VICTORIA UNIVERSITY OF WELLINGTON
*Te Whare Wānanga o te Ūpoko o te Ika a Māui*

School of Engineering and Computer Science
*Te Kura Mātai Pūkaha, Pūrorohiko*

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

# Evolving Deep Neural Networks for Image Classification

Bin Wang

Supervisors: Bing Xue, Yanan Sun, Mengjie Zhang

Submitted in partial fulfilment of the requirements for
Bachelor of Science with Honours in Computer Science.

**Abstract**

Convolutional Neural Networks (CNNs) have shown their dominance in solving image classification problems. However, the architectures of CNNs vary in different image classification tasks, which makes the design of the architectures become an active, but challenging research area.This project aims to use Evolutionary Computation (EC) automatically evolve the architectures of CNNs, where three contributions are achieved: Firstly, two EC methods - Differential Evolution (DE) and Genetic Algorithm (GA) with an existing encoding strategy, are developed; Secondly, a hybrid DE approach is proposed to break the major limitation of the two aforementioned methods; Lastly, we propose a new hybrid two-level EC method to automatically evolve more advanced CNNs with shortcut connections, which were invented last year in ResNet and DenseNet. By comparing with the state-of-the-art algorithms, all of the proposed methods are capable to achieve competitive or even better performance, and among the three contributions in this project, the latter-proposed method outperforms the previous methods.

# Acknowledgments

I am profoundly grateful to my supervisor, Bing Xue, Yanan Sun and Mengjie Zhang, whose expert advice has guided me through every step of my writing of this project. They gave me lots of inspirations and encouragement. They read every draft report of this project, and gave me important feedback. Their broad and profound knowledge helped me a lot.

# Contents

# Figures

# Chapter 1

# Introduction

Image analysis is the process of extracting meaningful information from images. In the past two decades, along with the exponential growth of the technologies, tremendous images have been created and collected, and the growing trend is very likely to keep for a long time. As a result, image analysis has become more and more popular because heaps of useful knowledge can be extracted from the images, which produces promising results. There are a few fields in image analysis among which image classification is one of the most crucial and widely-used fields. Generally speaking, image classification is the task of extracting the classes of images from a set of images each of which belongs to a specific class. For example, suppose there is an image dataset of handwritten digits from 0 to 9, matching the images to the corresponding digits is an image classification task.

## 1.1 Motivations

Image classification is a difficult machine learning task due to a couple of reasons. First of all, The dimensionality of the input image is very high. The image is comprised of a number of pixels each of which is one dimension. Assume the image size is 256 pixels $\times$ 256 pixels, which is much smaller than the real-life images, the dimensionality is $65,536$, which is huge comparing to other machine learning tasks. Secondly, the diversity of images in the same class can be large. In order to correctly distinguish the images of various classes, the variability of images in the same class needs to be minimised and the variability of images between different classes has to be maximised. The large diversity of images in the same class makes it extremely difficult to minimise the variability of images in the same class, which therefore causes the complication in image classification.

Convolutional Neural Networks (CNNs) have shown their dominating spot in various machine learning tasks, such as speech recognition [1][2], sentence classification [16][14] and especially, image classification [18][20]. However, from the existing efforts taken by researchers such as LeNet [21][22], AlexNet [18], VGGNet [29], SqueezeNet [11], Inception [38], Xception [5] and GoogLeNet [37], it can be found that designing CNNs for specific tasks is very complicated. There are a couple of main issues when utilising CNNs to solve machine learning tasks of image classification, which are listed below.

- It is challenging to manually search for the optimal architecture of CNNs including the number of convolutional layers, pooling layers and fully-connected layers, the attributes of each layer, and the order of the layers in the architecture of CNNs;
- Different machine learning tasks need different CNN architectures, so personalised architectures of CNNs need to be designed for specific tasks. There are tremendous

image classification tasks in real-world applications, which requires different CNNs to solve.

Since the limitations of manually designing the architectures of CNNs have been raised more frequently in recent years, neuroevolution [31], which uses evolutionary algorithms to generate artificial neural networks (ANN), has come into the spotlight to resolve the issues. Interested researchers have accomplished promising results on the automatic design of the architectures of CNNs by using Genetic Programming [36][42] and Genetic Algorithms (GAs) [33][32]. The project aims to explore more efficient and effective methods to automatically evolve the architectures of CNNs, which are comprised of three parts. First of all, since the potential of Differential Evolution (DE), GAs and Particle Swarm Optimisation (PSO) used in evolving CNNs is not fully investigated, based on the encoding strategy of an IP-Based Particle Swarm Optimisaion (IPPSO) [40] proposed in our summer project, which achieves encouraging results, the methods using DE and GAs with the same encoding strategy are developed and compared with the state-of-the-art algorithms and IPPSO; Furthermore, during the experiments in the first part, a limitation of the above Evolutionary Computation (EC) methods is exposed, which is that the maximum-length of the CNN architectures is fixed, so in the second part, A hybrid DE approach is proposed in order to improve the performance by breaking the maximum-length limitation; Last but not least, in the past two years, shortcut connections used in DenseNet [10] have been proven its effectiveness and efficiency in Deep CNNs, but in the first two parts of this project, only the traditional CNN architectures without shortcut connections are evolved, so a hybrid two-level method is proposed to evolve both the structure of CNN architectures and the shortcut connections between layers.

## 1.2 Goals

The overall goal of this project is to design and develop effective and efficient EC methods to automatically search for good architectures of CNNs, and analyse and compare their performance with the state-of-the-art algorithms. The specific objectives of this project are to

1. Develop effective DE and GA methods using the same encoding strategy in IPPSO to automatically evolve the architecture of CNNs and obtain a good architecture of CNNs. To be more specific, the proposed DE and GA methods are expected to be able to search for the total number of layers, choosing the layer type for each layer, and obtain the best parameter values for each layer in order to find the CNN architecture achieving an optimal performance. For example, the DE method might find a good architecture of CNNs with 5 layers, having a convolutional layer at the third layer and having a feature map of $2 \times 2$ convolutional mask as a parameter of the convolutional layer;

2. Design and develop a hybrid DE approach which are composed of three main steps - refining the existing IP-Based encoding scheme [40] to break the constraint of predefining the maximum depth of CNNs; developing new mutation and crossover operators for the proposed method, which can be applied on variable-length vectors to conquer the fixed-length limitation of the traditional DE method; and designing and integrating a second crossover operator into the proposed method to produce the children in the next generation representing the architectures of CNNs whose lengths differ from their parents;

3. Design and develop a hybrid two-level method, which is able to automatically evolve both the structure of CNN architectures and the shortcut connections. The first level of the evolution searches for the structure using PSO, e.g. the number of layers and the layer types; while at the second level of the algorithm, the topology of the shortcut connections is evolved by DE given the structure obtained at the first level;

4. Since training a CNN can be crawling, and the fitness function used by many other researchers evaluate each architecture of CNNs on the given dataset by fully training the CNNs, whose accuracy would be used as the fitness value of the individual, the whole evolution process can take an unacceptable amount of time. In order to make the IP-Based EC methods as efficient solutions, this project aims to design an efficient fitness evaluation method to significantly reduce the time of the evolution process by reducing the number of epochs and raising the learning rate to obtain the fitness value faster, which is used as the fitness evaluation of all of the algorithms in this project.

## 1.3 Major Contributions

The following contributions are offered by this project:

1. DE and GA methods based on IP-Based encoding strategy are developed, and DE, GA and PSO methods of evolving CNNs are investigated by comparing their performance. The IP-Based encoding strategy proposed by us has been published in the proceedings of 2018 IEEE World Congress on Computational Intelligence [40];

2. A second crossover in the hybrid DE method is proposed to evolve the length of CNNs, and a new DE method is developed by extending the DE operators to be able to cater for individuals with variable-length. The hybrid DE has been accepted as a paper in the proceedings of The Australasian Joint Conference on Artificial Intelligence 2018 [41];

3. A new CNN architecture with additional connections between layers that are not next to each other inspired by ResNet [9] and DenseNet [10] is designed, and a two-level hybrid EC method is developed to evolve the new CNN architecture. We plan to form a paper, with title of A Two-level Hybrid Evolutionary Computation Method to Evolve Dynamically-connected Convolutional Neural Networks, based on the two-level hybrid EC method, which may be submitted to 2019 IEEE Congress on Evolutionary Computation;

4. A new fitness evaluation method is developed to reduce the computational cost but still keep the ability of learning the performance trend of the CNNs.

## 1.4 Organisation

The rest of the report is organised as follows:

- Chapter 2 provides the background of this project consisting of brief introductions of CNN architectures, EC algorithms, Internet Protocol Address, and the related work of using EC methods to evolve CNN architectures;

- Chapter 3 describes the IP-Based GA and DE methods, and the comparison and analysis of the experimental results of IP-Based GA, DE and PSO;

- Chapter 4 illustrates the hybrid DE method, and analyses its performance;
- Chapter 5 shows the details of the two-level hybrid EC method, its experimental designs and the result analysis;
- Chapter 6 concludes the project by analysing the performance of all of the EC algorithms developed in this project and pointing out the future work in the area of utilising EC methods to evolve CNN architectures;
- Appendix lists the resources that are related to this project, but can not be deemed as the main part of the project.

