the middle part. The formula for O_1 and O_2 remains the same but for O_3 and O_4 will be as follows:

$$O_1 = a_c + b_f$$

$$O_2 = b_c + a_f$$

$$O_3 = (g_1, \ldots, g_{p-1}, g', h_{t+1}, \ldots, h_{t-1}, i', \ldots, g_{p+1}, \ldots, g_n)$$

$$O_4 = (h_1, \ldots, h_{p-1}, h', g_{t+1}, \ldots, g_{t-1}, j', \ldots, h_{p+1}, \ldots, h_n)$$

$$g' = r_1 * (g_p - h_p) + r_2 * g_p + (1 - r_2) * h_p$$

$$h' = r_1 * (g_p - h_p) + r_2 * h_p + (1 - r_2) * g_p$$

$$i' = r_1 * (g_t - h_t) + r_2 * g_t + (1 - r_2) * h_t$$

$$j' = r_1 * (g_t - h_t) + r_2 * h_t + (1 - r_2) * g_t$$

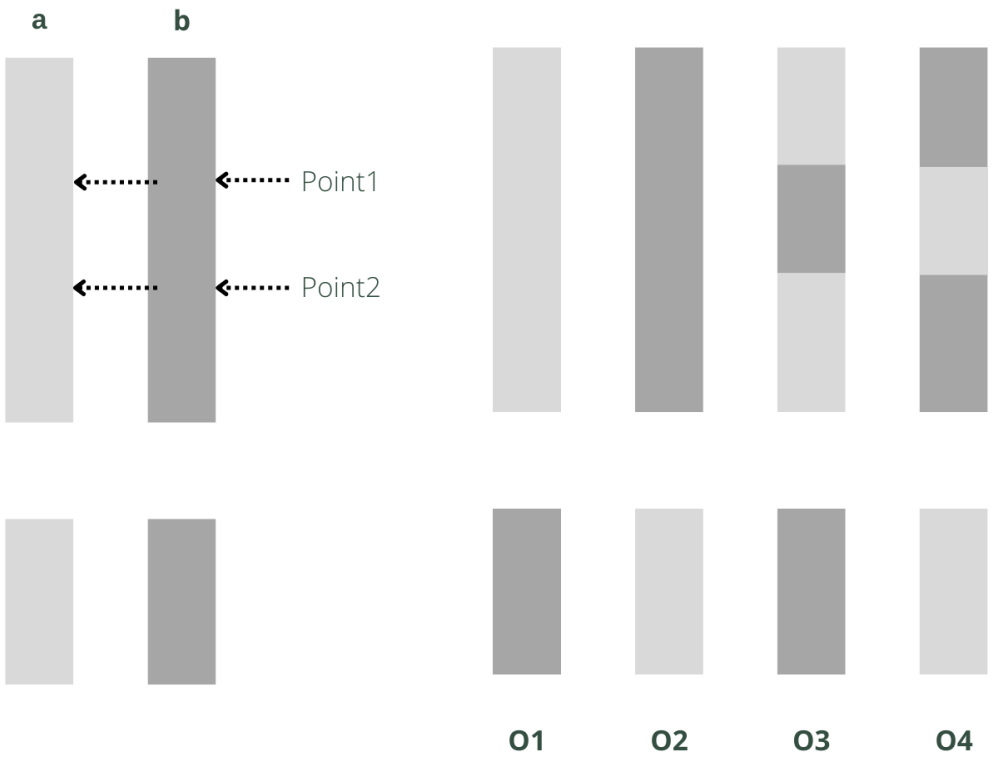

Figure 3.7: *Fully Connected Exchange + Two Points Fusion Crossover. Let a and b be the parental individuals, we will have O_1, O_2, O_3, and O_4 as our crossed offspring.*

Figure 3.7 shows that, on the left side, *a* and *b* are the parents encodings and graphically one of them is darker than the other. On the right side you see O_1, O_2, O_3, and O_4 are the outputs of this crossover. First, O_1, O_2 are acquired by first separating the fully connected part from the convolutional part and then exchanging the fully connected parts. Next, O_3 and O_4 are obtained through the exchange of parents' convolutional part from *Point1* and *Point2*. Note that these points are randomly selected and can be anywhere in the convolutional part of the encoding. In fact, O_1 and O_2 are earned through Fully Exchange Crossover while O_3 and O_4 obtained through the Two-Points Fusion and Mixed Crossover. Therefore, it is a mixture of two crossovers. That is why we named this crossover *Fully Connected Exchange + Two Point Fusion Crossover*.

In details, O_1 gains its convolutional encoding from *a* and its fully exchange encoding from *b*. O_2 gains its convolutional encoding from *b* and its fully exchange encoding from *a*. O_3 obtains its fully connected encoding from O_1 and its convolutional part from the top

until *Point1* from O_1 and gains its encoding from *Point1* to *Point2* from convolutional encoding of O_2 and from *Point2* to the end of convolutional encoding from the convolutional encoding of O_1. O_4 acquires its fully connected encoding from O_2 and its convolutional part from the top until *Point1* from O_2 and obtains its encoding from *Point1* to *Point2* from convolutional encoding of O_1 and from *Point2* to the end of convolutional encoding from the convolutional encoding of O_2.

3.5.7 Fully Connected Exchange + mean value for two important parameters + Single Point Fusion Crossover (FCE+Mean+SPF) - thirdly proposed

This operation is in fact the firstly proposed (FCE+SPF), however, there is a difference in the fully connected exchange operation. When exchanging the fully connected parts, we change the value of the number of epochs and the number of neurons in each layer to be equal to the mean of the corresponding values in the parents. Collected with the single-point crossover we will have the following illustration and formula for all four offspring.

According to Figure 3.8, on the left side, *a* and *b* are the parents encodings and graphically one of them is darker than the other. On the right side you see O_1, O_2, O_3, and O_4 are the outputs of this crossover. First, O_1, O_2 are acquired by first separating the fully connected part from the convolutional part and then exchanging the fully connected parts. Moreover, since we know that number of epochs and number of neurons help model acquire a better performance, we get an average of these two values and replace their original values by this average. Next, O_3 and O_4 are obtained through the exchange of parents' convolutional part from *Point1*. Note that this point is randomly selected and can be anywhere in the convolutional part of the encoding. In fact, O_1 and O_2 are earned through Fully Exchange Crossover while O_3 and O_4 obtained through the single Point Fusion and Mixed Crossover. Therefore, it is a mixture of two crossovers. That is why we named this crossover *Fully Connected Exchange* + mean value for two important parameters + *Single Point Fusion Crossover*.

In details, O_1 gains its convolutional encoding from *a* and its fully exchange encoding from *b* where we also replace the mean value for number of neurons and number of epochs. O_2 gains its convolutional encoding from *b* and its fully exchange encoding from

a where we also replace the mean value for number of neurons and number of epochs. O_3 obtains its fully connected encoding from O_1 and its convolutional part from the top until *Point1* from O_1 and from *Point1* to the end of convolutional encoding from convolutional encoding of O_2. O_4 acquires its fully connected encoding from O_2 and its convolutional part from the top until *Point1* from O_2 and from *Point1* to the end of convolutional encoding from convolutional encoding of O_1.

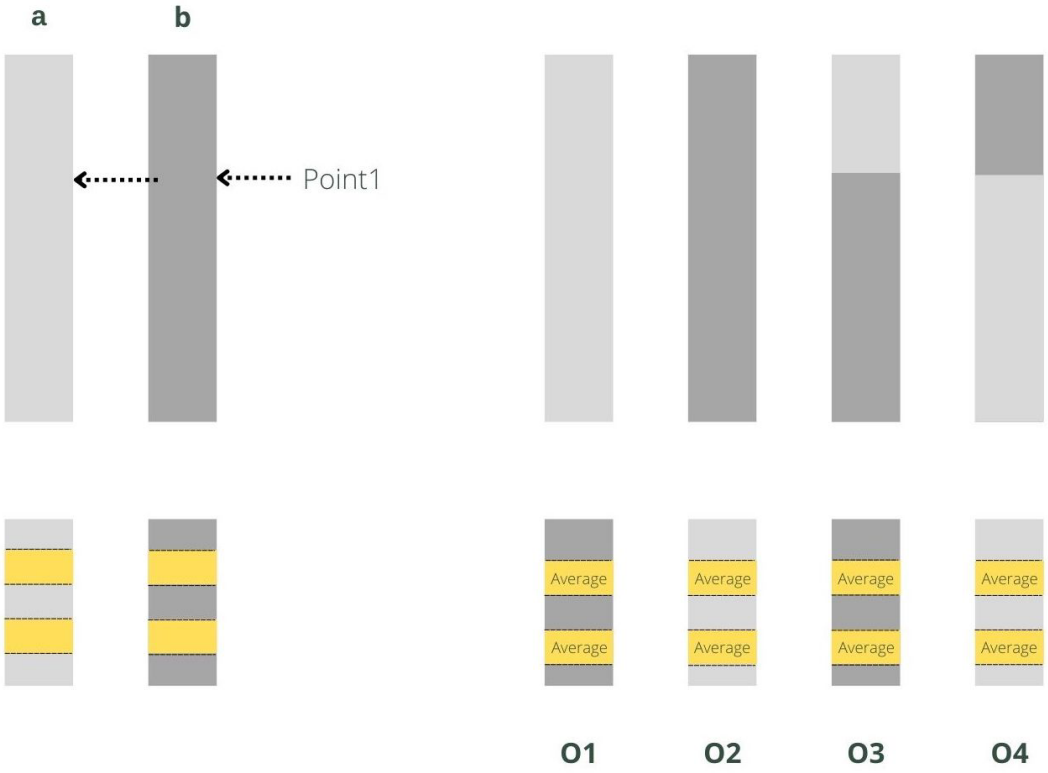

Figure 3.8: *Fully Connected Exchange + mean value for two important parameters + Single Point Fusion Crossover. Let a and b be the parental individuals, we will have O_1, O_2, O_3, and O_4 as our crossed offspring.*

