VIETNAM NATIONAL UNIVERSITY
HANOI UNIVERSITY OF SCIENCE

Computer and Information Science (Honors Program)

Introduction to Artificial Intelligence

Handwritten Digit Recognition

Students: Tran Nam Khanh - 2001559
Tran Hoang Anh - 20001528
Nguyen Thi Phuong Hoa - 20001549
Ngo Phuong Anh - 20001523

Hanoi, November 24th, 2022

VIETNAM NATIONAL UNIVERSITY
HANOI UNIVERSITY OF SCIENCE

Computer and Information Science (Honors Program)

Introduction to Artificial Intelligence

Handwritten Digit Recognition

Supervisor: Dr. Nguyen Hai Vinh

Students: Tran Nam Khanh - 2001559
Tran Hoang Anh - 20001528
Nguyen Thi Phuong Hoa - 20001549
Ngo Phuong Anh - 20001523

Hanoi, November 24th, 2022

Contents

List of Figures

ABSTRACT

The 4.0 revolution brought a technological explosion, many high-level technologies took the throne and became ubiquitous in most data entered into computers through the keyboard. But in some cases, using handwriting is still more appropriate, such as taking notes in class. In that situation, the handwriting recognition problem is studied to improve communication between humans and machines.

The handwriting recognition problem has been researched and developed for the past 40 years and has also achieved many remarkable results. Handwriting recognition (HWR), also known as Handwritten Text Recognition (HTR), is the ability of a computer to receive and interpret intelligible handwritten input from sources such as paper documents, photographs, touch screens, and other devices.

In recent years, deep learning has been extensively used in both supervised and unsupervised learning problems. Among the deep learning models, CNN has outperformed all others for the object recognition tasks. Although CNN achieves exceptional accuracy, still a huge number of iterations and chances of getting stuck in local optima make it computationally expensive to train. Genetic Algorithm is a metaheuristic approach inspired by the theory of natural selection and has been used for solving both bounded and unbounded optimization problems with large success. To handle these issues, we have developed a hybrid deep learning model using the Genetic Algorithm and L-BFGS method for training CNN. To test our model, we have taken the MNIST handwritten numeral dataset. Our results show that GA-assisted CNN produces better results than non-GA-assisted CNN. This study concludes that the evolutionary technique can be used to train CNN more efficiently.

Chapter 1

INTRODUCTION

We all know that HWR is considered as a *classification*, more specific, an *image classification* problem. The aforementioned learning problem is called classification because its results are discrete class numbers. When solving an HWR, in this study, we will propose there are two subproblems we need to take care of. First, we need to solve *learning problem.* After teaching the computer how to classify and tag a given handwritten, we will think about *optimization problem.*

In recent years, deep convolutional neural networks (CNNs) have achieved considerable success in many image analysis tasks, such as image classification, object detection, semantic segmentation, etc. CNN can learn low/mid/high-level features from a specific benchmark through alternating convolutional layers and pooling layers. Compared to traditional feed-forward neural networks with similarly-sized layers, CNNs have much fewer connections and parameters. In the literature, with the development of CNNs, many successful architectures have been proposed, such as VGGNet, GoogleNet, ResNet, etc.

Moreover, these smartly designed networks further reduced the large quantities of connections and parameters, and achieved superior learning performance in many real-world applications. However, despite the great successes enjoyed by these various CNN architectures, most of them are developed manually by human experts, which is a time-consuming and error-prone process. As different applications or tasks may require unique CNN architecture, it is thus desirable to develop intelligent approaches which can automate the process of CNN architecture engineering.

In the literature, to automate the architecture configuration of CNN, there is a growing interest in Neural Architecture Search (NAS), and many optimizations and learning approaches. Besides the optimization methods discussed before, the evolutionary algorithm (EA) also plays an important role to explore the search space for NAS, since it does not require much domain knowledge of the problem when compared to existing algorithms, such as reinforcement learning and gradient-based methods. The early approach which employed EA for NAS is the neuro-evolution of topologies, which only evolved simplified neural network topologies together with weights. With the improvement of hardware, EA can now produce complex architectures for NAS.

Beside the algorithms above, in this paper, we propose a memetic algorithm (MA) based neural network MANet, which integrates local search into the global search of EvoCNN.

Using MA will help us solve some classic EvoCNN problem which is:

1. The architectures found in EvoCNN are all chain-structured networks, which may cause the degradation problem mentioned
2. There may exist some fully connected layers in the final solutions, which are prone to overfitting due to the large quantities of parameters
3. The performance estimation is still computationally expensive

The rest of this paper is organized as follows. Chapter 2 gives a brief introduction to Convolutional Neural Networks, Neural Architecture Search, and Evolutionary Optimization. Next, the proposed algorithm is presented in Chapter 3. Then, the experimental study and the obtained results are shown in Chapter 4. Lastly, chapter 5 gives the concluding remarks of this work

Chapter 2

BACKGROUND

2.1 Convolutional Neural Network

A convolutional neural network (CNN or ConvNet) is a subset of machine learning. It is one of the various types of artificial neural networks which are used for different applications and data types. A CNN is a kind of network architecture for deep learning algorithms and is specifically used for image recognition and tasks that involve the processing of pixel data.