# Chapter 2

# Background

## 2.1 Convolutional Neural Networks

A typical Convolutional Neural Network (CNN) is drawn in Fig. 2.1, whose architecture is constituted of four types of layers - convolution layer, pooling layer, fully-connected layer and output layer. The output layer depends only on the specific classification problem. For the example of image classification, the number of classes decides the size of the output layer. Therefore, when designing an architecture of CNNs, the output layer is fixed once the specific task is given. However, to decide the other three types of layers, first of all, the depth of the CNN architecture has to be decided; Then, the type of each layer needs to be chosen from convolution layer, pooling layer and fully-connected layer; Last but not least, since there are different sets of attributes for different types of layers - filter size, stride size and feature maps for the convolution layer; kernel size, stride size and pooling type enclosing max-pooling or average pooling for the pooling layer; and the number of neurons for the fully-connected layer, the attributes of each layer have to be tuned based on its layer type in order to accomplish a CNN architecture that can obtain good performance.

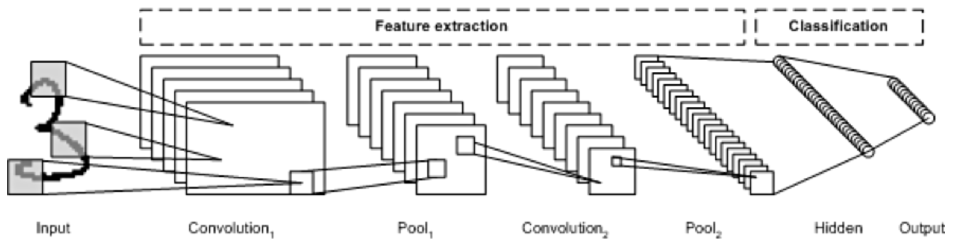


Figure 2.1: An typical architecture of the Convolutional Neural Network [12]

## 2.2 Internet Protocol Address

As the proposed encoding strategy used by the first two contributions of this project is inspired by the Internet Protocol Address, it is essential to introduce IP Protocol Address. An Internet Protocol address (IP address) is a numerical label assigned to each device connected to a computer network that uses the Internet Protocol for communication [25]. A standard IP address is made of four decimal integers concatenated by dots, each of which ranges from 0 to 255, e.g. 172.16.254.1. Another representation of the IP address is to convert the decimal values into the corresponding binary numbers with a fixed-length of 8 bits, e.g. 1010 1100.0001 0000. 1111 1110. 0000 0001 as the binary representation of the aforementioned example of the standard IP address - 192.168.1.101.

In the computer network, an IP address is configured into a network card on a device, e.g. a desktop, a laptop or a server. The IP address is used as the identifier of the device in a local network, so the devices can find and talk to each other within the network. Furthermore, if the devices in different local networks want to communicate with each other, the subnet can be utilised to fulfil the objective. A subnet defines a specific local network by specifying an IP range of the subnet, so the target IP address can be easily routed to the correct subnet and the message from the source device can be effectively delivered to the device carrying the target IP address. For example, suppose there are two IP address - 172.16.254.1 and 172.16.255.1, and the corresponding IP ranges of their subnets are [172.16.254.1 - 172.16.254.255] and [172.16.255.1 - 172.16.255.255], when the device with the IP of 172.16.254.1 intends to send a message to the device with the IP address of 172.16.255.1, if the idea of subnets is not introduced, the message will not be successfully delivered because the two local networks are isolated during the process of searching an IP address; however, if the subnets are set, the message and the target IP address will be routed to the correct subnet first, and then the message can be sent to the target IP address by seeking the target IP address within the subnet.

## 2.3 Evolutionary Computation Algorithms

Since the main target of this project is to use EC methods including GAs, DE and PSO, and their hybrid methods to evolve the architectures of CNNs, in the following sub-sections, the background of these three EC algorithms is given.

### 2.3.1 Differential Evolution

Differential Evolution (DE) is a population-based EC method which searches for the optimal solutions of a problem. It has been proved to be a simple and efficient heuristic method for global optimisation over continuous spaces [35][34]. Overall, there are four major parts in a DE algorithm, which are initialisation, mutation, crossover and selection based on the fitness [26]. First of all, a population of candidate vectors are randomly initialised. Secondly, mutation is applied according to Formula (2.1), where $\mathbf{v}_{i,g}$ means the *ith* temporary candidate vector of the *gth* generation; $\mathbf{x}_{r0,g}$, $\mathbf{x}_{r1,g}$ and $\mathbf{x}_{r2,g}$ indicate three randomly picked candidates of the *gth* generation; and $F$ is the differential rate, which is used to control the evolution rate. Thirdly, the crossover is performed based on Formula (2.2), where $u_{j,i,g}$ represents the *jth* dimension of the *ith* candidate at the *gth* generation. At the beginning of the crossover process for each candidate, a random number $j_{rand}$ is generated, and then for each dimension of each candidate vector, another random number $rand_j$ is generated, which then is compared with the crossover rate $Cr$ and $j_{rand}$ as shown in Formula (2.2) to decide whether the crossover applies on this dimension. After applying the DE operators, a trial vector $\mathbf{u}_{i,g}$ is produced, which is then compared with the parent vector to select the one that has better fitness. By iterating the mutation, crossover and selection until the stopping criterion is met, the best candidate can be found.

$$\mathbf{v}_{i,g} = \mathbf{x}_{r0,g} + F \times (\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g}) \tag{2.1}$$

$$u_{j,i,g} = \begin{cases} v_{j,i,g} & \text{if } rand_j(0,1) < Cr \text{ or } j = j_{rand} \\ x_{j,i,g} & \text{otherwise} \end{cases} \tag{2.2}$$

### 2.3.2 Genetic Algorithm

Genetic Algorithm (GA) is a metaheuristic optimisation method inspired by the process of natural selection in the area of biology. The bio-inspired operators, such as mutation, crossover and selection, are utilised to evolve the population in order to obtain a high-quality solution [24]. The procedure of GA is composed of five parts - initialisation, selection, mutation, crossover and fitness evaluation. At the stage of initialisation, a random vector of a fixed dimension is repetitively generated and stored in an individual until reaching the population size; Next, the selection is performed by using a selection algorithm to select the individuals into a mating pool; After that, one individual is selected from the mating pool and the value of each dimension is randomly chosen to be changed in order to evolve a new individual; Another way of creating a new individual is to select two individuals in the mating pool and apply crossover operator by combining a part of the dimensions of one individual's vector with those of the other. By iterating the selection, mutation, crossover and fitness evaluation, the new population can be filled with new individuals to form a new generation, which can then be evaluated by the fitness evaluation function to find the best individual. The best individual is recorded for each generation, which is compared with the best individuals of the previous generations to output the best individual of all generations. The whole process terminates when the stopping criteria are met, and the best individual of all generations is reported as the evolved solution.

### 2.3.3 Particle Swam Optimisation

Particle Swarm Optimization (PSO) is a population-based algorithm, motivated by the social behaviour of fish schooling or bird flocking proposed by Kennedy and Eberhart in 1995 [15] [7]. In PSO, there is a population consisting of a number of candidate solutions also called particles, and each particle has a position and a velocity. The representation of the position is described in Formula (2.3), where $\mathbf{x_i}$ is a vector of a fixed dimension representing the position of the $ith$ particle in the population and $x_{id}$ means the $dth$ dimension of the $ith$ particle's position. Formula (2.4) illustrates the velocity of a particle, where $\mathbf{v_i}$ is a fix-length vector expressing the velocity of the $ith$ particle and $v_{id}$ means the $dth$ dimension of the $ith$ particle's velocity. The way that PSO solves the optimisation problems is to keep moving the particle to a new position in the search space until the stopping criteria are met. The position of the particle is updated according to the update equation which incorporates two equations - the velocity update equation 2.5 and the position update equation 2.6. In Formula (2.5), $v_{id}(t+1)$ indicates the updated $dth$ dimension of the $ith$ particle's velocity, $r_1$ and $r_2$ carry random numbers between 0 and 1, $w, c_1$ and $c_2$ are PSO parameters that are used to fine-tune the performance of PSO, and $P_{id}$ and $P_{gd}$ bear the $dth$ dimension of the local best and the global best, respectively. After updating the velocity of the particle, the new position can be achieved by applying Formula (2.6).

$$\mathbf{x_i} = (x_{i1}, x_{i2}, ...x_{id}) \tag{2.3}$$

$$\mathbf{v_i} = (v_{i1}, v_{i2}, ...v_{id}) \tag{2.4}$$

$$v_{id}(t+1) = w * v_{id}(t) + c_1 * r_1 * (P_{id} - x_{id}(t)) + c_2 * r_2 * (P_{gd} - x_{id}(t)) \tag{2.5}$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \tag{2.6}$$

## 2.4 Related Work

Recently, more and more research has been done using EC methods to evolve the architectures of CNNs. Genetic CNN [42] and CGP-CNN [36] are two of the most recent proposed methods that have achieved promising results in comparison with the state-of-the-art human-designed CNN architectures.

Genetic CNN uses a fixed-length binary string to encode the connections of CNN architectures in a constrained case. It splits a CNN architecture into stages. Each stage is comprised of numerous convolutional layers which may or may not connect to each other, and pooling layers are used between stages to connect them to construct the CNN architecture. Due to the fixed-length binary representation, the number of stages and the number of nodes in each stage have to be predefined, so a large fraction of network structures are not explored by this algorithm. Other than that, the encoding scheme of Genetic CNN only encodes the connections, i.e. whether two convolutional layers are connected or not; while the hyperparameters of the convolutional layers, e.g. the kernel size, the number of feature maps, and the stride size, are not encoded, so Genetic CNN does not have the ability to optimise the hyperparameters.