Considering m to be the length of convolutional encoding and n to be the length of fully connected encoding. Let a and b be the parent individuals and the a_c and b_c be the convolutional parts, if the $a_f = (a_{f1}, a_{f2}, \ldots, a_{fn})$ and $b_f = (b_{f1}, b_{f2}, \ldots, b_{fn})$ are the fully connected parts, where r_1 is a random number in [0,1], p is a random point from [2, m-1], we will have O_1 and O_2 two crossed individuals as follows:

$$O_1 = a_c + (b_{f1}, \text{mean } (a_{f2}, b_{f2}), b_{f3}, \text{mean } (a_{f4}, b_{f4}))$$

$$O_2 = b_c + (a_{f1}, \text{mean } (a_{f2}, b_{f2}), a_{f3}, \text{mean } (a_{f4}, b_{f4}))$$

Now from these two crossed individuals we will have one point tale switch in the convolutional parts $O_{1c} = (g_1, g_2, \ldots, g_n)$ and $O_{2c} = (h_1, h_2, \ldots, h_n)$ to obtain O_3 and O_4 as follows:

$$O_3 = (g_1, \ldots, g_{p-1}, g', h_{p+1}, \ldots, h_n)$$

$$O_4 = (h_1, \ldots, h_{p-1}, h', g_{p+1}, \ldots, g_n)$$

Note that g' and h' are the value of the p^{th} in the convolutional part encoding that is selected to cross and is calculating from the formula below:

$$g' = r_1 * (g_p - h_p) + r_2 * g_p + (1 - r_2) * h_p$$

$$h' = r_1 * (g_p - h_p) + r_2 * h_p + (1 - r_2) * g_p$$

3.5.8 Convolutional Fusion & Fully Connected Exchange and Maximization Crossover (CF&FCEM) - fourthly proposed

This set of operations is the same as the third proposed, instead of replacing the mean value of number of neurons and the number of epochs we simply replace both with maximum value between them. In fact, the winner among the proposed algorithms is this fourth one. In human chromosome crossovers, there are two chromosomes from parent 1 and two chromosomes from parent 2, which in this work we associated them with convolutional part and fully connected parts. The difference is that in Meiosis II - Telophase II, both of chromosomes are crossed like point fusion, but here we only cross the convolutional part and for fully connected part we take the maximum value instead of crossing them. We decided to do so because the fully connected integer coding is not long enough to be crossed like an exchange of tails. Plus, it makes more sense for hyperparameters such as number of epochs and number of neurons to get increased for the purpose of adding to the model complexity from a machine learning point of view.

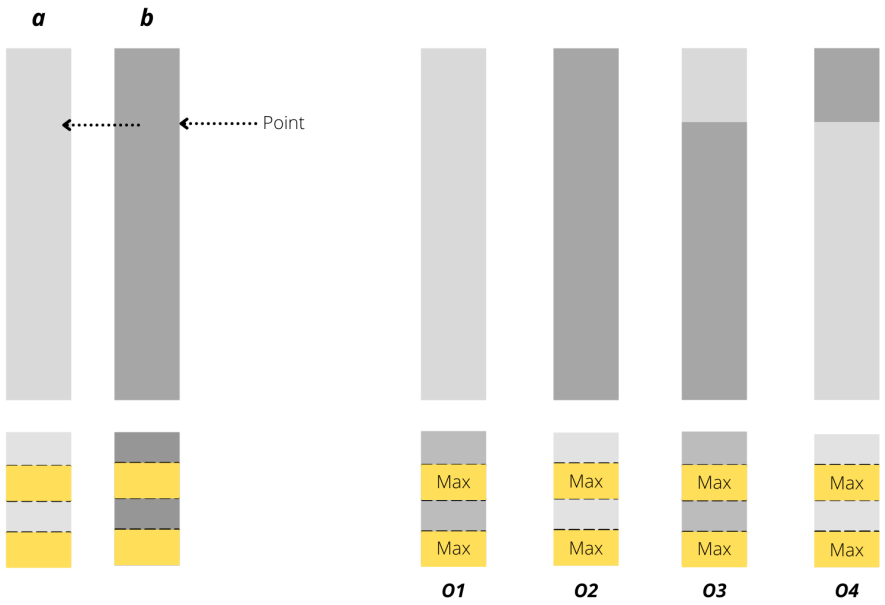

Figure 3.9: *Convolutional Fusion & Fully Connected Exchange and Maximization Crossover. Let a and b be the parental individuals, we will have O_1, O_2, O_3, and O_4 as our crossed offspring.*

Figure 3.9 depicts the operation. According to this figure, on the left side, *a* and *b* are the parents encodings and graphically one of them is darker than the other. On the right side you see O_1, O_2, O_3, and O_4 are the outputs of this crossover. First, O_1, O_2 are acquired by first separating the fully connected part from the convolutional part and then exchanging fully connected parts. Moreover, we know that the number of epochs and the number of neurons help our model acquire a better performance. Therefore, we get an average of these two values and replace their original values by this average. Next, O_3 and O_4 are obtained through the exchange of parents' convolutional part from *Point1*. Note that this point is randomly selected and can be anywhere in the convolutional part of the encoding. In fact, O_1 and O_2 are earned through Fully Exchange Crossover while O_3 and O_4 are obtained through the single Point Fusion and Mixed Crossover. Therefore, it is a mixture of two crossovers. That is why we named this crossover *Fully Connected Exchange* + maximum value for two important parameters + *Single Point Fusion Crossover*.

In details, O_1 gains its convolutional encoding from *a* and its fully exchange encoding from *b*. We also replace the maximum value for number of neurons and number of epochs.

O_2 gains its convolutional encoding from b and its fully exchange encoding from a, we then repeat this same process of replacing maximum values and number of epochs, neurons. O_3 obtains its fully connected encoding from O_1 and its convolutional part from the top until *Point1* from O_1 and from *Point1* to the end of convolutional encoding from convolutional encoding of O_2. O_4 acquires its fully connected encoding from O_2. On the other hand, O_4 's convolutional part, from the top until *Point1*, comes from O_2. And the rest of it comes from convolutional encoding of O_1.

Considering m to be the length of convolutional encoding and n to be the length of fully connected encoding. Let a and b be the parent individuals and the a_c and b_c be the convolutional parts, if the $a_f = (a_{f1}, a_{f2}, \ldots, a_{fn})$ and $b_f = (b_{f1}, b_{f2}, \ldots, b_{fn})$ are the fully connected parts, where r_1 is a random number in [0,1], p is a random point from [2, m-1], we will have O_1 and O_2 two crossed individuals as follows:

$$O_1 = a_c + (b_{f1}, max\ (a_{f2}, b_{f2}), b_{f3}, max\ (a_{f4}, b_{f4}))$$

$$O_2 = b_c + (a_{f1}, max\ (a_{f2}, b_{f2}), a_{f3}, max\ (a_{f4}, b_{f4}))$$

Now from these two crossed individuals we will have one point tale switch in the convolutional parts $O_{1c} = (g_1, g_2, \ldots, g_n)$ and $O_{2c} = (h_1, h_2, \ldots, h_n)$ to obtain O_3 and O_4 as follows:

$$O_3 = (g_1, \ldots, g_{p-1}, g', h_{p+1}, \ldots, h_n)$$

$$O_4 = (h_1, \ldots, h_{p-1}, h', g_{p+1}, \ldots, g_n)$$

Note that g' and h' are the value of the p^{th} in the convolutional part encoding that is selected to cross and is calculating from the formula below:

$$g' = r_1 * (g_p - h_p) + r_2 * g_p + (1 - r_2) * h_p$$

$$h' = r_1 * (g_p - h_p) + r_2 * h_p + (1 - r_2) * g_p$$

Now, let us move on to the mutation operators.

3.6. Mutation Operations

There are five different operations defined as mutation operations as elaborated below.

- The first one is removing a layer from individual encoding from a random point selected. This operation is borrowed from Mattioli et al. (2019) and Sun et al. (2020).

- The second operation is inserting a layer to a random point selected. This layer has hyperparameters that are randomly selected from the hyperparameter ranges. This operation is inspired by Mattioli et al. (2019).

- In the third operation we investigate the existing layers' hyperparameters, and we randomly choose one of them and replace it with a new hyperparameter from the hyperparameter range. This is inspired by Yang & Shami (2020).

- The fourth operation is removing and adding a layer. It means we first select a random point; we remove a layer from that point. We then select another point and insert a new layer to that point.

- Finally, for the fifth operation we decided to have it increase the number of epochs by one. We are aware that from using this operator, we may get better individuals, or we may get worse individuals.

Note that we select one of these five operations for each individual which is an offspring of crossover phase. For each of these operations there is a probability to be selected. This probability is 5%, 50%, 25%, 10%, 10% respectively.

3.7. Experiment Setting

The dataset used is MNIST and we used the torch library for python which has useful and fast functions for neural networks. MNIST has 60,000 images of handwritten digits, that are of size 28*28 (black & white scale so depth is equal to 1) for the training set. There are 10,000 images as the testing set. However, for faster computation we take only 100 images from the training set which is first randomly shuffled and are mutually exclusive.

This way, we may get even extremely low accuracy levels as low as 0% or 1% so when we move forward with the algorithm, we can see the increase in the level of accuracy as we create better individuals. otherwise starting the first generation with individuals of around 90% when we fit the whole train set, we will not be able to see how algorithm performs from 90% to 98% versus from 0% to 98%.