A deep learning CNN consists of three layers: a convolutional layer, a pooling layer, and a fully connected (FC) layer.

From the convolutional layer to the FC layer, the complexity of the CNN increases. It is this increasing complexity that allows the CNN to successively identify larger portions and more complex features of an image until it finally identifies the object in its entirety

- ***Convolutional layer***: The majority of computations happen in the convolutional layer, which is the core building block of a CNN. A second convolutional layer can follow the initial convolutional layer. The process of convolution involves a kernel or filter inside this layer moving across the receptive fields of the image, checking if a feature is present in the image. Over multiple iterations, the kernel sweeps over the entire image. After each iteration, a dot product is calculated between the input pixels and the filter. The final output from the series of dots is known as a feature map or convolved feature. Ultimately, the image is converted into numerical values in this layer, which allows the CNN to interpret the image and extract relevant patterns from it

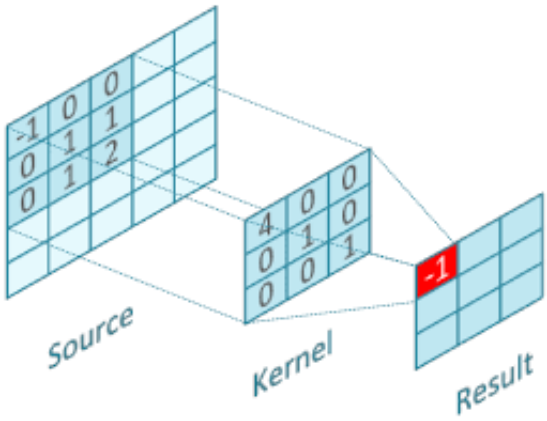

Figure 2.1: The convolution step

- ***Pooling layer***: Like the convolutional layer, the pooling layer also sweeps a kernel or filter across the input image. But unlike the convolutional layer, the pooling layer reduces the number of parameters in the input and also results in some information loss. On the positive side, this layer reduces complexity and improves the efficiency of the CNN.

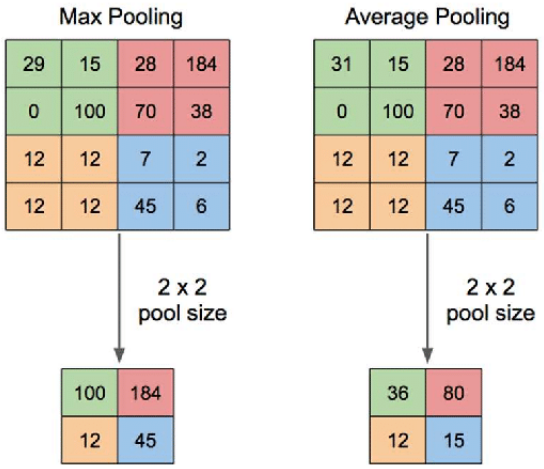

Figure 2.2: Two types of pooling: Max pooling and average pooling

- ***Fully connected layer***: The FC layer is where image classification happens in the CNN based on the features extracted in the previous layers. Here, fully connected means that all the inputs or nodes from one layer are connected to every activation unit or node of the next layer.

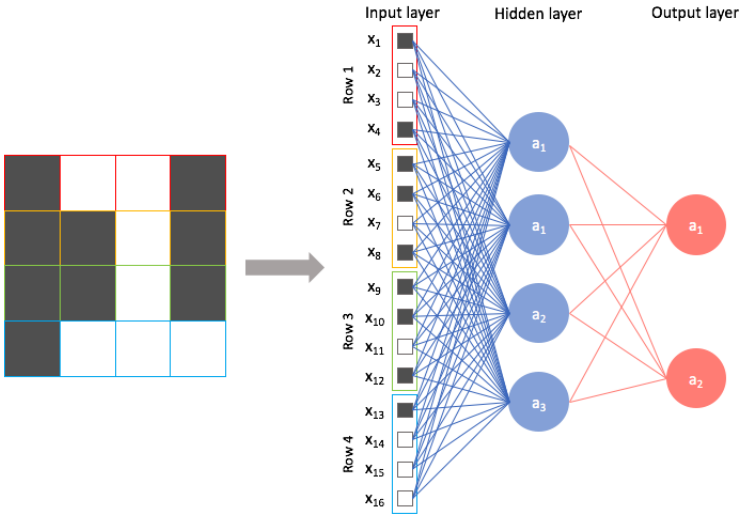

Figure 2.3: Fully connected layer

2.2 Neural Architecture Search

Neural architecture search is the task of automatically finding one or more architectures for a neural network that will yield models with good results (low losses), relatively quickly, for a given dataset.