CGP-CNN utilises Cartesian Genetic Programming (CGP) [23] because the flexibility of CGP's encoding scheme is suitable to effectively encode the complex CNN architectures. CGP-CNN employs a matrix of $N_r$ rows and $N_c$ columns to represent the layers of a CNN architecture and their connections, respectively, so the maximum number of layers is predefined. In addition, as six types of node functions called ConvBlock, ResBlock, max pooling, average pooling, concatenation and summation are prepared, CGP-CNN is confined to explore the limited types of layers of CNN architectures. Last but not least, from the experimental results, the computational cost of CGP-CNN is quite high because training CNNs in fitness evaluation is time-consuming.

In summary, manually design of CNN architectures and parameters is very challenging and time-consuming. Automatically evolving the architectures of deep CNNs is a promising approach, but their potential has not been fully explored. DE has shown as an efficient method in global optimisation but has not been used to evolve deep CNNs. Therefore, we would like to investigate new approaches using EC methods to automatically evolve the architectures of deep CNNs.

# Chapter 3

# IP-Based EC algorithms

## 3.1 Introduction

In this part of the project, GAs and DE algorithms are investigated and compared with the PSO algorithm proposed by us with the name of IPPSO [40], which has already achieved promising results in image classification tasks. There are a couple of motivations for the investigation. First of all, the IP-Based Encoding Strategy (IPES) used in IPPSO is easy to be applied on GAs and DE algorithms; Secondly, the potential of GAs and DE applications for evolving CNNs is not fully investigated, so it is worth doing the investigation and comparison.

**Goals:** The goal of this part is to propose DE and GAs methods using IP-Based encoding strategy and compare the performance of these two methods with the state-of-the-art algorithms and the existing IPPSO. The specific objectives are

- design and develop a DE-Based method to evolve CNN architectures based on IP-Based encoding strategy;
- design and develop a GA-Based method to automatically design the architectures of CNNs using the IP-Based encoding strategy;
- investigate the hyperparameters of the fitness evaluation function used in IPPSO, and find patterns and differences between the DE, GA and PSO approaches.

## 3.2 The Proposed Algorithms

Section 3.2.1 introduces the IP-Based Encoding Strategy, which was proposed in IPPSO [40] and adapted in this project; The fitness evaluation method is depicted in Section 3.2.2; The details of he population initialisation are given in Section 3.2.3, which is used by all of the algorithms in this project; Section 3.2.4 gives an overview of the algorithms implemented in this project and the pseudo-code of each algorithm.

### 3.2.1 IP-Based Encoding Strategy

The IP-Based Encoding Strategy proposed in IPPSO [40] is to use one IP Address to represent one layer of CNNs and push the IP address into a sequence of interfaces, each of which bears an IP address and its corresponding subnet, in the same order as the order of the layers in CNNs. The typical CNNs are composed of three types of layers - Conv Layer, Pooling Layer and Fully-Connected Layer. In Section 3.2.1, the process of encoding a CNN into a sequence of IP addresses is elaborated, and in Section 3.2.1, the decoding procedure is described.

**Encoding Procedure** The proposed IP-Based Encoding Strategy is to use one IP Address to represent one layer of CNNs and push the IP address into a sequence of interfaces, each of which bears an IP address and its corresponding subnet, in the same order as the order of the layers in CNNs. The typical CNNs are composed of three types of layers - convolutional layer, pooling layer and fully-connected layer. The first step of the encoding is to work out the range that can represent each attribute of each type of the CNN layer. There are no specific limits for the attributes of CNN layers, but in order to practically apply optimisation algorithms on the task, each attribute has to be given a range which has enough capacity to achieve an optimal accuracy on the classification problems. In this project, the constraints for each attribute are designed to be capable of accomplishing a relatively low error rate. To be specific, for the convolutional layer, there are three attributes, which are filter size ranging from 1 to 8, number of feature maps from 1 up to 128, and the stride size with the range from 1 to 4. As the three attributes need to be combined into one number, a binary string with 12 bits can contain all the three attributes of the convolutional layer, which are 3 bits for filter size, 7 bits for the number of feature maps, and 2 bits for the stride size. Following the similar way, the pooling layer and fully-connected layer can be carried in the binary strings with 5 bits and 11 bits, respectively. The details of the range of each attribute are listed in Table 3.1.

Table 3.1: The ranges of the attributes of CNN layers - Convolutional, Pooling, Fullly-connected layer

| Layer Type | Parameter | Range | # of Bits |
|---|---|---|---|
| Conv | Filter size | [1,8] | 3 |
| | # of feature maps | [1,128] | 7 |
| | Stride size | [1,4] | 2 |
| | **Total** | | 12 |
| Pooling | Kernel size | [1,4] | 2 |
| | Stride size | [1,4] | 2 |
| | Type: 1(maximal), 2(average) | [1,2] | 1 |
| | **Total** | | 5 |
| Fully-connected | # of Neurons | [1,2048] | 11 |
| | **Total** | | 11 |

Once the number of bits of the binary strings has been defined, a specific CNN layer can be easily translated to a binary string. Suppose a convolutional layer with the filter size of 2, the number of feature maps of 32 and the stride size of 2 is given, the corresponding binary strings of [001], [000 1111] and [01] can be calculated by converting the decimal numbers[1] to the corresponding binary numbers. The final binary string that stands for the given convolutional layer is [001 000 1110 01] by joining the binary strings of the three attributes together. The details of the example are shown in Fig. 3.1.

Similar like network engineering where the subnet has to be defined before allocating an IP address to an interface, i.e. a laptop or desktop, the IP-Based Encoding Strategy needs to design a subnet for each type of CNN layers. Since the number of bits of each layer type decides its size of the search space, and the pooling layer takes much fewer bits than the other two, the chances of a pooling layer being chosen would be much smaller than the other two. In order to balance the probability of each layer type being selected, a place-holder of 6 bits is added to the binary string of the pooling layer to make it 11 bits, which brings the odds of picking a pooling layer the same as that of a fully-connected layer. As there are three types of layers with the maximum bits of 12, a 2-byte binary string has sufficient capacity to bear the encoded CNN layers. Starting with the convolutional layer of 12 bits, as this is the

[1]Before the conversion, 1 is subtracted from the decimal number because the binary string starts from 0, while the decimal value of the attributes of CNN layers begins with 1

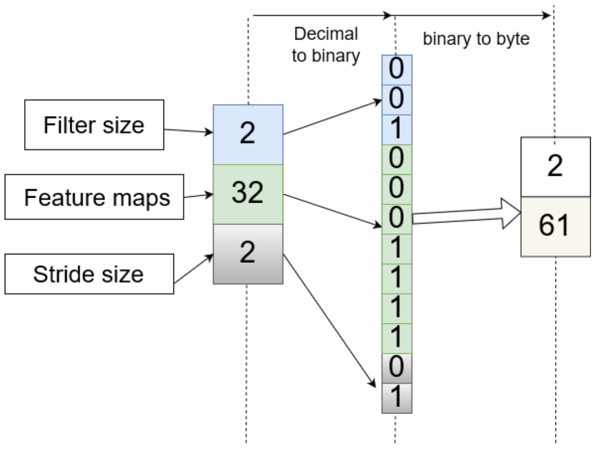


Figure 3.1: An example of how to encode a convolutional layer using a byte array

first subnet, the 2-byte binary representation of the starting IP address would be [0000 0000 0000 0000], and the finishing IP address would be [0000 1111 1111 1111]; The fully-connected layer of 11 bits starts from the binary string [0001 0000 0000 0000] by adding one to the last IP address of the convolutional layer, and ends to [0001 0111 1111 1111]; And similarly, the IP range of the pooling layer can be derived - from [0001 1000 0000 0000] to [0001 1111 1111 1111]. The IP ranges of the 2-byte style for each subset are shown in Table 3.2, which are obtained by converting the aforementioned binary strings to the 2-byte strings. Now it is ready to encode a CNN layer into an IP address, and the convolutional layer detailed in Fig. 3.1 is taken as an example. The binary representation of the IP address is [0000 0010 0011 1001] by summing up the binary string of the convolutional layer and the starting IP address of the convolutional layer's subnet, which can be converted to a 2-byte IP address of [2.61]. Fig. 3.2 shows an example vector encoded from a CNN architecture with 2 convolutional layers, 2 pooling layers and 1 fully-connected layer. In order to encoding CNNs of variable-lengths, any of the layers has the chance to be replaced by a Disabled layer, which is used to mark some of the CNN layers as removed.

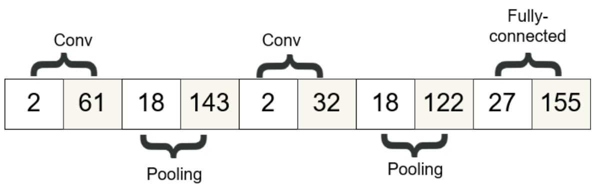


Figure 3.2: An example of the encoded vector of a CNN architecture

Table 3.2: Subnets distributed to the three types of CNN layers

| Layer type | IP Range |
|---|---|
| Convolutional Layer | 0.0-15.255 |
| fully-connected layer | 16.0-23.255 |
| pooling layer | 24.0-31.255 |

**Decoding Procedure** The encoding process transfers the complicated search space of architectures of CNNs to a fixed-dimension search space with the range of 0 to 255 for each dimension, which makes it straightforward to apply optimisation algorithms; While, during the fitness evaluation, it is necessary to decode the sequence of interfaces back to a CNN architecture. First of all, the interfaces in the sequence have to be translated to the values of attributes of CNN layers. In each interface, the IP address can be converted to a binary string, and the number of bits of the attributes of a specific layer type can be retrieved from the subnet in the interface. According to the number of bits of the attributes, the binary string can be split into several binary strings, which can then be converted to decimal values corresponding to the values of the attributes. Next, the CNN architecture can be re-built by creating each layer of the CNN from the parsed attributes and connecting each layer by keeping the same order as they are in the sequence of interfaces.