The following table shows the hyperparameter range. For example, any neural network architecture that comes out of this algorithm has at least one convolutional layer and at most 5 convolutional layers. The kernel size will be one of the elements in the set {3, 5, 7, 9, 11}. To store the name of activation function used, {1,2,3,4} corresponds to {relu, selu, softsign, sigmoid} respectively. The choice of these ranges is based on the conventional values used in deep learning as hyperparameters and it can be changed according to the user's wish.

Number of convolutional layers	{1, 2, 3, 4, 5}
Kernel size	{3, 5, 7, 9, 11}
Stride	{1, 2, 3, 4}
Pooling	{2, 3, 4}
Type of activation function	{relu (1), selu (2), softsign (3), sigmoid (4)}
Number of fully connected layers	{0, 1, 2}
Number of neurons per layer	[100, 500]
Dropout	{0.0, 0.2, 0.4, 0.6}

Table 3.1: *Search Space Restrictions*

We have used torch library to implement machine learning in our work. Since this algorithm is fully automated, we may want to fix some of the hyperparameters such as batch size equal to 1, output classes equal to 10, default number of epochs for the first-generation individuals is equal to 1 and it can increase up to a maximum of 50 epochs. There will be no probability for mutation or crossover since we want to give everyone the same chance to cross and mutate in a hope for greater accuracy. The optimization function

is an Adam optimizer with a learning rate of 0.0003. The type of pooling layers is all set to be Max Pooling.

3.8. Design Choice

3.8.1 Toward a Strategy

A. A Two-Generation Crossover Tentative

The idea of the first proposed crossover operation is borrowed from the real crossover in the sex cells of human beings. In every human body two chromosomes, each coming from a grandparent, will be crossed for creating sex cells. Each of these chromosomes has two parts: a longer segment and a shorter segment. We resemble the longer segment to the convolutional layer and the shorter segment to the fully connected layer. That is why we first exchange the fully connected segments exactly like what happens in Meiosis cell division. Then by a randomly selected point we cross the tails of longer segments which is in fact the convolutional part of an individual. This will create four crossed individuals.

At first, we tried to have crossover using the grandparents in the same manner as human biology. This results in a wave of better individuals every other generation, plus a slower convergence compared to using parental crossover. This is justified if we do not want to have children contrasting too far from their parents in each generation. However, in computer science we may want to produce the best individual faster, so we changed to use the parents only as inputs to the crossover operation instead of grandparents.

B. Using Two-Point Crossover

The outcome of the first proposed crossover operation compared to the *hybrid of point fusion and mixed* crossover was not quite different. They were quite the same in terms of diversity and the accuracy level. Therefore, we tried to change one operation in the convolutional part from being one-point fusion crossover to two-point fusion crossover. This only worsened the situation in terms of computation, while the accuracy level remained quite the same.

C. Making Changes in Fully Connected Parts

Then we moved to the third proposed crossover operation. Keeping the first proposed operation, while trying to make changes in the fully connected part. According to Table B, since we observed that an alternation in the convolutional parts in the second proposed algorithm did not make the maximum accuracy any better. In fact, they are almost equal. Thus, there must be something within the fully connected part that may make the algorithm better which we highly likely neglected.

D. Refinements from Strategy C

Therefore, for the fourth crossover operation we took the third proposed operator and made a change in the fully connected part. We then observed that fully connected part is crucial, making the networks perform in a robust manner. In general, the more the neurons, the greater the accuracy. Subsequently we decided to take the maximum of between two values instead of the mean value. This also holds true for the number of epochs. In fact, the results were more robust when changing epochs, and we finally selected our best performing crossover operator among these four. In fact, the fourth one is the winner.

3.8.2 Improvements and Fine Tuning

- One challenge is to see how we can reduce the number of fitness evaluations. One issue specific to tuning hyperparameters for neural networks, whenever the fitness is defined to be the accuracy on the test set, we may find it difficult to train and test on the total dataset. Thusly, we may try to reduce that to a lower amount for the train set, which we practiced within this work. It is suggested to do the same for the custom dataset we may have. Once we have the best hyperparameters we will continue training on the whole dataset and test on the whole test set.

- Another solution is to make sure to evaluate each distinct individual only once during the whole algorithm process. This is achievable by storing the individual encoding with their accuracy level in a global pool. We will then write a function to

help check if the new individual has already been evaluated and exists within the global pool. Again, we made use of this solution as well in this work.

This work is limited to the convolutional neural network architecture design and the fact that some of the hyperparameters are fixed. These include batch size, optimizers, and their learning rate. However, the focus here is to develop an algorithm for tuning any set of hyperparameters with a potential to expand into to more hyperparameter search spaces. Note that this will become computationally expensive so we may want to choose the most affecting hyperparameters based on the type of the neural network we have. In fact, how we tune any set of hyperparameters remains the same when the type of neural network differs. In this work it is limited to CNNs, but it can be further expanded to other NN models.

3.9. Challenges

3.9.1 Feasibility

One issue that arises is the fact that not all combinations for hyperparameters will be oversufficient regarding the size of our image. This issue includes a choice of strides or kernel size. For this case in particular, images in MNIST must maintain a dimension of 28 * 28. To elaborate, the combination of (11, 9, 11) as a kernel size for three convolutional layers is not feasible, as they remain in order, respectively. We need to resolve this issue after both an initialized population and two other stages further along in the algorithm called crossover and mutation. In fact, we will always make sure that for the individual we possess, it remains valid and feasible through utilization of a troubleshooting function. Otherwise, we will have a zero-denominator error.

3.9.2 Fix the Unwanted Offspring Challenge

One issue that arises is that some of the individuals will change in a way that causes them to become invalid. Take the example of crossing a single convolutional layer individual with a three-convolutional layer individual from the second point. The result is this; an individual that has a first layer, second layer is empty, and the third layer is again nonempty. This individual is unwanted, and we want to fix it either by removing the last layer or by discarding the individual.

Another important case of the same nature happens within the mutation phase; after removing a middle layer of a three-layer convolutions individual. You then have an individual with two convolutional layers, but they are sparse and far apart from one another.

In addition to this occurrence in the mutation phase, a second example would transpire when adding or removing layers. When this happens, we need to adjust the number of layers' indicator in the individual encoding. Otherwise, further computations will become problematic because it is implemented based on the number of layers' indicator value. Moreover, when we remove one layer from a one-convolutional layer network, the number of layers will become zero. Due to the randomness involved in all the phases of this algorithm, it will take a long time to encounter these issues along the way and discover their basis for debugging.

To solve abovementioned issues, we defined a function called "fix" so that we can fix these problems right after the crossover. Then once more after the mutation phase, just before we update the pool for the next selection. Other little functions were also implemented to help this fix function work properly.

In this function, we check if the list that includes individual encoding is valid. In case we see abnormality, for instance an empty hidden layer, then we shift the next layers one layer toward the layer that is missed so that it replaces the empty layer. Another example is when a newly picked hyperparameter from the range of search space, is making the individual invalid. This especially happens when we randomly change the kernel size of a layer in the mutation process. In this case we try to pick another kernel size, we do this up to 20 times and if after 20 attempts we still have a non-valid individual, then the function returns the individual without any mutations.

3.9.3 Mimicking Human Body Challenge

During the development of this algorithm, we tried to mimic all the various steps from human evolution, but some of them did not work as intended. For example, in the human body, a crossover occurs between the grandparents but not the parents themselves. First, we implemented the code where individual's crossover was between the grandparents.

Then comparing the results, we realized that it only makes the convergence rate slower, so we then used the parents' architecture to perform the crossover. In nature it would make sense if we do not wish to have higher quality individuals in every consecutive generation. Perhaps we wish to consider a slower convergence but in computer science we might also want to use our resources timely and converge faster.

The follow up chapter fully elaborates the results and interprets them.

Chapter 4
Discussion & Results

4.0. Introduction

In the previous chapter, we described how we developed our new algorithm. This chapter is the analysis of our new algorithm experiments. Computationally speaking, practicing the RAY package of python, which is a general purpose and distributed compute framework, we managed to reduce the computation time by 32% by parallelization. We benefited from a computer with a configuration of 16 GB RAM, Intel core i7 9750H CPU processor with 8 cores. From this resource, we allocated 7 cores and a memory of 7.6 GB of RAM.

We have fixed some of the hyperparameters such as batch size equal to 1, output classes equal to 10, the default number of epochs for the first-generation individuals is equal to 1 and it can increase up to a maximum of 50 epochs. Note that, for the final experiment, we have changed the initial number of epochs from 1 to 5 which only includes the results from Table 4.5. Although the larger the number of epochs, the longer it takes for the algorithm to compute, starting from 5 epochs makes the convergence faster.

On the mutation side, there will be no probability for mutation or crossover since we want to give everyone the same chance to cross and mutate in a hope for greater accuracy. The optimization function is an Adam optimizer with a learning rate of 0.0003. The type of pooling layers is all set to be Max Pooling.

The number of convolutional layers varies among {1, 2, 3, 4, 5} and number of fully connected layers varies among {0, 1, 2}. The value of the kernel size is chosen among the set {3, 5, 7, 9, 11}. The stride and pooling values are respectively chosen from the sets {1, 2, 3, 4} and {2, 3, 4}. The type of activation functions is restricted to ReLU, SeLU, Softsign, and sigmoid. The number of neurons is narrowed to [100, 500] and finally the probability of a dropout is from the set {0.0, 0.2, 0.4, 0.6}.