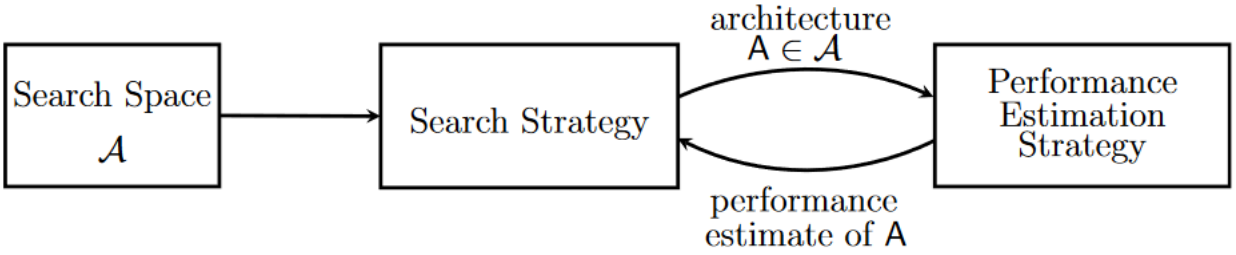

Figure 2.4: Illustrated summary of Neural Architecture Search methods [1]

There are 3 core components that should be noted. They will be explained in detail in Chapter 3:

- ***The search space***: Search all architectures or different submodules of neural networks that can be created. Thereby, we have two popular strategies:
  - **Macro-search**: A strategy for designing the entire neural network architecture.
  - **Micro-search**: A strategy to connect modules or blocks together to form the final neural network architecture. This strategy will be much more complicated than Macro-search.

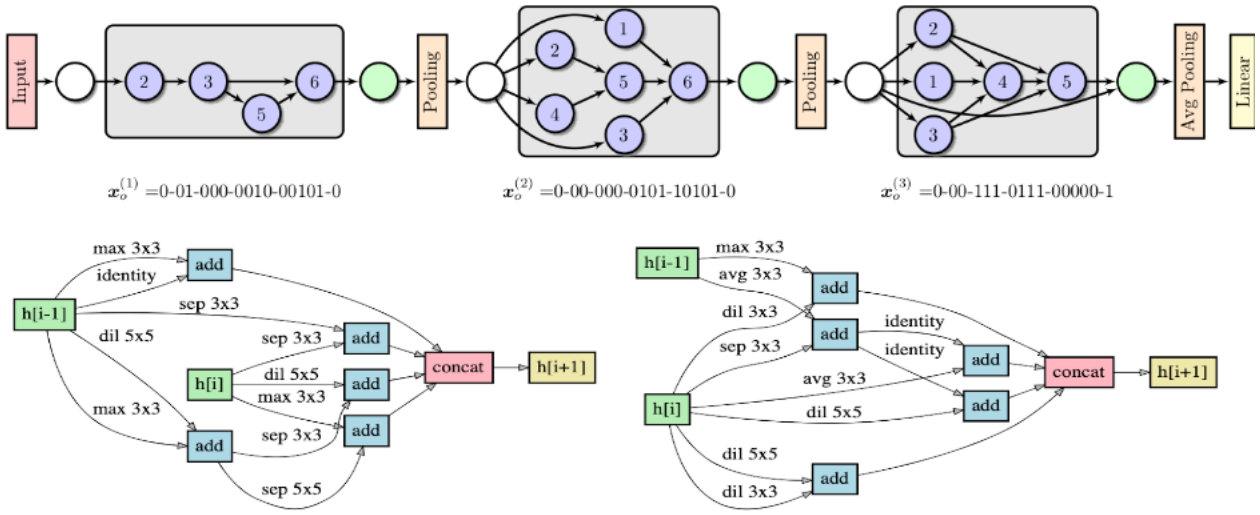

Figure 2.5: Simulating Macro-search (top image) and Micro-search (bottom image) [1]

- ***The search strategy***: Methods to explore the defined search space, to find well-performing architecture quickly
- ***The performance evaluation technique***: A method to measure the effectiveness of the newly created architecture.

2.3 Evolutionary Optimization

2.3.1 What are Evolutionary Algorithms?

Evolutionary Algorithms are Population based Metaheuristics. They are high-level procedures that control the searching through strategies. These strategies are naturally inspired by interesting names: Genetic Algorithm (GA), Memetic Algorithm (MA), and Ant Colony Optimization (ACO),... They have three main characteristics:

- ***Population-based***: EAs maintain a group of solutions, called a *population*, to optimize or learn the problem in a parallel way. The population is a basic principle of the evolutionary process.
- ***Fitness-oriented***: Every solution in a population is called an *individual*. Every individual has its representation, called its *code*, and performance evaluation, called its *fitness value*. EAs prefer fitter individuals, which is the foundation of the optimization and convergence of the algorithms.
- ***Variation-driven***: Individuals will undergo a number of various operations to mimic the natural behavior, which is fundamental to search the solution space.

2.3.2 Memetic Algorithm

A Memetic Algorithm is an extension of the traditional genetic algorithm. It uses a local search technique to reduce the likelihood of premature convergence.

1 build an initial population Pop
2 **while** *the stopping criteria is not met* **do**
3 selection: draw randomly two parents P_1 end P_2
4 reproduction: apply a crossover to get children
5 improvement: apply local search to the children with a given probability
6 mutation: randomly perturb children solutions (with small probability)
7 replacement: select some solutions in the population and replace them by the children
8 return the best solution

Figure 2.6: The general structure of an MA [4]

Figure 2.6 will show us the general structure of an MA in its incremental version. The detailed step-by-step of MA will be discussed further in Chapter 3. In this section, we will focus on clarifying *reproduction* and *improvement step* in MA.