### 3.2.2 Fitness Evaluation

The fitness evaluation process is illustrated in Algorithm 1. First of all, four arguments are taken in by the fitness evaluation function - the candidate solution which represents an encoded CNN architecture, the number of training epochs for training the model decoded from the candidate solution, the training set which is used to train the decoded CNN architecture, and the fitness evaluation dataset on which the trained model is tested to obtain the accuracy used as the fitness value. Secondly, the fitness evaluation process is pretty straightforward by training the decoded CNN architecture on the training set for a fixed number of epochs, and then obtaining the accuracy on the fitness evaluation set, which is actually used as the fitness value.

**Algorithm 1:** Fitness Evaluation

```
Input: The candidate solution $c$, the training epoch number $k$, the training set $D_{train}$, the
    fitness evaluation dataset $D_{fitness}$;
Output: The fitness value $fitness$;
    Train the connection weights of the CNN represented by the candidate $c$ on the training
    set $D_{train}$ for $k$ epochs;
    $acc \leftarrow$ Evaluate the trained model on the fitness evaluation dataset $D_{fitness}$
    $fitness \leftarrow acc$;
    return $fitness$
```

### 3.2.3 Population Initialisation

The population initialisation process consists of multiple repetitions of the procedure of initialising a candidate until the size of the population is fulfilled. To be specific with the candidate initialisation, first of all, a type of CNN layer is randomly chosen from the four types of CNN layers - Conv layer, Pooling layer, Fully-connected layer and Disabled layer; secondly, once the layer type is set, the range of the IP address is fixed as well, so a random IP address can be generated within the IP range; Thirdly, an interface is initialised by attaching the IP address and the subnet of the layer type as its attributes, which then is pushed in the candidate vector in order; Finally, repeat the previous steps to add more interfaces each of which bears a random CNN layer until the maximum depth of the CNN architecture is accomplished.

### 3.2.4 IP-Based Evolutionary Computation Algorithms

The DE and GA algorithms with the IP-Based encoding strategy are depicted in the following sections. However, for the comparison purpose, there are two other algorithms implemented, which are IPPSO [40] and IP-Based Random Search (IPRS) described in Appendix A.1.

**IP-Based Differential Evolution** The basic DE variant with the mutation scheme of DE/rand/1 is implemented, and the IP-Based Encoding Strategy is utilised to convert the variable-length architectures of CNNs to a fixed number of dimensions in the search space to meet the fixed-length encoding requirement of traditional DE methods. The IP-Based Differential Evolution (IPDE) are detailed as follows.

**IPDE Algorithm Overview** The workflow of IPDE algorithm is to initialise the population, apply the DE genetic operation to update the individual for the whole population, retrieve the best individual from the whole population, repeat step 2 and 3 until the stopping criteria are met, whose pseudo-code is documented in Algorithm 2.

**Algorithm 2:** Framework of IPDE

```
$P \leftarrow$ Initialize the population with IP-Based Encoding Strategy elaborated in Section 3.2.3;
$P_{best} \leftarrow empty$;
while termination criterion is not satisfied do
    Apply the DE genetic operations to update each individual in the population described in Algorithm 3;
    evaluate the fitness value of each individual;
    $P_{best} \leftarrow$ retrieve the best individual in the population;
end while
```

**IPDE Individual Update** By following the standard DE method, the first step of applying the genetic operations on a parent individual is to randomly select three different individuals from the population. The DE mutation cannot be applied to the individual vector because the individual is represented by a vector, and each interface enclosing the information of the IP address and the subnet is stored as one dimension of the individual vector. Therefore, the individual vector has to be converted into a fixed-length vector with a number in each dimension by extracting each byte of the IP addresses in the individual vector. Once the vector conversion is done, the DE mutation can be performed on the three selected individuals to produce a candidate. The crossover can then be performed on the candidate and the parent individual to obtain the final candidate. Finally, the parent individual will be replaced by the candidate if the candidate achieves a better fitness; otherwise, the parent individual will not be changed.

**IP-Based Genetic Algorithm** The IP-Base Encoding Strategy is used to construct a search space with a fixed number of dimensions by encoding the variable-length architectures of CNNs into a byte vector with a fixed length, and GA is implemented to solve the optimisation problem in the search space. The IP-Based Genetic Algorithm (IPGA) method adopts the roulette wheel selection and the two point crossover from the traditional Genetic Algorithm.

**IPGA Algorithm Overview** The framework of IPGA is composed of four main steps - initialise the population, apply GA genetic operations to produce individuals until the size

**Algorithm 3:** IPDE Update An Individual

```
Input: parent $x_i$, population $P$;
Output: child $u_i$;
  $v_i \leftarrow$ Empty candidate
  $x_{r0}, x_{r1}, x_{r2} \leftarrow$ Randomly select three unique individuals from the population $P$;
  $v_i \leftarrow$ Perform DE/rand/1 mutation in Formula (2.1) on $x_{r0}, x_{r1}, x_{r2}$;
  $u_i \leftarrow$ Perform DE crossover in Formula (2.2) on $x_i$ and $v_i$;
  $u_{fitness} \leftarrow$ Decode the child $u_i$ to a CNN and evaluate it;
  if $u_{fitness} <$ the fitness of $x_i$ then
    $u_i \leftarrow x_i$
  end if
  return $u_i$
```

of the child population reach the pre-defined population size, retrieve the best individual from the new population, repeat step 2 and 3 until the stopping criteria are met, which is specified in Algorithm 4.

**Algorithm 4:** Framework of IPGA

```
  $P \leftarrow$ Initialize the population with IP-Based Encoding Strategy elaborated in Section 3.2.3;
  $P_{best} \leftarrow$ Empty best individual;
  while termination criterion is not satisfied do
    Apply the GA genetic operations to produce a new population illustrated in Algorithm 5;
    evaluate the fitness value of each individual;
    $P_{best} \leftarrow$ retrieve the best individual in the population;
  end while
```

**IPGA Generating New Population** At the very beginning the population generation, the elitism process is executed by selecting a certain number of individuals from the population of the current generation and pushing them through to the population of the new generation. In order to fill up the rest of the population of the new generation, an individual is created by applying GA operations on the population of the current generation, which then is added into the population of the new generation, and the process of producing an individual is repeated until reaching the pre-defined population size. The main part of IPGA is that the IP-Base Encoding Strategy is used to transform the variable-length architectures of CNN to the byte vectors with a fixed dimension and vice versa. During the genetic operations of crossover and mutation, the byte vectors are used; while during the fitness evaluation, the decoded architectures of CNNs are utilised. The detailed process of IPGA population generation is explained in Algorithm 5

## 3.3 Experiment Design

### 3.3.1 Benchmark Datasets

In the experiments, six widely-used benchmark datasets are chosen to examine the proposed algorithms, which are the datasets of MNIST Basic (MB), MNIST with a black and white im-

**Algorithm 5:** IPGA Generate A New Population

```
Input: population of the current generation $P$;
Output: population of the next generation $P_{next}$;
  $P_{next}$ ← Find the best individuals of a certain number from the population $P$ and pushed
  them into the new population $P_{next}$
  while the new population $P_{next}$ is not filled up do
    $ind_1, ind_2$ ← Select two individuals from the population $P$ using roulette wheel
    selection
    $child$ ← Apply two point crossover on $ind_1, ind_2$
    $child$ ← Mutate some of the bytes by replacing the number of the byte to be mutated
    with a random number within the range of 0 to 255
    $P_{next}$ ← Add the new individual $child$ into the new population $P_{next}$
  end while
  return $P_{next}$
```

age as the Background Image (MBI), MNIST Digits Rotated with a black and white image as the Background Image (MDRBI), MNIST with a Random Background (MRB), MNIST with Rotated Digits (MRD), and CONVEX. The MB benchmark dataset and its four variants - the MBI, MDRBI, MRB and MRD datasets consist of handwritten digits and the corresponding labels from 0 to 9, and each of the datasets is composed of a training set of 12,000 instances and a test set of 50,000 instances; while convex images and non-convex images with the corresponding labels constitute the CONVEX dataset, which is split into a training set of 8,000 examples and a test set of 50,000 examples.

The reason for picking the six aforementioned datasets is to fulfil the purpose of thoroughly testing the proposed algorithms. First of all, the classification task on the MB dataset and its variants is to classify the handwritten images into the correct labels, which is a multi-class classification problem, and the difficulties of the tasks vary by adding different factors of noises into the MB dataset, so the proposed methods can be tested across multi-class classification problems with diverse complexities. Since the factors of rotation and background are introduced separately into the MBI, MRB and MRD datasets; while a combination of these two factors is applied on the MDRBI dataset, the MDRBI dataset holds the highest complexity among all of the five MNIST datasets. In addition, the binary classification task on the CONVEX dataset is utilised as a complement to the multi-class classification problem of the MNIST datasets in order to extend the evaluation of the proposed algorithms to cover both the multi-class and binary classification tasks. Last but not least, numerous state-of-the-art methods have reported promising results on these benchmark datasets, which makes it convenient to collect the results for the comparison purpose.