4.1. Findings

Table 4.1 shows the results of 12 generations of genetic algorithms that are only different in their crossover operators and reports their execution time in minutes, maximum accuracy in the 12th generation solution set, the average accuracy in this set and the standard deviation. This is one run and all six of these experiments begin with the exact same initial population with a size of ten, with an average accuracy of 18%, and the maximum accuracy of 48%. Note that all the experiments in this chapter have been obtained on MNIST dataset for the purpose of the fair comparison. The standard deviation of the last population refers to standard deviation of the individuals in the last population not from population to its previous population but rather only for the last population of that algorithm which means the 12th population.

Experiment	Execution Time (minutes)	Last population's Maximum Accuracy	Last population's Average Accuracy	Last population's Standard Deviation	Last Population
Crossover Proposed 1rst version (FCE+SPF)	94	79%	68%	12.1	75,74,76,60,72,77,71,67,66,72,49,75,74,72,76,79,75,70,72,72,74,17,70,65,68
Crossover Proposed 2nd version FCE+TPF	37	77%	58%	15.82	56,73,36,77,58,55,43,73,32,73,32,74,22,53,68,64,68,71,68,34,61,72,69,60,71
Crossover Proposed 3rd version (FCE+Mean+SPF)	55	77%	60%	17.7	72,72,43,54,60,67,71,73,73,50,43,66,77,40,54,60,71,56,30,75,74,74,73,73,0
Crossover Proposed 4th version (CF&FCEM)	52	***84%***	***71%***	***18.8***	82,63,80,70,75,79,67,78,74,73,81,10,82,81,81,16,78,78,83,81,81,76,84
A Hybrid of Point Fusion and Mixed Crossover (HPF&M)	40	80%	72%	6.2	73,69,73,64,65,78,69,57,79,79,76,80,78,71,66,79,71,71
A Hybrid of Point Fusion and Mixed Crossover Replacement (HPF&M-R)	21	81%	67%	12.8	70,63,72,28,45,65,67,52,80,81,79,76,74,69,70,73,69,78,73

Table 4.1. *A comparison between six crossover operators.*

After comparing these results, although HPF&M-R and CF&FCEM are front runners in maximum accuracy. We therefore perform experiments only on CF&FCEM and HPF&M crossover from this point on. This is due to fair comparison between the crossover proposed in the paper Xiang & Zhining (2019) (or HPF&M) and our fourth proposed crossover operation, CF&FCEM. In fact, HPF&M-R is built upon HPF&M and is a better

version of it, therefore we will use the original operation in paper Xiang & Zhining (2019) for the following experiments.

Moreover, we add a new column for comparison and see which selection mechanism yields more diversity. For establishing a fair judgment, we run each experiment or row three times, then later report the average +- standard deviation of those three result values. Additionally, they all start with the same initial population.

Experiment	Execution Time (minutes)	Last population's Maximum Accuracy	Last population's Average Accuracy	Last population's Standard Deviation	Selection Mechanism (to decide which one to use for the next generation)
Convolutional Fusion & Fully Connected Exchange and Maximization Crossover (CF&FCEM)	51.66 ± 18.8	85.77 ± 0.77	74.81 ± 5.61	13.33 ± 5.03	Ranked
	68 ± 15.57	84.7 ± 0.49	72.34 ± 2.51	13.78 ± 2.12	Random
	64.66 ± 0.47	85.24 ± 2.09	72.60 ± 5.58	14.11± 4.2	Tournament
A Hybrid of Point Fusion and Mixed Crossover (HPF&M)	46.66 ± 6.8	82.11 ± 2.24	75.41 ± 1.34	8.63 ± 6.07	Ranked
	50.66 ± 5.79	84.01 ± 0.49	75.08 ± 0.71	7.15 ± 0.77	Random
	46 ± 22.1	82.98 ± 1.81	71.28 ± 2.09	13.07 ± 4.42	Tournament

Table 4.2. *Two crossover operators versus three selection mechanisms*

Based on Table 4.2, focusing on the standard deviation in the last population, we see that tournament selection mechanism has the best result. Additionally, this table shows the average results of the three experiments in all cells. Take the example of (85.24 ± 2.09), this means we have run three experiments and from the three, the average is 85.24 with a margin of 2.09. This makes the results promising. Focusing on the maximum accuracy among the set of solutions (converged population), we compare both our CF&FCEM crossover and HPF&M crossover against each other. Results indicate that CF&FCEM crossover retains a higher maximum accuracy in each of the selection mechanism categories.

Comparing the standard deviations of the final set of solutions, taking ranked selection mechanism, CF&FCEM crossover provides a much higher degree of standard deviations

to that of HPF&M crossover counterpart. This holds true for Random and Tournament selections as well.

Looking at these three selection mechanisms prioritized for the next generation, we note that the tournament selection establishes greater impacts on the standard deviation, solely within HPF&M crossover. This proves that tournament selection plays an outstanding role in diversification.

For the next set of experiments, we will repeat two algorithms for thirty times total, to ensure that these results are reliable. Both algorithms have the same mutation operators mentioned in Section 3.6 and selection mechanism is set to tournament. They only differ in their crossover operators. Each algorithm iterates for five populations where the first one is the same for both. Table 4.3 presents the statistics on the maximum accuracy for each of the 30 runs written in population row.

Descriptive Statistics:	**Symbol**	CF&FCEM Crossover with our mutation operations from Section 2.6	HPF&M Crossover with our mutation operations from Section 2.6
Minimum	min	71.74	60.09
Maximum	max	83.13	82.68
Range	R	11.39	22.59
Size	n	30	30
Mean	μ	78.61	75.32
Median	x	78.59	76.09
Population		71.74, 73.99, 74.57, 76.08, 76.48, 76.52, 76.97, 77.3, 77.39, 77.48, 77.67, 77.84, 77.88, 78.12, 78.35, 78.83, 78.86, 79.19, 79.48, 80.02, 80.24, 80.58, 80.93, 81.01, 81.25, 81.28, 81.51, 81.84, 81.88, 83.13	66.76, 73.54, 76.16, 72.47, 70.57, 70.01, 77.84, 81.24, 78.5, 81.0, 81.69, 75.11, 65.52, 64.78, 76.97, 72.06, 79.18, 80.18, 80.34, 77.03, 76.03, 82.68, 82.3, 75.42, 81.96, 60.09, 71.27, 74.66, 72.81, 81.62
Standard Deviation	σ	2.54	5.74

Table 4.3. *A comparison between two crossover operators, CF&FCEM and HPF&M.*

Figure 4.1 presents the boxplots of the abovementioned populations. As mentioned before this population refers to the maximum accuracy of the individuals of the fifth generation in this experiment. Each of CF&FCEM and HPF&M has run thirty times. These boxplots highlight that using the CF&FCEM crossover leads to more reliable set of solutions as we see among thirty runs. This is because the minimum pertaining to the maximum accuracy values in CF&FCEM are higher than that of HPF&M crossover. If we compare the minimum accuracy, we realize that CF&FCEM crossover reaches an accuracy that is 10 percent higher. The following maximum accuracy is less than 1 percent higher, with a mean accuracy more than 3 percent higher, and the median following suit which is two percent higher than HPF&M. Standard deviation of CF&FCEM crossover is less than half of HPF&M's.

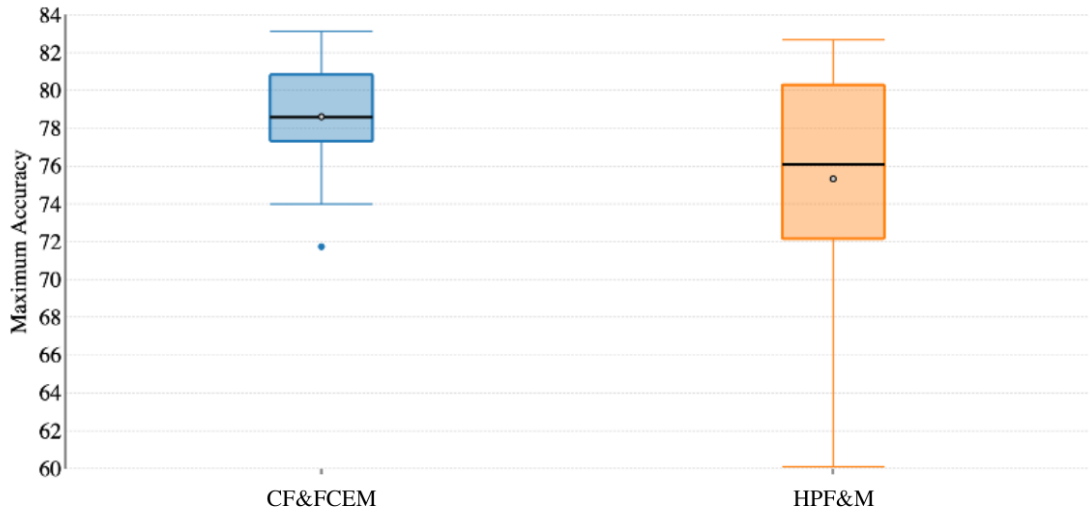

Figure 4.1. *Boxplot of maximum accuracy distribution, a comparison between CF&FCEM crossover and HPF&M crossover both with our mutation operations.*

The next set of experiments corresponds to the same experiment as above, except that we begin with a population that is not fixed across every experiment or run. We allow randomness to determine the initialization of the population for all case experiments instead. Note that the size of the first population is set to 10.

Moreover, a new column has been added to the table where we can see our results when we change the mutation operators. First column is the combination of CF&FCEM

crossover with the mutation operations from Section 3.6. Second column is the combination of the HPF&M crossover with similar mutation operations from Section 3.6. The third column displays results of the HPF&M crossover along with its own mutation operators mentioned by Xiang & Zhining (2019).