- ***Line 4 - Reproduction.*** In this step, we will use **crossover** - a genetic operator to vary the programming of a chromosome or chromosomes from one generation to the next. More specifically, each gene is selected randomly from one of the corresponding genes of the parent chromosomes, which we call **uninform crossover.** Beside crossover operator, we combine our algorithm with **mutation operator** to increase the chance MA produce a better population whose chromosomes are nearer to our desired solution

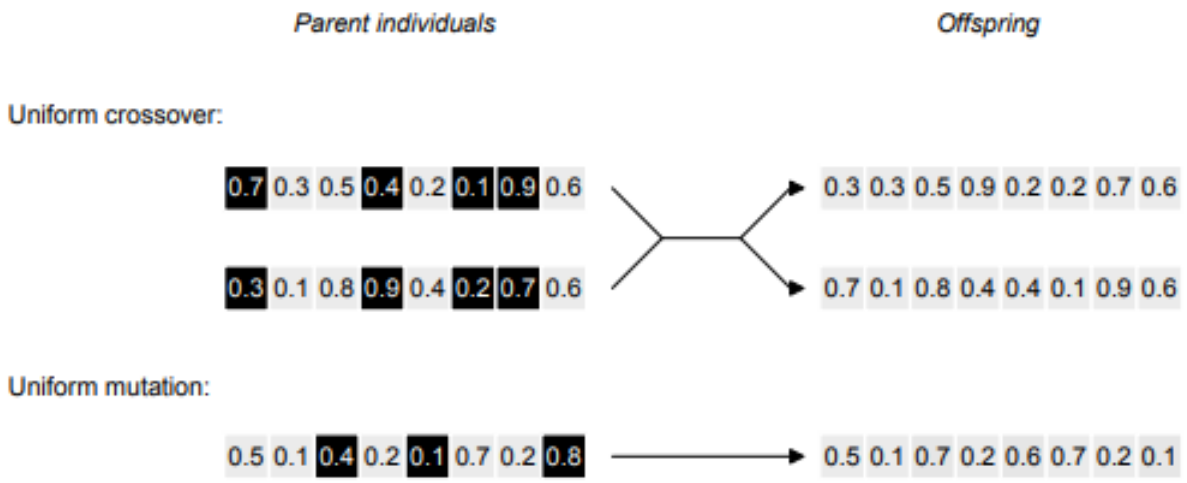

Figure 2.7: Uninformed crossover and uninformed mutation [3]

- ***Line 5 - Improvement.*** As the name of this step, we will use a local search algorithm which is **Hill Climbing** to find "good" children after reproduction. Basically, this algorithm will continuously move in the direction of increasing elevation/value to find *the peak of the mountain*, which we can understand is the best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.

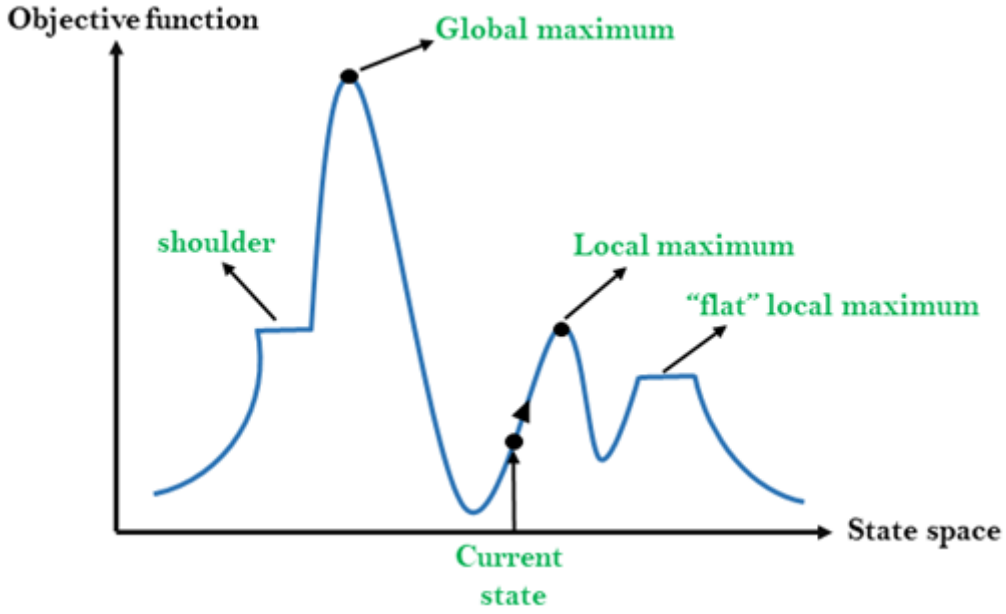

Figure 2.8: Local search illustration

Chapter 3

METHODOLOGY

This section gives detailed descriptions of the MA-Net algorithm proposed by Dong et al., 2020 [6]. Firstly, we shall give the outline of the proposed algorithm. Next, we present the details of the proposed method with respect to the three aspects as discussed in Section 2.2, which are search space, search strategy, and performance estimation strategy