### 3.3.2 State-of-the-art Competitors

The state-of-the-art methods, which are reported to have achieved promising results on the aforementioned benchmark datasets in the literature [4] and on the website[2] of the benchmark datasets' provider, are picked as the peer competitors of the proposed algorithms. The state-of-the-art methods are listed as follows: CAE-2 [27], TIRBM [30], PGBM+DN1 [13], ScatNet-2 [3], RandNet-2 [4], PCANet-2 (softmax) [4], LDANet-2 [4], SVM+RBF [19], SVM+Poly [19], NNet [19], SAA-3 [19] and DBN-3 [19].

[2]http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/DeepVsShallowComparisonICML2007

### 3.3.3 Experiment Design

**Experiment Workflow** Fig. 3.3 shows the experimental process, which is used by all of the experiments in this project. Firstly, the dataset is split into training set and test set, and the training set is then divided into training part and test part. The training part and the test part are passed to the EC application which is the proposed EC algorithm. During the fitness evaluation, the training part is used to train the neural network, and the test part is used to obtain an accuracy of the trained neural network, which is used as the fitness value. Secondly, EC application produces the evolved CNN architecture, which is the best individual. Lastly, the training set, test set and the evolved CNN architecture are passed to CNN evaluation, and the test accuracy of the CNN architecture can be achieved, which is the final result of the experiment.

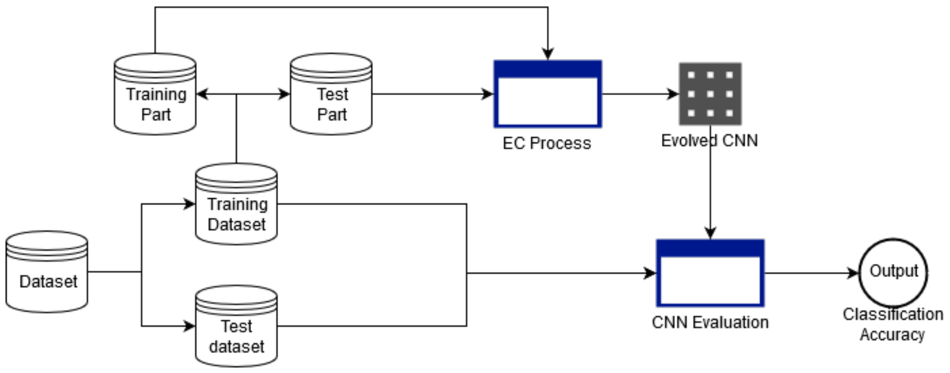


Figure 3.3: The flowchart of the experimental process

**Parameter settings** As neuralevolution is literally comprised of two parts - neural network and evolutionary computation, and evolutionary computation encompasses the evolutionary algorithm (e.g. DE, PSO or GA) and the fitness evaluation, the parameter settings can be split into three groups. In regard to the neural network parameters, based on the complexity of the benchmark datasets that are selected, a CNN architecture with the maximum depth of 10 used. In terms of the parameters of the evolutionary algorithms, as training neural network usually takes tremendous computational cost, the population size and the evolutionary generation has to be limited. After experimenting the population size from 10 to 100 with the step of 10, and the generation from 5 to 15 with the step of 5, the population size and the generation are set to 30 and 5, respectively, by contemplating the time cost for the evolutionary process without compromising the accuracy in the final result. Apart from these two parameters that are commonly used by all of the proposed IPEC methods, the other method-specific parameters are defined in Section 3.3.3. With regard to the parameters of the fitness evaluation, since training the whole dataset takes an enormous amount of time, using the partial dataset tends to drastically speed up the fitness evaluation, which brings the percentage of the dataset as a hyperparameter of the fitness evaluation. In addition, since training the very deep neural network is crawling, it is more efficient to train the neural network for a small number of epochs to learn a trend of the neural network, which is incorporated as another parameter of the fitness evaluation. The two parameters related to the fitness function vary in different experiments, and they will be specified in the following specific experiments.

All of the parameters are configured according to the conventions in the communities of DE [8], PSO [39] and GA [6] along with taking into account the small population and complexity of the search space obtained by applying the IP-Base Encoding Strategy, which are listed in Table 3.3.

Table 3.3: Parameter list

| Parameter Name | Parameter Meaning | Value |
|---|---|---|
| | IPDE | |
| F | differential rate | 0.6 |
| cr | cross over rate | 0.45 |
| | IPPSO | |
| c1 | acceleration coefficient array for $P_{id}$ | 1.49618 |
| c2 | acceleration coefficient array for $P_{gd}$ | 1.49618 |
| w | inertia weight for updating velocity | 0.7298 |
| | IPGA | |
| mr | mutation rate | 0.01 |
| cr | cross over rate | 0.9 |
| er | elitism rate | 0.1 |
| | IPRS | |
| radius | random search radius | 5 |

**Experiments to compare IPECs and the state-of-the-art methods** First of all, the proposed algorithms need to be compared with the state-of-the-art methods in order to prove the competence of these algorithms. There are three IPEC methods - IPDE, IPGA and the previously proposed IPPSO being inspected, and from the statistical point of view, it is necessary to obtain the results from 30 runs for each algorithm, so running the experiments could take a huge amount of time. However, as the fitness evaluation plays a fundamental role in the computational cost, if the fitness evaluation can speed up, the total time of the experiments can accomplish a magnificent plunge. Therefore, the partial dataset with 10% is used to train the CNN architecture during the fitness evaluation and the number of epochs is limited to 5 epochs, which cut down the time of each run of each algorithm on each dataset to a couple of hours approximately.

In addition, using 10% of the dataset might not be able to yield a good model due to the under-fitting issue caused by the lack of training data, so it is worth doing an experiment using the whole training dataset. However, whether to run the experiments of using whole training dataset for only one of the algorithm or all of the three algorithms can be decided based on the results of the experiments with 10% of the dataset because if the difference of the performance among these methods is not obvious, the IPDE method with the whole training dataset can be experimented, where the results are used as representatives of these three algorithms to be compared with the state-of-the-art methods; otherwise, we will perform the experiments of using the whole training dataset for all of the three IPEC algorithms.

**Experiments to compare IPECs and IPRS** In order to verify the effectiveness of IP-Based Encoding Strategy, the experiment of IPRS with 10% of the dataset and 5 epochs will be performed. If the IP-Based random search performs as good as the other EC methods, and it is competitive to the state-of-the-art methods, it will prove that the IP-Based Encoding Strategy is capable to transform the complicated search space of the architectures of CNNs to a search space that is easy to be optimised. Furthermore, if all of the IPEC methods and IPRS with 10% of the dataset gain a competing accuracy to the state-of-the-art methods, it will demonstrate that the optima of the transformed search space correctly reflect the best architectures of CNNs, which can again demonstrate the powerfulness of the IP-Based Encoding Strategy.

**Experiments to compare IPECs with each other** For the purpose of analysing which of the three IPEC methods has better performance, the results of the experiments of all of the three IPEC algorithms are compared by applying the statistical test. As the results of the IPEC methods have been obtained in the above experiments, there are no extra experiments needed.

**Experiments to fine-tune the hyperparameters of fitness evaluation** An experiment is designed to fine-tune the two hyperparameters of fitness evaluation, which are the number of training epochs and the the percentage of dataset. The details of the experiment can be found in Appendix A.2.

**Statistical Tests on the results** Since both Random Search and EC methods are stochastic, statistical significance test is required to make the comparison result more convincing. When comparing the Random Search or EC methods with the state-of-the-art method, One Sample T-Test is applied to test whether the group of samples is better or not because only the best error rates of state-of-the-art methods are reported, but not the statistical results; when a comparison of error rates between any two of the stochastic methods is performed, Two Sample T-test is utilised to determine whether the difference is significant enough or not; when analysing the computational cost, the generation, when the optimised CNN architecture is accomplished, is taken for the statistical comparison, and Mann-Whitney-Wilcoxon (MWW) is chosen as the significance test method because the generation samples are not continuous data.

## 3.4 Results and Discussions

As there are two hyperparamters of the IP-Based method - the percentage of dataset and the training epochs used for fitness evaluation, and each experiment is done with a certain percentage and a fixed number of epochs, in order to make it easier to refer to the method used by a specific experiment, the method name, the percentage of the dataset and the training epochs are concatenated with a delimiter of hyphen as the unique name of the method used in an experiment. Taking IPDE with 10% of dataset and 5 training epochs as an example, the name of IPDE-10%-5 is used wherever the method is referred. Each group of the results are collected from one experiment which performs 30 runs of one unique method on a specific dataset. For example, one group of the result may be obtained from 30 runs of IPDE-10%-5 on the CONVEX dataset. The full results contain many groups of the results captured by performing all of the methods to be compared on all of the datasets.

### 3.4.1 Performance Comparison between IPECs and the state-of-the-art methods

The experimental results and the comparison between the proposed EC algorithms and the state-of-the-art (SOA) methods are shown in Table 3.4. In order to clearly show the comparison results, the terms (+) and (-) are provided to indicate the result of proposed method is better or worse than the best result obtained by the corresponding peer competitor; The term (=) shows that the mean error rate of the proposed method are slightly better or worse than the competitor, but the difference is not significant from the statistical point of view; The term – means there are no available results reported or cannot be counted. As there are results from 5 experiments compared to the state-of-the-art methods, 5 mathematical operators in the parentheses to represent the comparison conclusions of IPDE-10%-5, IPGA-10%-5, IPPSO-10%-5, IPRS-10%-5 and IPDE-100%-5, respectively from left to right.