Descriptive Statistics:	**Symbol**	CF&FCEM Crossover with our mutation operations from Section 3.6	HPF&M Crossover with our mutation operations from Section 3.6	HPF&M Crossover with Xiang & Zhining (2019)'s original mutation operations
Minimum	min	70.7	54.29	49.7
Maximum	max	85.01	82.91	73.9
Range	R	14.31	28.62	24.2
Size	n	30	30	30
Mean	μ	79.07	74.82	67.43
Median	x	79.85	77.40	68.01
Population		71.74, 73.99, 74.57, 76.08, 76.48, 76.52, 76.97, 77.3, 77.39, 77.48, 77.67, 77.84, 77.88, 78.12, 78.35, 78.83, 78.86, 79.19, 79.48, 80.02, 80.24, 80.58, 80.93, 81.01, 81.25, 81.28, 81.51, 81.84, 81.88, 83.13	66.76, 73.54, 76.16, 72.47, 70.57, 70.01, 77.84, 81.24, 78.5, 81.0, 81.69, 75.11, 65.52, 64.78, 76.97, 72.06, 79.18, 80.18, 80.34, 77.03, 76.03, 82.68, 82.3, 75.42, 81.96, 60.09, 71.27, 74.66, 72.81, 81.62	49.7, 60.89, 61.84, 62.39, 63.85, 64.03, 65.18, 65.55, 65.78, 65.85, 66.54, 67.66, 67.73, 67.87, 68.15, 68.34, 68.38, 69.12, 69.45, 69.86, 69.99, 70.06, 70.54, 72.02, 72.05, 72.15, 73.51, 73.87, 73.9, 66.68
Standard Deviation	σ	3.62	6.98	4.75

Table 4.4. *A comparison between two crossover operators and two mutation operators.*

To better understand Table 4.4, we introduce the following boxplots that manifests a comparison of the distributions for maximum accuracies in these two experiments: first, the CF&FCEM crossover with our own proposed mutation operations. Second, HPF&M crossover operation with the paper Xiang & Zhining (2019)'s mutation operations.

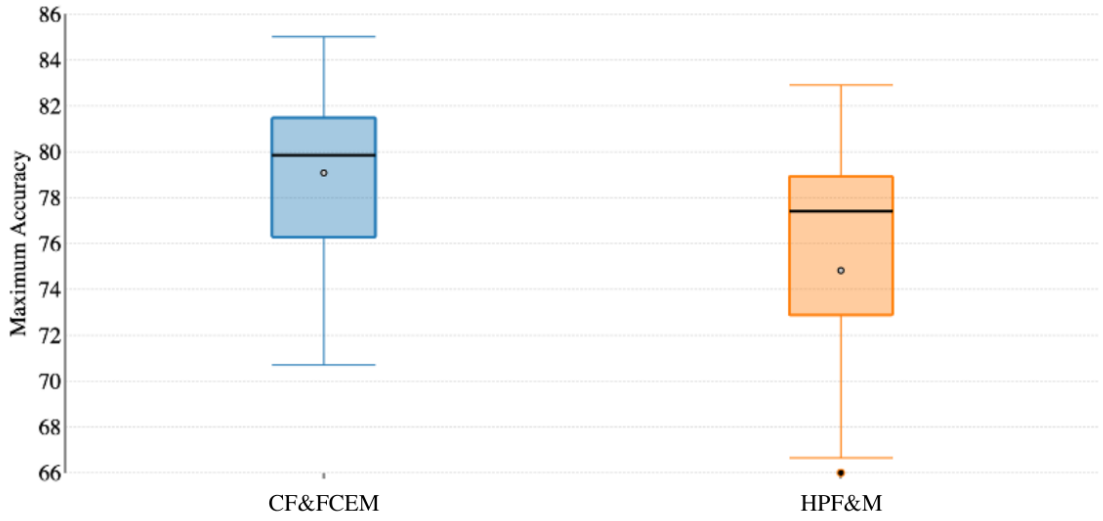

Figure 4.2. *Boxplot of maximum accuracy distribution, a comparison between CF&FCEM crossover and our mutation operations versus HPF&M crossover with Xiang & Zhining (2019)'s original mutation operations.*

Figure 4.2 suggests that using CF&FCEM crossover results in steadier and trustworthy set of solutions. As we see among thirty runs, the minimum value of all the maximum accuracy values remains higher than in comparison with HPF&M crossover.

From the beginning there were many experiments in this research aiming for finding the best performing crossover. The very first experiments (Table A. in the appendix) showed that HPF&M-Collected has the lowest maximum accuracy among all with 47%. After that, the fully exchange crossover had the lowest accuracy level with 60%. HPF&M crossover was the front runner with 81%. After that it was the first proposed crossover with 74%. This set of experiments guided us to the fact that we need to invest more on the first proposed crossover to compete with HPF&M crossover. Therefore, we eliminated crossover operations with the least effective accuracy values from our further experiments. Those of which were FCE crossover and HPF&M-Collected crossover.

Heading to the next set of experiments, we had the results in Table B which you can locate within the appendix. We then introduced the third crossover proposed along with a fourth version. The fourth version surpassed HPF&M crossover by a small amount in accuracy value. Both versions surpassed HPF&M crossover with a larger amount for standard

deviation. Between these last two, the fourth crossover was selected for the crossover phase of our final algorithm, the name of which being algo-new.

Table 4.5 shows the last population individuals. As you see individuals are sorted based on their accuracy level. This accuracy is obtained after four generations of genetic algorithm. The encodings consist of two parts: the first bracket is there to describe the hyperparameters of convolutional part and the next bracket corresponds to the fully connected part hyperparameters.

Individual	Encoding	Accuracy (%)
1	[3, 3, 1, 2, 4, 5, 1, 7, 2, 0, 0, 0, 0] [1, 100, 0.2, 7, 98.88]	98.88
2	[3, 7, 1, 2, 4, 3, 1, 9, 1, 0, 0, 0, 0] [1, 100, 0.2, 7, 98.81]	98.81
3	[2, 9, 2, 2, 3, 7, 1, 0, 0, 0, 0, 0, 0] [1, 100, 0.2, 6, 98.70]	98.70
4	[3, 5, 1, 2, 4, 7, 1, 9, 1, 0, 0, 0, 0] [1, 100, 0.2, 7, 98.70]	98.70
5	[3, 5, 1, 2, 4, 3, 1, 9, 2, 0, 0, 0, 0] [2, 500, 0.2, 6, 98.63]	98.63
6	[3, 5, 1, 2, 4, 3, 1, 9, 1, 0, 0, 0, 0] [2, 100, 0.0, 7, 98.54]	98.54
7	[3, 7, 1, 2, 4, 3, 1, 7, 2, 0, 0, 0, 0] [1, 100, 0.2, 7, 98.54]	98.54
8	[2, 7, 1, 2, 3, 7, 2, 0, 0, 0, 0, 0, 0] [1, 500, 0.2, 7, 98.53]	98.53
9	[3, 7, 1, 2, 4, 5, 1, 9, 1, 0, 0, 0, 0] [1, 500, 0.2, 6, 98.47]	98.47
10	[3, 7, 1, 2, 4, 5, 1, 9, 1, 0, 0, 0, 0] [1, 500, 0.2, 6, 98.47]	98.47
11	[3, 3, 1, 2, 3, 3, 1, 7, 2, 0, 0, 0, 0] [1, 100, 0.2, 7, 98.46]	98.46
12	[2, 5, 2, 2, 3, 9, 2, 0, 0, 0, 0, 0, 0] [1, 500, 0.0, 7, 98.39]	98.39
13	[3, 5, 2, 2, 1, 7, 1, 5, 1, 0, 0, 0, 0] [2, 100, 0.0, 7, 98.32]	98.32

14	[3, 3, 1, 2, 4, 5, 1, 9, 1, 0, 0, 0, 0] [2, 100, 0.2, 6, 98.25]	98.25
15	[2, 5, 2, 2, 3, 7, 2, 0, 0, 0, 0, 0, 0] [1, 100, 0.2, 7, 98.25]	98.25
16	[2, 3, 1, 2, 4, 9, 2, 0, 0, 0, 0, 0, 0] [2, 500, 0.2, 6, 98.00]	98.00
17	[2, 3, 1, 2, 4, 9, 1, 0, 0, 0, 0, 0, 0] [2, 100, 0.2, 7, 97.73]	97.73
18	[3, 3, 1, 2, 4, 11, 2, 5, 1, 0, 0, 0, 0] [2, 100, 0.2, 6, 96.53]	96.53
19	[3, 3, 1, 2, 4, 3, 3, 4, 2, 0, 0, 0, 0] [2, 500, 0.2, 6, 95.99]	95.99
20	[2, 3, 4, 2, 3, 5, 2, 0, 0, 0, 0, 0, 0] [1, 100, 0.0, 7, 93.84]	93.84

Table 4.5 *Final Algorithm on its Evolution to Generation Number 4*

In Table 4.5, the convergence of our algorithm is obtained in the fourth generation using MNIST's full training set with the size of 60,000 images. Comparing this table to that of Xiang & Zhining (2019), which you can find within the appendix table *C*, we see that their algorithm converges to 98.81% with standard deviation of 0.0002396 in the thirtieth generation while our algorithm converges to 98.88% with standard deviation of 1.2200917 in the fourth generation.