3.1 Algorithm Outline

Algorithm 1: Framework of the proposed MA-Net

$P_0 \leftarrow$ Initialize the population with the new designed population initialization strategy;
Evaluate the fitness of individuals in P_0 with the proposed evaluation technique;
$t \leftarrow 0$;
while *termination criterion is not satisfied* **do**
    $S \leftarrow$ Select parent solutions with slack binary tournament selection;
    $Q_t \leftarrow$ Generate offsprings with crossover and mutation operators from S;
    Evaluate the fitness of individuals in Q_t;
    Use the well designed local search strategy to explore the multi-branch search space for each individual in Q_t then evaluate the extra generated architectures and update each individual;
    $P_t + 1 \leftarrow$ Environmental selection from $P_t \cup Q_t$;
    $t \leftarrow t + 1$;
end
Select the best individual from P_t and decode it into the corresponding convolutional neural network.

Figure 3.1: Pseudo code of MA-Net

Figure 3.1 shows the workflow of the proposed MA-Net. There are main components we should note in this algorithm:

- Firstly, the population P_0 is initialized by a new designed strategy (line 1). The detail in Section 3.2
- Secondly, the algorithm will randomly select 2 parents from the population to generate offspring (lines 5-6). The offspring here will be created via uniform crossover and mutation operator
- Thirdly, the generated offspring will undergo a local search strategy to improve their current quality. After a local search, the extra-generated architectures are compared against the initial individual and the initial one will be replaced if the performance of the extra architectures is better (line 8)
- Finally, once termination criteria are satisfied (generally after several generations), the best individual is selected from the final generation, and then decoded into the corresponding convolutional neural network (line 12)

3.2 Search space

Search space contains a set of solutions found in a NAS approach. In a traditional way, we would use three different blocks in the search space, the convolutional layer, the pooling layer and the fully connected layer. However, the fully connected layers are prone to overfitting and heavily depend on dropout regularization (M. Lin et al, 2013) [5]. What's more, the weight in the fully connected layers is extremely larger than that in the convolutional layers. Thus, the proposed algorithm employs the global average pooling layer to replace the fully connected layer

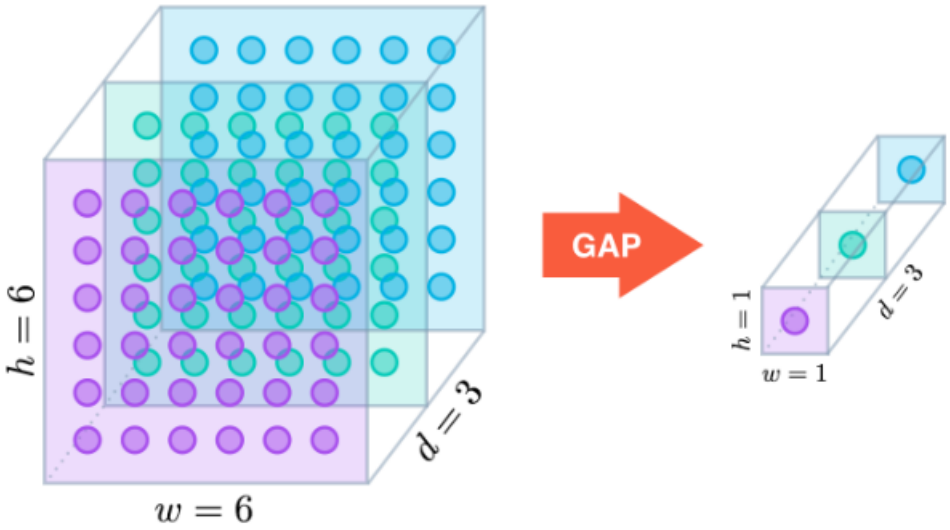

Figure 3.2: Global pooling layer [2]

The global pooling layer has the following advantages:

- There is no weight in this layer, which means this layer can save large quantities of calculation

- The global pooling layer could strengthen the correspondence between the feature maps and the CNN model

To optimize connections between layers of the CNN model, the search space will be multi-branched (Figure 3.3). A node in the graphs stands for either a convolution layer or a pooling layer. Different colors mean different settings or types. Each connection shows the flow of information between two nodes.

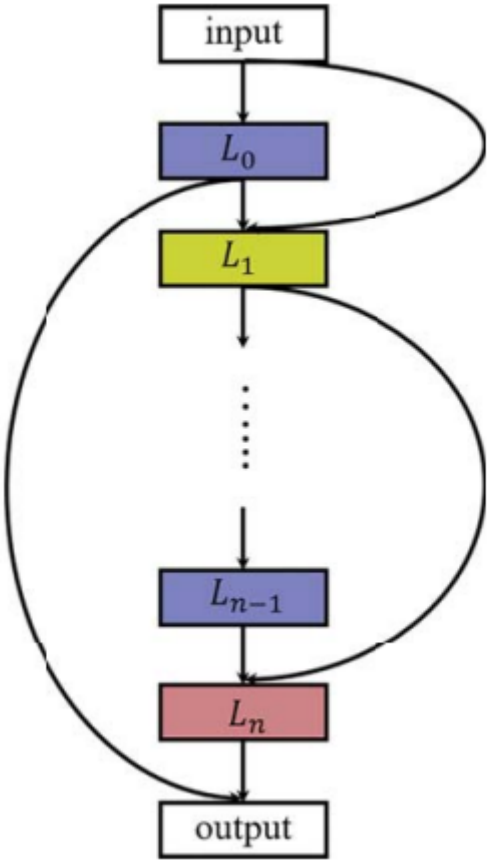

Figure 3.3: A Multi-branch search space

After defining the search space, we will have the population initialization strategy below:

Algorithm 2: Population Initialization

Input: The population size N, the maximal number of convolutional and pooling layers N_{cp}.
Output: Initialized population P_0.