**IPDE-10%-5 vs. SOA methods**: By comparing IPDE-10%-5 with the state-of-the-arts methods using One Sample T-Test to determine whether the results are better than a mean value from the statistical point of view, it can be observed that IPDE achieves encouraging performance in terms of error rates shown in Table 3.4. To be specific, on the CONVEX benchmark dataset, IPDE-10%-5 outperforms five of the nine state-of-the-art methods; on the MB benchmark, the IPDE-10%-5 attains a better error rate than six of the ten state-of-the-art methods; on the MBI benchmark, IPDE-10%-5 beats all of the state-of-the-art methods; on the MDRBI benchmark, although the mean error rate of IPDE-10%-5 is less than 38.54% achieved by LDANet-2, the P-value of One Sample T-Test is 0.4692 which indicates that the difference between the results of IPDE-10%-5 with 38.54 is not statistically significant, so IPDE-10%-5 can be deemed equivalent to LDANet-2 ranking the third among the twelve state-of-the-art methods; for MRB benchmark, the mean error rate of IPDE-10%-5 is better than 6.08% achieved by PGBM+DN-1, but the P-value of 0.5985 between the error rate of IPDE-10%-5 and 6.08% does not prove the significance of the difference, so IPDE-10%-5 can be expected to perform just as good as PGBM+DN-1; for MRD benchmark, IPDE-10%-5 outruns the state-of-the-arts method apart from TIRBM.

Table 3.4: The classification errors of IPDE-10%-5, IPGA-10%-5, IPPSO-10%-5, IPRS-10%-5 and IPDE-100%-5 against the peer competitors

| classier | CONVEX | | MB | | MBI | | MDRBI | | MRB | | MRD | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CAE-2 | | – | 2.48 | (+ + + + +) | 15.50 | (+ + + + +) | 45.23 | (+ + + + +) | 10.90 | (+ + + + +) | 9.66 | (+ + + + +) |
| TIRBM | | – | | – | | – | 35.50 | (- - - = +) | | – | 4.20 | (- - - - -) |
| PGBM+DN-1 | | – | | – | 12.15 | (+ + + + +) | 36.76 | (- - - = +) | 6.08 | (= = = = +) | | – |
| ScatNet-2 | 6.50 | (- - - - -) | 1.27 | (- - - - -) | 18.40 | (+ + + + +) | 50.48 | (+ + + + +) | 12.30 | (+ + + + +) | 7.48 | (+ + + + +) |
| RandNet-2 | 5.45 | (- - - - -) | 1.25 | (- - - - -) | 11.65 | (+ + + + +) | 43.69 | (+ + + + +) | 13.47 | (+ + + + +) | 8.47 | (+ + + + +) |
| PCANet-2 (softmax) | 4.19 | (- - - - -) | 1.40 | (- = - - -) | 11.55 | (+ + + + +) | 35.86 | (- - - = +) | 6.85 | (+ = + + +) | 8.52 | (+ + + + +) |
| LDANet-2 | 7.22 | (- - - - -) | 1.05 | (- - - - -) | 12.42 | (+ + + + +) | 38.54 | (= = = =) | 6.81 | (+ = = + +) | 7.52 | (+ + + + +) |
| SVM+RBF | 19.13 | (+ + + + +) | 30.03 | (+ + + + +) | 22.61 | (+ + + + +) | 55.18 | (+ + + + +) | 14.58 | (+ + + + +) | 11.11 | (+ + + + +) |
| SVM+Poly | 19.82 | (+ + + + +) | 3.69 | (+ + + + +) | 24.01 | (+ + + + +) | 54.41 | (+ + + + +) | 16.62 | (+ + + + +) | 15.42 | (+ + + + +) |
| NNet | 32.25 | (+ + + + +) | 4.69 | (+ + + + +) | 27.41 | (+ + + + +) | 62.16 | (+ + + + +) | 20.04 | (+ + + + +) | 18.11 | (+ + + + +) |
| SAA-3 | 18.41 | (+ + + + +) | 3.46 | (+ + + + +) | 23 | (+ + + + +) | 51.93 | (+ + + + +) | 11.28 | (+ + + + +) | 10.30 | (+ + + + +) |
| DBN-3 | 18.63 | (+ + + + +) | 3.11 | (+ + + + +) | 16.31 | (+ + + + +) | 47.39 | (+ + + + +) | 6.73 | (+ = = + +) | 10.30 | (+ + + + +) |
| IPDE-10%-5(best) | 8.56 | | 1.16 | | 6.63 | | 32.20 | | 3.88 | | 3.84 | |
| IPDE-10%-5(mean) | 11.65 | | 1.47 | | 10.30 | | 39.33 | | 5.89 | | 5.81 | |
| IPDE-10%-5(standard deviation) | 1.94 | | 0.15 | | 1.58 | | 5.87 | | 1.97 | | 1.17 | |
| IPGA-10%-5(best) | 7.61 | | 1.08 | | 6.61 | | 31.25 | | 3.93 | | 4.62 | |
| IPGA-10%-5(mean) | 11.57 | | 1.46 | | 10.09 | | 38.84 | | 6.37 | | 5.69 | |
| IPGA-10%-5(standard deviation) | 2.10 | | 0.17 | | 2.15 | | 5.33 | | 1.60 | | 0.66 | |
| IPPSO-10%-5(best) | 8.75 | | 1.22 | | 5.91 | | 30.42 | | 3.27 | | 4.62 | |
| IPPSO-10%-5(mean) | 12.65 | | 1.56 | | 9.86 | | 38.79 | | 6.26 | | 6.07 | |
| IPPSO-10%-5(standard deviation) | 2.13 | | 0.17 | | 1.84 | | 5.38 | | 1.54 | | 0.71 | |
| IPRS-10%-5(best) | 8.93 | | 1.01 | | 6.36 | | 26.88 | | 3.73 | | 3.78 | |
| IPRS-10%-5(mean) | 12.60 | | 1.51 | | 9.86 | | 36.94 | | 5.99 | | 5.54 | |
| IPRS-10%-5(standard deviation) | 2.20 | | 0.15 | | 2.03 | | 4.67 | | 1.35 | | 0.68 | |
| IPDE-100%-5(best) | 7.25 | | 1.23 | | 5.45 | | 26.83 | | 3.54 | | 4.45 | |
| IPDE-100%-5(mean) | 11.62 | | 1.57 | | 7.54 | | 32.07 | | 4.88 | | 5.91 | |
| IPDE-100%-5(standard deviation) | 3.87 | | 0.18 | | 1.03 | | 2.65 | | 0.69 | | 1.34 | |

**IPGA-10%-5 vs. SOA methods**: It can be discovered that IPGA-10%-5 excels by examining the error rates of IPGA-10%-5 and its competitors listed in Table 3.4. More specifically, it obtains the same rankings as IPGA-10%-5 with the fifth, the first, tied for fourth place and the second on the CONVEX, MBI, MDRBI and MRD benchmark datasets, respectively;

for the MB benchmark, the P-value calculated by One Sample T-Test between the error rate and 1.4% acheived by PCANet-2 (softmax) is 0.0668, so the significance of the difference is not satisfied, which implies the IPGA-10%-5 ties PCANet-2 (softmax) at the fourth place among the listed methods; for the MRB benchmark, the P-values of 0.3270 and 0.1338 are received from One Sample T-test of 6.08% and 6.85%, respectively, which implies that there is no significant difference by comparing the error rate of IPGA-10%-5 with the best four state-of-the-art algorithms, so IPGA-10%-5 could be claimed as one of the best solutions on the MRB benchmark.

**IPPSO-10%-5 vs. SOA methods**: As shown in Table 3.4, IPPSO-10%-5 achieves promising results across all of the six benchmark datasets. To be detailed, for the CONVEX, MBI, MDRBI and MRD benchmark datasets, it stays in the same positions exactly as IPDE-10%-5 and IPGA-10%-5; for the MB benchmark, it is defeated by four of the state-of-the-art methods, which is the same as IPDE-10%-5; for the MRB benchmark, the P-value of 0.5388 argues that IPPSO-10%-5 does not perform worse than the best state-of-the-art method named PGBM+DN-1, even though the mean error rate of 6.26% is a bit worse than 6.08% of PGBM+DN-1, and the P-values of 0.0585 and 0.0434 calculated by applying One Sample T-test against 6.81% and 6.85%, respectively, indicates that IPPSO-10%-5 does not outperform LDANet-2, but it is superior to PCANet-2 (softmax).

**IPDE-100%-5 vs. SOA methods**: From the results listed in Table 3.4, overall, IPDE-100%-5 accomplishes a more promising performance than IPDE-10%-5 by being compared with the state-of-the-art methods. It ranks the fifth, the fifth, the first and the second for the benchmark datasets of CONVEX, MB, MBI and MRD, respectively, which is the same as IPDE-10%-5; However, it performs the best on both the MDRBI and MRB benchmark datasets, which outperforms IPDE-10%-5.

### 3.4.2 Performance Comparison between IPECs and IPRS

**IPRS-10%-5 vs. SOA methods**: IPRS-10%-5 can be proved as a strong competitor by inspecting Table 3.4. Considering the error rates on the CONVEX, MBI and MRD datasets, it accomplishes the same rankings as IPDE-10%-5, IPGA-10%-5, and IPPSO-10%-5; in terms of the performance on MB, it possesses the fifth ranking position which is the same as that of IPDE-10%-5 and IPPSO-10%-5; regarding the MRB benchmark, the same as IPDE-10%-5, it takes the tied for first place with PGBM+DN-1, which is supported by the P-value of 0.7112 and outperforms the other state-of-the-art methods; with regard to the MDRBI benchmark, it achieves an error rate no better than 38.54% of LDANet-2 and no worse than TIRBM upheld by the P-values of 0.1020 and 0.0706 collected by One Sample T-Test agaist 35.5% and 38.54%, respectively, which means it ties the best four state-of-the-art methods for first place and performs better than the others.