4.2. Discussion

4.2.1 Tables

- Table 4.2 shows that standard deviation results of ranked selection and random selection mechanisms are also notable. Observing these two mechanisms proves that operations defined in the CF&FCEM crossover were the root cause behind larger standard deviation. Regardless of any selection mechanism utilized in its implementation. This is because of the crossover operation brought forward within our proposed algorithm. Doing so allows for a fully connected part to be treated differently from the convolutional part and even includes its own separate operation formula. Versus in the HPF&M crossover operation, convolutional and fully connected parts of the gene representation are both treated as a single part. Therefore, our new algorithm has a better enhanced crossover operation

specifically tuned for CNNs that have a fully connected part. This is simply because the more operation you place on values within the gene representations, the higher degree of variance is subsequently generated among the offspring individuals. As well the degree to which they remain disparate regarding their parents.

- In Table 4.2, when comparing between the standard deviation column and the average column, we realize the higher value of standard deviation, the lower average accuracy. We conclude a trade-off remains between a higher average accuracy and obtaining diversity.

- The results of Table 4.3 are extracted from thirty runs each using a tournament selection mechanism combined with a random crossover selection for five generations each. Based on this table we can claim that the CF&FCEM crossover is more promising. Meaning, we can guarantee a better accuracy using these new functions, regardless of the high degree of randomness involved in distinct stages of the algorithm. Note that to have a fair comparison, we started all sixty experiments in Table 4.3 from one unique pool, as in the first generation.

- Unlike Table 4.3, in Table 4.4 we started all experiments from a randomly generated initial population and the results for our algo-new proceeded much further after consideration of the randomness factor. Up until now we always kept the same mutation operations in the experiments, however in this table we present the results of a new experiment, in the last column, that is the exact implementation of crossover and mutation operations from the paper Xiang & Zhining (2019). Comparing these results with the other two results, the minimum accuracy is lower, the maximum accuracy is lower. This shows that our mutation operation surpasses operations from the paper Xiang & Zhining (2019) in performance. One reason is that in our algorithm all the individuals can mute. Second reason is that there are five different mutation operations in our algorithm instead of one. Third reason is that between these mutation operations, there is higher probability of usage for adding layer and increasing the epochs by one. This probability allocation comes from the fact that we want to increase the capacity of our model.

- Looking into Table 4.5, the reason behind this faster convergence comparing algo-new to the algorithm belong to Xiang & Zhining (2019) is of two; first, Xiang & Zhining (2019) uses multi perceptron layer neural network, however, we are using CNNs which has proven to work better with images. Second, we have increased the number of epochs in the first generation from 1 in all the past experiments to 5 which helps boosting the accuracy of the model. However, in Xiang & Zhining (2019), the number of epochs is not a hyperparameter, it is a parameter of 20. The reason we did not use 20 epochs is that it requires a huge resource and that based on experience and knowledge we have from deep learning, we know that using a high number of epochs without early stopping results in model to start overfitting.

- The results of the third and fourth tables both promise reliability following an enhanced degree of minimum accuracy. Additionally, using the CF&FCEM crossover, we may retain similar accuracy levels for individuals within the last generation.

4.2.2 Gene Representations

An example of a final set of solutions or the last generation individuals' gene representations is in the appendix. Looking into these encodings of each individual we note two observations:

First, sometimes we have two accuracy values that are not so much different from one another, take two individuals between 77% and 77%. However, we see that the hyper parameters corresponding to them are disparate. In fact, a five-layer CNN may retain the same accuracy of a three-layer CNN. The same phenomenon as observed for one percent accuracy value difference, take a pair of individuals with 76% or 75%. In these cases, we see that the hyperparameters contrast in either the convolutional part or in the fully connected part.

Second, there are individuals with 10% or even 7% accuracy within the last generations, which results in greater diversity. In the case of a 7% accuracy individual, we know that we only have a softmax as an output layer and no fully connected layer before it. In addition, we have just one convolutional layer. Therefore, we expect a low accuracy due

to the diminished capacity of the model. However, for a 9% we do not expect similar low accuracy with three convolutions and one fully connected layer.

1. One reason is that a single change of one hyper parameter will not make the highest accuracy, rather, a combination plays the key role. Meaning that a combination makes a perfect impact.

2. Another reason is that, only increasing the model capacity may not always be a promising idea. In this regard, we must pay attention to the dataset size as well since we can be easily overfit by only increasing this capacity of the model. Of which is sensitive to the size of the dataset. Due to our small dataset, increasing the number of layers within this area does not help to provide a better solution.

3. Additionally increasing the number of epochs is an idea worthy of consideration, but only up to a certain point. An optimal solution does not necessarily include an optimal single hyperparameter, in fact, the combination of hyperparameters next to one another is preferred. When we tune hyperparameters manually, we may miss this combination, simply because we cannot always attempt all possible combinations. In a directed search, this it is within reason.

4. As we compare an individual of 10% accuracy with other three-convolutional individuals that have higher accuracy, for instance 75%, we see that the kernel size is larger in 10% individual compared to the other. Therefore, we conclude that the smaller the kernel size, the better features extracted. This means only adding additional layers may not help our architectural performance, we must continue tuning kernel size along with added layers.

Collectively, there are rules on hyper-parameter tuning in deep learning that make the network more accurate. For example, the deeper and the wider the network (increasing capacity by adding layers or adding neurones per layer) the better yields for network accuracy. Another example is the more you optimize training through back propagation, in fact the higher the number of epochs, the better the accuracy. These points remain true until we pass an optimal point, which leads to a lower accuracy. Our algorithm brings the

individual with highest accuracy and a set of individuals with the highest diversity, includes the following phases:

First, randomly initialization of first generation.

Second, a tournament selection for league selection.

Third, random selection of two individuals from the league set to pass to the crossover operations. This means one individual may be picked more than once.

Fourth, applying *Convolutional Fusion & Fully Connected Exchange and Maximization* Crossover to the selected individuals.

Fifth, mutation of all individuals with one of the five mutation operations defined.

Sixth, creating neural networks for the mutated individuals and calculating their accuracy.

Seventh, going to the first step and repeating.

Chapter 5
Conclusion

There are many examples of algorithms that manage the hyperparameter tuning for neural networks in such a way that results in a final architecture with the best accuracy. Some are fully automatic, some need more human intervention. Among meta-heuristic algorithms, genetic algorithms retain the ability to be parallelized and find a larger set of solution space. Some of the genetic algorithms focus on selection mechanisms, some focus on operations to control the speed of convergence or to acquire the best possible fitness.

This research has introduced a semi-automatic genetic algorithm to find a set of solutions with the highest variances in which there is an architecture with the highest accuracy. Once we set some of the hyperparameters, mostly the ones related to optimization process in the neural networks, our proposed genetic algorithm works automatically and gives us a final architecture. This architecture design is for convolutional neural networks against many other works with only MPL architectures and the set of hyperparameters is set according to the dataset. The following algorithm is the fourth attempt after careful comparison between the implementation of the four algorithms proposed. Including the state-of-the-art algorithm in the paper by Xiang & Zhining (2019). One drawback in the existing genetic algorithm implementations is the lack of diversity within generations that we addressed throughout this research. The goal obtained in this work is to introduce new operations for crossover that mimics the human natural evolutional operator. This operator captures the high diversity as well as maintaining high accuracy level.

Our results show that this algorithm is designed to converge toward 85% accuracy within an average of 61 minutes trained on the limited 100 images from MNIST and tested on the MNIST test set. Additionally, Xiang & Zhining (2019) HPT algorithm converges to 98.81% with standard deviation of 0.00 in the thirtieth generation while our algorithm converges to 98.88% with standard deviation of 1.22 in the fourth generation. This comparison is based on the same dataset MNIST.

These results continue to matter as we deem higher accuracies and difference between the individuals within various generations. Determining these results are imperative because tuning hyperparameters is a challenging task. It comes with an excessive cost of computation and if not execute in an optimized manner, one may extend many resources in addition to passing by the optimal solution. Acknowledging this fact, we have managed to achieve a directed automatic search result that promises a range of solutions, in which we find the best candidate.

Limitations

This work is an attempt to tune hyperparameters of convolutional neural networks, and exclusively an algorithm in both crossover and mutation operations developed for convolutional neural networks. Have no guarantee on how this algorithm would perform if we attempted further tuning the hyperparameters for graph neural networks, or recurrent neural networks and any other NNs.

Aside from standard deviation, one can try to define new functions as a measurement of diversity between individuals of a generation.

This study has shown how crossover operation affects diversity. We have no guarantee how mutation operations dictate this parameter and its accuracy level. Solely after consideration of the same crossover operation in two genetic algorithms.

Recommendation

The purpose of this research is to find an automated algorithm that can find the best hyperparameters for a CNN that is widely used in computer vision. In this way, the users who have no knowledge in hyperparameter tuning will benefit from the automation. In the future, we will bring to bear two ways for developing this work. First, the hyperparameter range can be defined with different value sets based on the current architecture in both crossover and mutation operations. This may help the exploitation but can also lead to infeasibility. Second, a set of selection mechanisms can be tested to decide which two individuals are going to crossover with each other. This is important since we can define it randomly as we did in this research, where an individual has the chance to

cross more than once. Or we define it in-a-row, meaning we take two individuals that are stored next to each other in the data structure used for crossing. This helps to control the chance of an individual achieving a crossover only once. Or it can be the same for all, in case of in a row selection. In addition, with the climate change, future research will focus on how to reduce the computation in the fitness evaluation.

Bibliography

A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States, 2012, pp. 1106–1114.

A. Lentzas, C. Nalmpantis and D. Vrakas, "Hyperparameter Tuning using Quantum Genetic Algorithms," 2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI), 2019, pp. 1412-1416, doi: 10.1109/ICTAI.2019.00199.

B. L. Miller, D. E. Goldberg et al., “Genetic algorithms, tournament selection, and the effects of noise,” Complex Systems, vol. 9, no. 3, pp. 193–212, 1995.