1 $P_0 \leftarrow \emptyset$;
2 $n_c \leftarrow$ Randomly generate an integer between [1, N_{cp}];
3 $n_p \leftarrow$ Randomly generate an integer between [1, $min\{n_c - 1, N - n_c\}$];
4 $A_0 \leftarrow$ Generate an array whose length is n_c, and all elements in this array are set to 1;
5 Randomly select n_p nodes from A_0, except the first node. Then insert number zero before all choosed nodes in A_0;
6 $i \leftarrow 0$;
7 **while** $i < N$ **do**
8 **if** $A_0[i] == 1$ **then**
9 $l \leftarrow$ Initialize a convolutional layer with random settings;
10 **else**
11 $l \leftarrow$ Initialize a pooling layer with random settings;
12 **end**
13 $P_0 \leftarrow P_0 \cup l$;
14 $i \leftarrow i + 1$;
15 **end**
16 $part_2 \leftarrow$ Initialize a global average pooling layer;
17 $P_0 \leftarrow P_0 \cup part_2$;
18 Return P_0.

Figure 3.4: Pseudo code for population initialization

In this algorithm (Figure 3.4):

- P_0 is an empty array that is used to store the generated convolutional layers and pooling layers (line 1)
- n_c and n_p are randomly generated numbers of the convolutional layer and pooling layer (lines 2-3), respectively
- A_0 denotes a sequence, describing how the convolutional layers and pooling layers stack. For example:

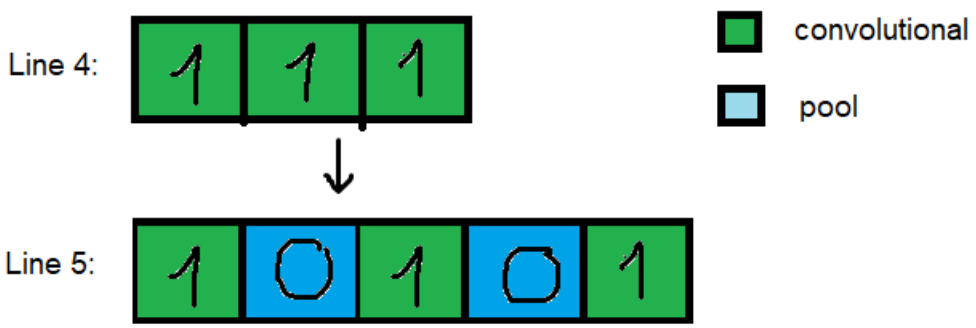

Figure 3.5: A_0 with $n_c = 3$ and $n_p = 2$

- Loop in lines 7 - 15 is to initialize layers with the random setting. Finally, an average pooling layer is added (line 17) to produce a complete architecture (line 18)

3.3 Search strategy

The link between layers is important to produce a good CNN architecture. The search strategy is used to explore the search space. In this section, we will mainly discuss crossover, mutation and local search procedures in multi-branch search space.

But before, we need an encoder to represent the outer connection of network architecture. Shown as in Figure 3.6

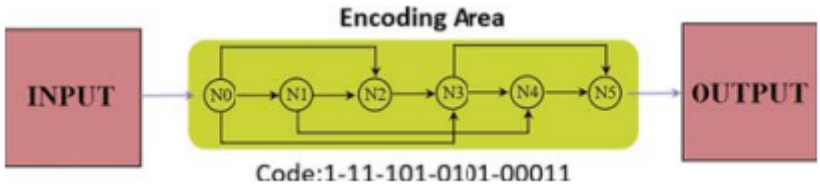

Figure 3.6: The encoded sequence expresses the outer connection of the network architecture

Each node which is a convolutional layer is assigned a number in the encoding area. Node N_i will have 1 bit to represent the connection between the node N_j $(j < i)$. For example, node N_3 has 3 bits. Each bit is "1", "0", and "1" indicates N_3 receives information from N_0 and N_2 (or N_3 connects to N_0 and N_2).