**IPDE-10%-5 vs. IPRS-10%-5**: In terms of the error rate, none of the test results of Two Sample T-Test between IPDE-10%-5 and IPRS-10%-5 across all of the six benchmark datasets can prove any significant difference between these two methods, which can be seen in Table 3.5. With regard to the generation which reflects the computational cost, according to the results of the MWW test, neither of these two methods outstands.

**IPGA-10%-5 vs. IPRS-10%-5**: By analysing the Two Sample T-Test results on the error rate and the MWW test result on the generation listed in Table 3.5, from the statistical point of view, it can be concluded that IPGA-10%-5 obtains the similar performance as that of IPRS-10%-5 both in terms of the accuracy and the computational cost.

**IPPSO-10%-5 vs. IPRS-10%-5**: In Table 3.5, one P-value on the first row and one P-value on the second row indicate that the significant difference is supported. The mean error rates of these two methods can be found in Table 3.4, and the mean error rate of IPRS-10%-5 on

Table 3.5: Two Sample T-Test between a pair of IPDE-10%-5, IPGA-10%-5, IPPSO-10%-5 and IPRS-10%-5

| | CONVEX | MB | MBI | MDRBI | MRB | MRD |
|---|---|---|---|---|---|---|
| IPDE-10%-5 and IPRS-10%-5 | | | | | | |
| Error Rate | 0.08 | 0.35 | 0.35 | 0.09 | 0.82 | 0.29 |
| Generation | 0.73 | 0.12 | 0.73 | 0.58 | 0.58 | 0.79 |
| IPGA-10%-5 and IPRS-10%-5 | | | | | | |
| Error Rate | 0.08 | 0.29 | 0.68 | 0.17 | 0.33 | 0.42 |
| Generation | 0.66 | 0.46 | 0.88 | 0.79 | 0.21 | 0.92 |
| IPPSO-10%-5 and IPRS-10%-5 | | | | | | |
| Error Rate | 0.94 | 0.21 | 1.00 | 0.16 | 0.48 | **0.0047** |
| Generation | 0.23 | 0.06 | 0.66 | 0.69 | **0.04236** | 0.17 |
| IPDE-10%-5 and IPGA-10%-5 | | | | | | |
| Error Rate | 0.88 | 0.82 | 0.67 | 0.75 | 0.32 | 0.66 |
| Generation | 0.43 | **0.02574** | 0.91 | 0.74 | 0.60 | 0.67 |
| IPDE-10%-5 and IPPSO-10%-5 | | | | | | |
| Error Rate | 0.06 | **0.0359** | 0.32 | 0.71 | 0.43 | 0.29 |
| Generation | 0.29 | **0.00086** | 0.28 | 0.25 | **0.02088** | **0.03486** |
| IPGA-10%-5 and IPPSO-10%-5 | | | | | | |
| Error Rate | 0.06 | **0.0416** | 0.67 | 0.98 | 0.78 | **0.0468** |
| Generation | 0.07 | 0.24 | 0.51 | 0.40 | **0.0035** | 0.24 |

the MRD benchmark dataset is less than that of IPPSO-10%-5, so IPRS-10%-5 outperforms IPPSO-10%-5 on the MRD benchmark in terms of the error rate; However, the mean generation of IPPSO-10%-5 on the MRB benchmark dataset is 1.40 which is much less than 2.07 achieved by IPRS-10%-5 on the MRB benchmark, so IPPSO-10%-5 spends less computational cost than IPRS-10%-5 on the MRB benchmark.

### 3.4.3 Performance Comparison between IPECs

**IPDE-10%-5 vs. IPGA-10%-5**: Among all of the P-values in Table 3.5, 0.02574 in bold is the only one that shows a significant difference. By comparing 2.43 - the mean generation of IPDE-10%-5 on the MB benchmark dataset with 1.59 - the corresponding mean generation of IPGA-10%-5, IPGA-10%-5 attains better efficiency in terms of computational cost; while, neither of these two methods defeats each other on the MB benchmark in terms of the classification accuracy. In regard to the other five benchmark datasets, both of them gain a similar performance both in regard to the error rate and the generation used to attain the optimised CNN architecture.

**IPDE-10%-5 vs. IPPSO-10%-5**: In the generation row of Table 3.5, three of the P-values on the MB, MRB and MRD columns are less than 0.05 - the confidence level. The mean generation values of IPDE-10%-5 on the MB, MRB and MRD are 2.43, 2.27 and 2.30, and the corresponding mean generation values of IPPSO-10%-5 are 1.17, 1.40 and 1.63, so IPPSO-10%-5 is more efficient than IPDE-10%-5 on these three benchmarks. Among the above three benchmarks, for the MB dataset, IPPSO-10%-5 accomplishes a less error rate than that of IPDE-10%-5, which can be retrieved from Table 3.4, so from the statistical point of view, IPDE-10%-5 excels on the MB benchmark in terms of the classification accuracy; However, for the other two benchmark among the MB, MRB and MRD, the difference between these two methods is not significant. Apart from the aforementioned three benchmarks, neither of IPDE-10%-5 or IPPSO-10%-5 exceeds each other.

**IPGA-10%-5 vs. IPPSO-10%-5**: In Table 3.5, the significant P-values in bold appear in the columns of MB, MRB and MRD which are the same columns as those of the comparison between IPDE-10%-5 and IPPSO-10%-5. For the MB and MRD benchmark datasets, as the mean error rates of IPGA-10%-5 are less than the corresponding mean error rates of IPPSO-10%-5, which can we observed in Table 3.4, IPGA-10%-5 demonstrates its superiority to IPPSO-10%-5 in terms of the classification accuracy on the MB and MRD benchmark datasets. In addition, by checking the P-values of the generation on MB and MRD columns in Table 3.5, IPGA-10%-5 does not compromise the computational cost to gain the better accuracy on these two datasets. For the MRB benchmark, the bold 0.0035 shows the significant difference between the generation samples of these two methods, and the mean generations of IPGA-10%-5 and IPPSO-10%-5 are 2.47 and 1.40, respectively, so the generation used to obtain the best CNN architecture on the MB dataset by IPGA-10%-5 is larger than that used by IPPSO-10%-5, which means there is more computational cost needed for IPGA-10%-5 given the MB benchmark. Apart from these three significant P-values in Table 3.5, neither of these two methods outstrips each other from the statistical point of view.

### 3.4.4 Performance Comparison by tuning the hyperparameters of fitness evaluation

The experiments of fine-tuning the percentage of dataset used for the evolutionary process among 10%, 40%, 70% and 100%, and the training epochs during fitness evaluation among 5, 10 and 15 are done. The initial trend of how the hyperparameters affect the classification accuracy of IPDE has been obtained, and the details can be found in Appendix A.3.

## 3.5 Conclusions

It can be concluded that the proposed EC methods including IPDE and IPGA along with the previous IPPSO and the IP-Based Random Search method have obtained promising results on the benchmark datasets by comparing them to the state-of-the-art algorithms. It is also observed that the IP-Based EC methods do not outperform the IP-Based Random Search algorithm, and none of them significantly excels in terms of the accuracy and computational cost on the given dataset by applying statistical tests on their results. However, the limited complexity of the benchmark datasets might not tell the performance differences between these methods because it is easy to achieve a relatively hight accuracy for easy classification tasks. In addition, there is another limitation of the proposed EC methods, which is that the maximum depth of the architectures of CNNs has to be set before the commencement of the evolutionary process, so any CNN architectures that exceed the maximum depth are not explored by the EC methods. For simple tasks, setting a maximum depth is straightforward, e.g. a maximum depth of 10 layers for the above experiments, but for some extremely complicated tasks, it would be hard to set a maximum length of the architectures of CNNs, as the maximum depth is not known, and using an over large depth may result in terrible computational cost, while setting a small number as the depth may not be able to acquire an optimal CNN architecture.

# Chapter 4

# The Proposed Hybrid DE Approach

## 4.1 Introduction

The IP-Based Encoding Strategy (IPES) [40] has demonstrated its powerfulness for evolving deep CNNs in the first part of this project, but it has a critical drawback which is that the maximum depth of the CNN architectures has to be set before the commencement of the evolutionary process. Therefore, the encoding strategy is refined in the proposed algorithm to break the constraint of the predefined maximum length.

DE has been chosen as the base algorithm because of a couple of reasons. The first reason is that DE has been proved as an efficient algorithm to evolve CNN architectures in Chapter 3. Other than that, the more important reason is that DE has some kind of local search by selecting the best individual between the parent and the child after mutation and crossover, and this kind of local search is used for evolving the length of the CNN architecture because the length of CNN architectures represented by the parent and the child, which achieved better fitness, will be used as an individual in the next generation.

**Goals:** The goal of this part is to propose a new DE method for automatically evolving the structures and parameters of deep CNNs without restricting the maximum-length, which will be achieved by removing the disabled layer when encoding CNN architectures, new mutation and crossover operators of DE, and a second crossover operator. The proposed method named DECNN will be examined and compared with 12 state-of-the-art methods on six widely-used datasets of varying difficulty. The specific objectives are

- refine the existing effective encoding scheme used by IPPSO [40] to break the constraint of predefining the maximum depth of CNNs by removing the disabled layer;
- design and develop new mutation and crossover operators for the proposed DECNN method, which can be applied on variable-length vectors to conquer the fixed-length limitation of the traditional DE method;
- design and integrate a second crossover operator into the proposed DECNN to produce the children in the next generation representing the architectures of CNNs whose lengths differ from their parents.