B. Wang, Y. Sun, B. Xue, and M. Zhang, "Evolving Deep Convolutional Neural Networks by Variable-Length Particle Swarm Optimization for Image Classification," 2018 IEEE Congress on Evolutionary Computation (CEC), 2018, pp. 1-8, doi: 10.1109/CEC.2018.8477735.

Barret Zoph, & Quoc V. Le. (2017). *Neural Architecture Search with Reinforcement Learning.*

Basha, SH Shabbeer, Shiv Ram Dubey, Viswanath Pulabaigari, and Snehasis Mukherjee. "Impact of fully connected layers on performance of convolutional neural networks for image classification." *Neurocomputing* 378 (2020): 112-119.

C. M. Anderson-Cook, “Practical genetic algorithms,” p. 1099, 2005.

C. Paduraru, M. Paduraru and A. Stefanescu, "Automated game testing using computer vision methods," 2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW), 2021, pp. 65-72, doi: 10.1109/ASEW52652.2021.00024.

Chattopadhyay, A., Hassanzadeh, P. & Pasha, S. Predicting clustered weather patterns: A test case for applications of convolutional neural networks to spatio-temporal climate data. Sci Rep 10, 1317 (2020). https://doi.org/10.1038/s41598-020-57897-9

D. E. Goldberg and J. H. Holland, “Genetic algorithms and machine learning,” Machine Learning, vol. 3, no. 2, pp. 95–99, 1988.

D. M. Hawkins, “The problem of overfitting,” Journal of Chemical Information and Computer Sciences, vol. 44, no. 1, pp. 1–12, 2004.

Deepti Gupta, & Shabina Ghafir. An Overview of methods maintaining Diversity in Genetic Algorithms.

Dhelim, S., Aung, N., Bouras, M. et al. A survey on personality-aware recommendation systems. Artif Intell Rev 55, 2409–2454 (2022). https://doi.org/10.1007/s10462-021-10063-7

Eberhart, Russell, and James Kennedy. "A new optimizer using particle swarm theory." MHS'95. Proceedings of the sixth international symposium on micro machine and human science. IEEE, 1995.

E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin, “Large-scale evolution of image classifiers,” in Proceedings of Machine Learning Research, Sydney, Australia, 2017, pp. 2902–2911.

G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, “Densely connected convolutional networks,” in Proceedings of 2017 IEEE Con- ference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 2017, pp. 2261–2269.

G. Zhang, Y. Gu, L. Hu, and W. Jin, “A novel genetic algorithm and its application to digital filter design,” in Proceedings of 2003 IEEE Intelligent Transportation Systems, vol. 2. IEEE, 2003, pp. 1600–1605.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

Gutmann, HM. A Radial Basis Function Method for Global Optimization. Journal of Global Optimization 19, 201–227 (2001). https://doi.org/10.1023/A:1011255519438

H. L. Liu, F. Gu, Y.-m. Cheung, S. Xie, and J. Zhang, “On solving WCDMA network planning using iterative power control scheme and evolutionary multiobjective algorithm [application notes],” IEEE Com- putational Intelligence Magazine, vol. 9, no. 1, pp. 44–52, 2014.

H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, “Hierarchical representations for efficient architecture search,” in Pro- ceedings of 2018 Machine Learning Research, Stockholm, Sweden, 2018.

He, Xin, Kaiyong Zhao, and Xiaowen Chu. "AutoML: A survey of the state-of-the-art." *Knowledge-Based Systems* 212 (2021): 106622.

J. Redmon, S. K. Divvala, R. B. Girshick, A. Farhadi, You only look once: Unified, real-time object detection, in 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, IEEE Computer Society, 2016, pp. 779–788. doi:10.1109/CVPR.2016. 91. URL https://doi.org/10.1109/CVPR.2016.91

James S Bergstra et al. “Algorithms for hyper-parameter optimization.” In: Advances in Neural Information Processing Systems. 2011, pp. 2546–2554.

K. Nag and N. R. Pal, “A multiobjective genetic programming-based ensemble for simultaneous feature selection and classification,” IEEE Transactions on Cybernetics, vol. 46, no. 2, pp. 499–510, 2015.

K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” in Lecture Notes in Computer Science. Amsterdam, the Netherlands: Springer, 2016, pp. 630–645.

K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.

Kamilaris, A., & Prenafeta-Boldú, F. (2018). A review of the use of convolutional neural networks in agriculture. The Journal of Agricultural Science, 156(3), 312-322. doi:10.1017/S0021859618000436

L. Davis, Handbook of genetic algorithms. Bosa Roca, USA: Taylor & Francis Inc, 1991.

L. Xie and A. Yuille, "Genetic CNN," 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 1388-1397, doi: 10.1109/ICCV.2017.154.

LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. Nature 521, 436–444 (2015). https://doi.org/10.1038/nature14539

Li, YH., Chang, CC., Su, GD. et al. Coverless image steganography using morphed face recognition based on convolutional neural network. J Wireless Com Network 2022, 28 (2022). https://doi.org/10.1186/s13638-022-02107-5

Lin, Tsung-Yu, et al. "MistNet: Measuring historical bird migration in the US using archived weather radar data and convolutional neural networks." Methods in Ecology and Evolution 10.11 (2019): 1908-1922.

Lorenzo, Pablo Ribalta, et al. "Hyper-parameter selection in deep neural networks using parallel particle swarm optimization." Proceedings of the genetic and evolutionary computation conference companion. 2017.

Loshchilov, Ilya, and Frank Hutter. "CMA-ES for hyperparameter optimization of deep neural networks." arXiv preprint arXiv:1604.07269 (2016).

M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *in IEEE Transactions on Systems, Man, and Cybernetics, vol. 24, no. 4, pp. 656-667*, April 1994, doi: 10.1109/21.286385.

M. Srinivas and L. M. Patnaik, “Genetic algorithms: A survey,” Com- puter, vol. 27, no. 6, pp. 17–26, 1994.

Mattioli, Fernando & Caetano, Daniel & Cardoso, Alexandre & Naves, Eduardo & Lamounier Jr, Edgard. (2019). An Experiment on the Use of Genetic Algorithms for Topology Selection in Deep Learning. *Journal of Electrical and Computer Engineering*. 2019. 1-12. 10.1155/2019/3217542.

Miikkulainen, Risto, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju et al. "Evolving deep neural networks." In *Artificial intelligence in the age of neural networks and brain computing*, pp. 293-312. Academic Press, 2019.

Milano, Nicola, and Stefano Nolfi. "Scaling Up Cartesian Genetic Programming through Preferential Selection of Larger Solutions." *arXiv preprint arXiv:1810.09485* (2018).

Miller, J. F., & Harding, S. L. (2008). Cartesian Genetic Programming. *Proceedings of the 10th Annual Conference Companion on Genetic and Evolutionary Computation, 2701–2726*. https://doi.org/10.1145/1388969.1389075

Ming Li, Wenqiang Du, Fuzhong Nian, "An Adaptive Particle Swarm Optimization Algorithm Based on Directed Weighted Complex Network", Mathematical Problems in Engineering, vol. 2014, Article ID 434972, 7 pages, 2014. https://doi.org/10.1155/2014/434972

Mittal, S., Ranjan, A., Roy, B., Rathore, V. (2022). Mus-Emo: An Automated Facial Emotion-Based Music Recommendation System Using Convolutional Neural Network. In: Dhar, S., Mukhopadhyay, S.C., Sur, S.N., Liu, CM. (eds) Advances in Communication, Devices and Networking. Lecture Notes in Electrical Engineering, vol 776. Springer, Singapore. https://doi.org/10.1007/978-981-16-2911-2_29

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhut- dinov, “Dropout: a simple way to prevent neural networks from over- fitting,” The Journal of Machine Learning Research, vol. 15, no. 1, pp. 1929–1958, 2014.

Nalçakan, Yağız, and Tolga Ensari. "Decision of neural networks hyperparameters with a population-based algorithm." International conference on machine learning, optimization, and data science. Springer, Cham, 2018.

Nelder, John A. and Roger Mead. “A Simplex Method for Function Minimization.” Comput. J. 7 (1965): 308-313.

Olof, Skogby Steinholtz. “A Comparative Study of Black-box Optimization Algorithms for Tuning of Hyper-parameters in Deep Neural Networks.” (2018).

R. K. Srivastava, K. Greff, and J. Schmidhuber, “Training very deep networks,” in Advances in Neural Information Processing Systems, Montral, Canada, 2015, pp. 2377–2385.

Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019). Regularized Evolution for Image Classifier Architecture Search. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*(01), 4780-4789.

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

Rosentreter, Johannes & Hagensieker, Ron & Waske, Björn. (2020). Towards large-scale mapping of local climate zones using multitemporal Sentinel 2 data and convolutional neural networks. Remote Sensing of Environment. 237. 111472. 10.1016/j.rse.2019.111472.

S. Malik and S. Wadhwa, “Preventing premature convergence in genetic algorithm using dgca and elitist technique,” international Journal of Advanced Research in Computer Science and Software Engineering, vol. 4, no. 6, 2014.

Seungyeon Lee, & Dohyun Kim (2022). Deep learning based recommender system using cross convolutional filters. Information Sciences, 592, 112-122.

Silva, Nilton & Braz, Fabricio & de Campos, Teofilo. (2018). Document type classification for Brazil's supreme court using a Convolutional Neural Network. 7-11. 10.5769/C2018001.

Snoek, J., Larochelle, H., & Adams, R. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. In Advances in Neural Information Processing Systems. Curran Associates, Inc.

Suganuma, M., Shirakawa, S. and Nagao, T., 2017, July. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the genetic and evolutionary computation conference* (pp. 497-504).