3.3.1 Crossover

Crossover will randomly choose some nodes from parent to create offspring by node exchanging (the link will be exchanged too)

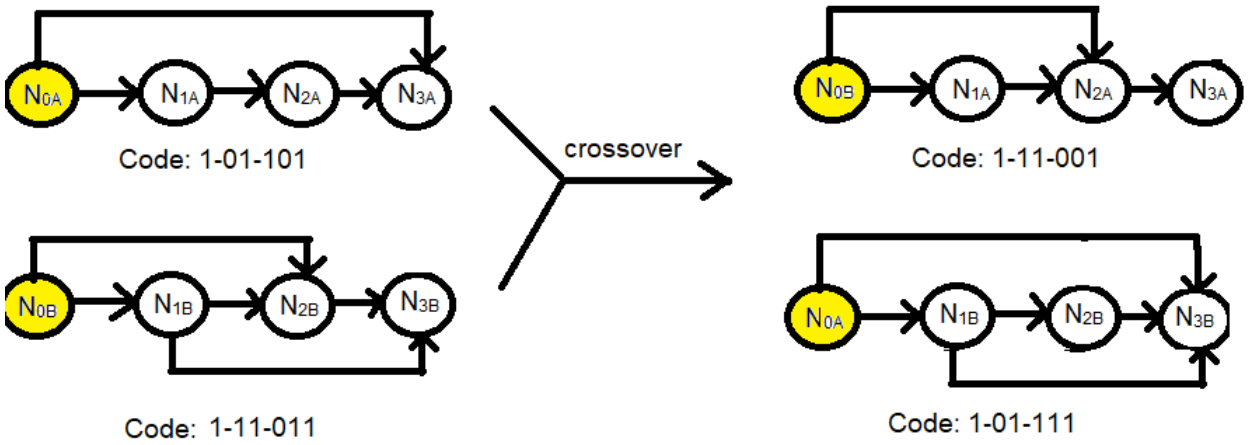

Figure 3.7: Crossover operation

3.3.2 Mutation

Each bit in code representation will be flipped with a certain probability

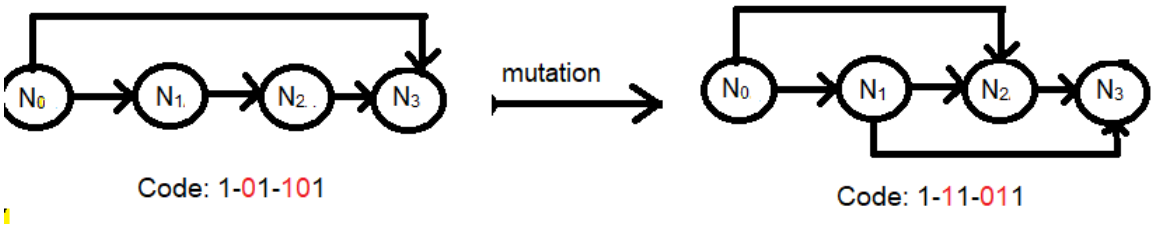

Figure 3.8: Crossover operation

3.3.3 Local search

Local search procedure will be given in Figure 3.9:

Algorithm 3: Local Search

Input: Individual I_0, the number N.
Output: An array of individuals A_{ind}.

1 **if** I_0 *doesn't have outer connection sequence* **then**
2 | generate the initial conntection sequence for I_0;
3 **end**
4 $A_c \leftarrow$ Generate all possible conntection sequences for I_0;
5 $C_0 \leftarrow$ The conntection sequence of I_0;
6 Remove the same sequence as C_0 from A_c;
7 $A_{ind} \leftarrow \emptyset$;
8 $length \leftarrow$ Length of A_c;
9 $N' \leftarrow min\{N, length\}$;
10 Random select N' non-repeating sequences from the A_c to add to the A_{ind};
11 Return A_{ind}

Figure 3.9: Local search pseudo code

The pseudo-code seems a little confusing. Let's break that down:

- Firstly, If I_0 does not have an encoding, we will initialize an encoding for it (lines 1-3)
- Secondly, generate all possible connection sequences for I_0 (line 4). For example, if we I_0 have 3 nodes. Node N_i can connect up to i node. As our encoding is binary, there are $2^{0+1+2} = 8$ different connecting choices for I_0, and all these choices are stored in the array A_c
- Return A_c without unnecessary and repeating sequence

After the local search, all generated sequences will be evaluated. The best one will be used to represent the outer connection of I_0. Through this strategy, all the individuals in a population could possess an outer connection

3.4 Performance Estimation Strategy

Performance estimation strategy is used to evaluate the solutions of a NAS method. A common way is to train the model on a specific benchmark until it converges. However, this way consumes large amounts of computation resources. In order to save large quantities of computation, we reduce the training time to a few epochs before it converges, based on the hypothesis proposed by (X. Zheng et al, 2019) [7].

Performance Ranking Hypothesis: If Cell A has higher validation performance than Cell B on a specific network and a training epoch, Cell A tends to be better than Cell B on different networks after the training of these networks converge