Sun, Y., Xue, B., Zhang, M., & Yen, G. (2019). Evolving Deep Convolutional Neural Networks for Image Classification. IEEE Transactions on Evolutionary Computation, 24(2), 394-407.

Sun, Yanan & Xue, Bing & Zhang, Mengjie & Yen, Gary & Lv, Jiancheng. (2020). Automatically Designing CNN Architectures Using the Genetic Algorithm for Image Classification. *IEEE Transactions on Cybernetics.* PP. 1-15. 10.1109/TCYB.2020.2983860.

T. Back, Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. England, UK: Oxford university press, 1996.

The lazy programmer. (2016). *Convolutional Neural Networks in Python: Master Data Science and Machine Learning with Modern Deep Learning in Python, Theano, and TensorFlow (Machine Learning in Python).* Kindle Store.

Turner, R., Eriksson, D., McCourt, M., Kiili, J., Laaksonen, E., Xu, Z., & Guyon, I. (2021). Bayesian Optimization is Superior to Random Search for Machine Learning Hyperparameter Tuning: Analysis of the Black-Box Optimization Challenge 2020. In H. J. Escalante & K. Hofmann (Eds.), Proceedings of the NeurIPS 2020 Competition and

Demonstration Track (Vol. 133, pp. 3–26). PMLR. https://proceedings.mlr.press/v133/turner21a.html

Yang, Li & Shami, Abdallah. (2020). On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice.

Young, Steven R., Derek C. Rose, Thomas P. Karnowski, Seung-Hwan Lim, and Robert M. Patton. "Optimizing deep learning hyper-parameters through an evolutionary algorithm." In *Proceedings of the workshop on machine learning in high-performance computing environments*, pp. 1-5. 2015.

Yu Guo, Jian-Yu Li, & Zhi-Hui Zhan (2021). Efficient Hyperparameter Optimization for Convolution Neural Networks in Deep Learning: A Distributed Particle Swarm Optimization Approach. Cybernetics and Systems, 52(1), 36-57.

Yuan Li, Guangjian Ning, & Haoxi Zhang (2022). Research on Position Recommendation System Based on Convolutional Neural Network. Journal of Physics: Conference Series, 2171(1), 012065.

X. Li, M. He, H. Li, and H. Shen, "A Combined Loss-Based Multiscale Fully Convolutional Network for High-Resolution Remote Sensing Image Change Detection," in IEEE Geoscience and Remote Sensing Letters, vol. 19, pp. 1-5, 2022, Art no. 8017505, doi: 10.1109/LGRS.2021.3098774.

Xiang, W., & Zhining, Y. (2019). Neural Network Hyperparameter Tuning Based on Improved Genetic Algorithm. Proceedings of the 2019 8th International Conference on Computing and Pattern Recognition, 17–24. https://doi.org/10.1145/3373509.3373554

Xie, Lingxi, and Alan Yuille. "Genetic cnn." *Proceedings of the IEEE international conference on computer vision*. 2017.

Appendix

Experiment	Max Accuracy Acquired among the individuals in the last population
FCE crossover	60%
Crossover Proposed 1st version	74%
Crossover Proposed 2nd version	73%
HPF&M crossover	81%
HPF&M- Collected crossover	47%
HPF&M-Replacement crossover	69%

Table A

Experiment	Max Accuracy Acquired among the individuals in the last population	Standard Deviation
Crossover Proposed 1st version	79%	12.1
Crossover Proposed 2nd version	77%	15.82
Crossover Proposed 3rd version	77%	17.7
Crossover Proposed 4th version (CF&FCEM)	84%	18.8
HPF&M crossover	80%	6.2
HPF&M-Replacement crossover	81%	12.8

Table B

Individual	Encoding	Accuracy
1	3-2-970-788-866-633-0-0-0-156	98.81
2	3-2-970-788-866-818-0-0-0-156	98.78
3	3-2-970-788-866-682-0-0-0-156	98.76
4	3-2-970-788-866-688-0-0-0-156	98.75
5	3-2-970-802-422-387-0-0-0-135	98.75
6	3-2-970-795-121-688-0-0-0-144	98.75
7	3-2-970-802-121-688-0-0-0-144	98.74
8	3-2-970-788-866-645-0-0-0-156	98.74
9	3-2-970-798-619-0-0-0-0-135	98.73
10	3-2-970-936-293-427-0-0-0-156	98.73
11	3-2-970-802-417-298-0-0-0-139	98.73
12	5-2-970-802-417-298-0-0-0-121	98.72

Table C: *Generation number thirty by Xiang & Zhining (2019)*

The output of the last generation:

The Testing Set Accuracy of the Network is: 77 %

[1 7 1 2 1 0 0 0 0 0 0 0 0]

[2 400 0.0 4 0]

Finished Training

The testing set accuracy of the network is: 64 %

[4 7 1 2 3 5 1 3 1 7 1 0 0]

[2 400 0.0 4 0]

Finished Training

The testing set accuracy of the network is: 83 %

[3 7 1 2 3 7 1 7 1 0 0 0 0]

[2 400 0.0 4 0]

Finished Training

The testing set accuracy of the network is: 10 %

[3 7 1 2 3 4 1 7 1 0 0 0 0]

[1 100 0.0 5 0]

Finished Training

The testing set accuracy of the network is: 76 %

[4 3 1 2 3 9 1 4 1 6 1 0 0]

[2 100 0.0 5 0]

Finished Training

The testing set accuracy of the network is: 75 %

[3 11 1 2 3 3 1 7 1 0 0 0 0]

[1 400 0.0 4 0]

Finished Training

The testing set accuracy of the network is: 7 %

[1 11 2 3 1 0 0 0 0 0 0 0 0]

[0 0 0 1 0]

Finished Training

The testing set accuracy of the network is: 77 %

[3 9 1 2 3 5 1 7 1 0 0 0 0]

[2 400 0.0 4 0]

Finished Training

The testing set accuracy of the network is: 78 %

[3 9 1 2 3 5 1 7 1 0 0 0 0]

[2 100 0.0 5 0]

Finished Training

The testing set accuracy of the network is: 78 %

[2 7 1 2 3 7 1 0 0 0 0 0 0]

[1 100 0.0 5 0]

Yes, we found it here!

The Testing Set Accuracy of the Network is: 77 %

[1 7 1 2 1 0 0 0 0 0 0 0 0]

[2 400 0.0 4 0]

Yes, we found it here!

The Testing Set Accuracy of the Network is: 77 %

[3 11 1 2 3 7 1 7 1 0 0 0 0]

[1 400 0.0 4 0]

Finished Training

The testing set accuracy of the network is: 72 %

[4 7 1 2 3 3 1 3 1 7 1 0 0]

[1 100 0.0 4 0]

Finished Training

The testing set accuracy of the network is: 75 %

[4 5 1 2 3 7 1 3 1 7 1 0 0]

[1 100 0.0 5 0]

Finished Training

The testing set accuracy of the network is: 81 %

[4 7 1 2 3 9 1 3 1 6 1 0 0]

[1 100 0.0 5 0]

Finished Training

The testing set accuracy of the network is: 67 %

[2 7 2 2 3 9 1 0 0 0 0 0 0]

[2 400 0.0 4 0]

Finished Training

The testing set accuracy of the network is: 52 %

[3 11 2 2 1 7 1 3 1 0 0 0 0]

[2 400 0.0 4 0]

Finished Training

The testing set accuracy of the network is: 66 %

[2 5 2 2 3 7 1 0 0 0 0 0 0]

[2 400 0.0 4 0]

Finished Training

The testing set accuracy of the network is: 69 %

[2 3 2 2 1 7 1 0 0 0 0 0 0]

[2 400 0.0 4 0]

Finished Training

The testing set accuracy of the network is: 42 %

[3 11 3 2 3 7 1 3 1 0 0 0 0]

[1 400 0.0 4 0]

Yes, we found it here!

The Testing Set Accuracy of the Network is: 77 %

[3 5 1 2 3 3 1 7 1 0 0 0 0]

[2 100 0.0 4 0]

Yes, we found it here!

The Testing Set Accuracy of the Network is: 82 %

[3 9 1 2 3 5 1 7 1 0 0 0 0]

[2 100 0.0 4 0]

Finished Training

The testing set accuracy of the network is: 77 %

[3 3 1 2 3 11 1 7 1 0 0 0 0]

[1 500 0.0 4 0]

Finished Training

The testing set accuracy of the network is: 81 %

[3 7 1 2 3 5 1 7 1 0 0 0 0]

[1 500 0.0 4 0]

Finished Training

The testing set accuracy of the network is: 75 %

[3 9 1 2 3 3 1 9 1 0 0 0 0]

[2 100 0.0 5 0]

Finished Training

The testing set accuracy of the network is: 79 %

[2 5 1 2 3 7 1 0 0 0 0 0 0]

[2 100 0.0 5 0]

Finished Training

The testing set accuracy of the network is: 81 %

[3 9 1 2 3 5 1 7 1 0 0 0 0]

[2 500 0.0 4 0]

Finished Training

The testing set accuracy of the network is: 73 %

[4 9 1 2 3 5 1 3 1 7 1 0 0]

[2 400 0.0 4 0]

Finished Training

The testing set accuracy of the network is: 74 %

[3 11 1 2 3 5 1 7 1 0 0 0 0]

[1 400 0.0 4 0]

29

[77.69 64.78 83.07 10.32 76.34 75.8 7.04 77.37 78.01 78.78 77.69 77.01 72.03 75.13
81.95 67.13 52.99 66.46 69.86 42.17 77.78 82.16 77.4 81.78 75.32 79.68 81.7 73.42
74.48]

===Standard Deviation of the generation===18.99586526179671

===Maximum of the generation===83.07

===Average of the generation===69.49448275862069

---00:35:46.25---