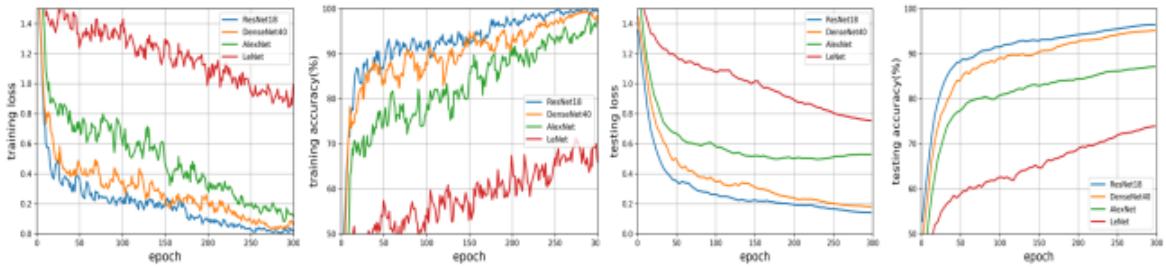

Figure 3.10: Widely used LeNet (Red), AlexNet (Green), ResNet-18 (Yellow), and DenseNet-BC (Blue) neural network architecture to illustrate the proposed Performance Ranking Hypothesis (X. Zheng et al, 2019) [7]

According to the evaluation hypothesis, we can compare the performance of networks found in the proposed algorithm in the early stage of the training process. Therefore, we can simply train these solutions in several or even one epoch to estimate their performance on different datasets.

Chapter 4

EXPERIMENT AND RESULTS

4.1 Experiment

To verify the performance of the proposed MA-Net, we first introduce the datasets and experimental settings. Next we present our experimental results and discuss the results on different datasets.

Benchmark Datasets

In these experiments, nine widely used image classification Benchmark datasets are used to examine the performance of the proposed MA-Net method. They are the MNIST with Background Images (MBI), the MNIST with Random Background (MRB), the MNIST with Rotated Digits (MRD), and the MNIST with RD plus Background Images (MRDBI) benchmark.

(b) Examples from the variations on the MNIST benchmark with 10 classes. From left to right, every two images are from the same benchmark dataset, and they are MBR, MBI, MRD, MRDBI.

Figure 4.1: Examples of the variations about MNIST benchmark

These four benchmark datasets are all 28 × 28 with single-channel as shown in Figure 4.1. Each benchmark only has 12,000 images in the training set but 50,000 images in the test set, which means these classification tasks become much more imbalance and difficult than the original MNIST benchmark.

4.2 Results

All the experimental results compared MA-Net against other methods in the variations of MNIST benchmarks are summarized in the Figure 4.2, respectively. The last row show the classification errors obtained by the proposed MA-Net, and the other columns show the corresponding results of other methods on MRD, MRB, MBI, MRDBI

Classifier	MRD	MRB	MBI	MRDBI
CAE-2	9.66	10.9	15.5	45.23
TIRBM	4.2			35.5
PGBM+DN-1		6.08	12.25	36.76
ScatNet-2	7.48	12.3	18.4	50.48
RandNet-2	8.47	13.47	11.65	43.69
PCANet-2(softmax)	8.52	6.85	11.55	35.86
LDANet-2	7.52	6.81	12.42	38.54
SVM+RBF	11.11	14.58	22.61	55.18
SVM+Poly	15.42	16.62	24.01	56.41
NNet	18.11	20.04	27.41	62.16
SAA-3	10.3	11.28	23	51.93
DBN-3	10.3	6.73	16.31	47.39
EvoCNN(best)	5.22	2.8	4.53	35.03
MA-Net(random)	3.33	2.48	3.56	15.92

Figure 4.2: The classification errors of the proposed EvoCNN method against the peer competitors on the MB, MRD, MRB, MBI, MRDBI benchmark datasets

4.3 Conclusion

In this paper, we have developed an evolutionary approach to automatically evolve the architectures and weights of CNNs for Handwritten Digit Recognition problem. This goal has been successfully achieved by proposing an EA algorithm - Memetic Algorithm and Neural Network Architecture (NAS) base.

More specific, We have presented MA-Net, a NAS method based on EvoCNN and the memetic algorithm. Experimental studies have verified that this algorithm can find better architectures with fewer parameters and fewer computational resources when compared to EvoCNN. In the future, we would like to explore more appropriate crossover and mutation operators to accelerate the automation process. In addition, we would also like to apply the proposed approach to real-world deep learning applications.

To easily understand, in our study, we have combined an EAs - Memetic Algorithm and CNN to give an answer to one big question ***How to build a Neuron Network to solve a Handwrritten Digit Problem?***

Bibliography

[1] Bảo, N. Neural architecture search là gì. https://ngbao161199.github.io/research/2020/04/15/Neural-Architecture-Search-l%C3%A0-g%C3%AC.html, Apr 2020.

[2] Cook, A. Global average pooling layers for object localization. https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/, Apr 2017.

[3] Kampouridis, M., and Barril Otero, F. Evolving trading strategies using directional changes. *Expert Systems with Applications 73* (05 2017), 145–160.

[4] Labadie, N., Prins, C., and Prodhon, C. *Metaheuristics for vehicle routing problems*. John Wiley & Sons, 2016.

[5] Lin, M., Chen, Q., and Yan, S. Network in network. *arXiv preprint arXiv:1312.4400* (2013).

[6] Wang, B., Sun, Y., Xue, B., and Zhang, M. Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification. In *2018 IEEE Congress on Evolutionary Computation (CEC)* (2018), pp. 1–8.

[7] Zheng, X., Ji, R., Tang, L., Zhang, B., Liu, J., and Tian, Q. Multinomial distribution learning for effective neural architecture search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019), pp. 1304–1313.