## 4.2 The Proposed Algorithm

As mentioned in Section 3.5, there is a limitation of the maximum length of the architectures of CNNs because the traditional methods of PSO, DE and GA are used to explore a search space with a fixed dimension. Even though by introducing the Disabled Layer in the encoding strategy, the aforementioned algorithms are able to evolve the variable-length architectures of CNNs to some extent, it would be better to develop a method without any constraints in terms of the maximum length of the architectures of CNNs that the algorithm is capable to learn. The proposed hybrid DE approach uses DE as the main evolutionary algorithm, and a second crossover operator is proposed to generate children whose lengths differ from their parents to fulfil the requirement of evolving variable-length architectures of CNNs.

### 4.2.1 DECNN Algorithm Overview

The overall procedure of the proposed DECNN algorithm is written in Algorithm 6. The pivotal part of DECNN is the second crossover described in Section 4.2.5, which produces two children representing variable-length CNNs. By selecting the best individual between the two children and their parent, a sort of local search is performed to search for the CNN architecture with a specific length that tend to achieve better fitness.

**Algorithm 6:** Framework of IPDE

```
$P \leftarrow$ Initialise the population elaborated in Section 4.2.2;
$P_{best} \leftarrow empty$;
while termination criterion is not satisfied do
    Apply the refined DE mutation and crossover described in Section 4.2.4;
    Apply the proposed second crossover to produce two children, and select the best
    between the two children and their parents illustrated in Section 4.2.5;
    evaluate the fitness value of each individual;
    $P_{best} \leftarrow$ retrieve the best individual in the population;
end while
```

### 4.2.2 Population Initialisation

As the individuals are required to be in different lengths, the population initialisation starts by randomly generating the lengths of individuals. In the proposed DECNN, the length is randomly sampled from a Gaussian distribution with a standard deviation $\rho$ of 1 and a centre $\mu$ of a predefined length depending on the complexity of the classification task as shown in Equation (4.1). After obtaining the candidate's length, the layer type and the attribute values can be randomly generated for each layer in the candidate. By repeating the process until reaching the population size to accomplish the population initialisation.

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2 / 2\sigma^2} \tag{4.1}$$

### 4.2.3 Fitness Evaluation

The fitness evaluation process is illustrated in Algorithm 7. First of all, four arguments are taken in by the fitness evaluation function, which are the candidate solution which represents an encoded CNN architecture, the training epoch number for training the model decoded from the candidate solution, the training set which is used to train the decoded CNN architecture, and the fitness evaluation dataset on which the trained model is tested to obtain the accuracy used as the fitness value. Secondly, the fitness evaluation process is pretty straightforward by using the back propagation to train the decoded CNN architecture on the training set for a fixed number of epochs, and then obtaining the accuracy on the fitness evaluation set, which is actually used as the fitness value. For the purpose of reducing computational cost, the candidate CNN is only trained on a partial dataset for a limited number of epochs, which are controlled by the arguments of the fitness function - $k$, $D\_train$ and $D\_fitness$.

```
Algorithm 7: Fitness Evaluation
  Input: The candidate solution $c$, the training epoch number $k$, the training set $D\_train$, the
    fitness evaluation dataset $D\_fitness$;
  Output: The fitness value $fitness$;
    Train the connection weights of the CNN represented by the candidate $c$ on the training
    set $D\_train$ for $k$ epochs;
    $acc \leftarrow$ Evaluate the trained model on the fitness evaluation dataset $D\_fitness$
    $fitness \leftarrow acc$;
    return $fitness$
```

### 4.2.4 DECNN DE Mutation and Crossover

The proposed DECNN operations are similar to the standard DE mutation and crossover as described in Section 2.3.1, but it introduces an extra step to trim the longer vectors before applying any operation because the DECNN candidates have different lengths and the traditional DE operations in Equation (2.1) and (2.2) only apply on fixed-length vectors. To be specific, the three random vectors for the mutation are trimmed to the shortest length of them, and during the crossover, if the trial vector generated by the mutation is longer than the parent, it will be trimmed to the length of the parent. The details are described in Algorithm 8.

```
Algorithm 8: DECNN Mutation and Crossover
  Input: $i_{th}$ parent $x_i$, population $P$;
  Output: child $u_i$;
    $v_i \leftarrow$ Empty candidate
    $x_{r0}, x_{r1}, x_{r2} \leftarrow$ Randomly select three unique individuals from the population $P$;
    $x_{r1}, x_{r2} \leftarrow$ Find the shorter individual between $x_{r1}, x_{r2}$ and randomly slice the other
    individual to the same length as that of the shorter one;
    $v_i \leftarrow$ Perform the latter part of DE/rand/1 mutation $F \times (\mathbf{x}_{r1,g} - \mathbf{x}_{r2,g})$ on $x_{r1}, x_{r2}$;
    $x_{r0}, v_i \leftarrow$ Find the shorter individual between $x_{r0}, v_i$ and randomly slice the other
    individual to the same length as that of the shorter one;
    $v_i \leftarrow$ Perform the first part of DE/rand/1 mutation $\mathbf{x}_{r0,g} + bytes\_vector\_candidate_g$ on
    $x_{r0}, v_i$;
    $x_i, v_i \leftarrow$ Find the shorter individual between $x_i, v_i$ and randomly slice the other
    individual to the same length as that of the shorter one;
    $u_i \leftarrow$ Perform crossover on $x_i$ and $v_i$;
    return $u_i$
```

### 4.2.5 DECNN second crossover

Similar as the crossover of GAs, each individual of the two parents is split into two parts by slicing the vector at the cutting points, and swap one part with each other. The cutting point is chosen by randomly finding a position based on Gaussian distribution with the middle point as the centre and a hyperparameter $\rho$ as the standard deviation to control the variety in the population. The flow of the second crossover is outlined in Fig. 4.1.

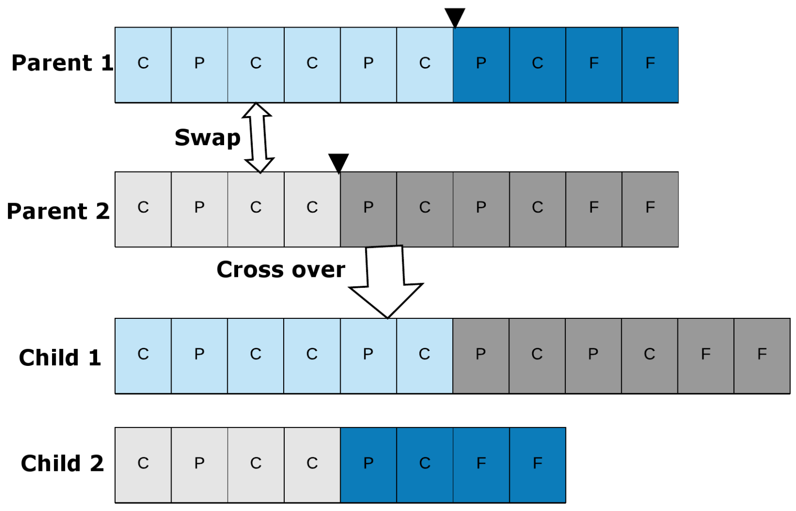


Figure 4.1: second crossover of the proposed DECNN algorithm

## 4.3 Experiment Design

### 4.3.1 Benchmark Datasets and State-of-the-art Competitors

In order to perform a fair comparison between DECNN and the proposed EC algorithms in the first part of this project, the same benchmark datasets and the same peer competitors are utilised, which are described in Section 3.3.1 and 3.3.2, respectively.

### 4.3.2 Parameter settings of the proposed EC methods

All of the parameters are configured according to the conventions in the communities of DE [8] along with taking into account a small population to safe computation time and the complexity of the search space. For the evolutionary process, 30 is set as the population size and 20 is used as the number of generations; In regard to the fitness evaluation, the number of training epochs is set to 5 and 10% of the training dataset is passed for evaluation; In terms of the DE parameters, 0.6 and 0.45 are used as the differential rate and the crossover rate, respectively; The hyperparameter $\rho$ of second crossover is set to 2, and $\mu$ of the population initialisation is set to 10; 30 independent runs is performed by the proposed DECNN on each of the benchmark dataset.

## 4.4 Results and Discussions

Since DE is stochastic, statistical significance test is required to make the comparison result more convincing. When comparing the proposed DECNN with the state-of-the-art methods, One Sample T-Test is applied to test whether the results of DECNN is better; when the comparison of error rates between DECNN and the peer EC competitor named IPPSO [40] is performed, Two Sample T-test is utilised to determine whether the difference is statistically significant or not. Table 4.1 shows the comparison results between the proposed DECNN and the state-of-the-art algorithms; Table 4.2 compares DECNN with IPPSO.

### 4.4.1 DECNN vs. State-of-the-Art methods

The experimental results and the comparison between the proposed DECNN and the state-of-the-art methods are shown in Table 4.1. In order to clearly show the comparison results, the terms (+) and (-) are provided to indicate the result of DECNN is better or worse than the best result obtained by the corresponding peer competitor; The term (=) shows that the mean error rate of DECNN are slightly better or worse than the competitor, but the difference is not significant from the statistical point of view; The term – means there are no available results reported from the provider or cannot be counted.

It can be observed that the proposed DECNN method achieves encouraging performance in terms of the error rates shown in Table 4.1. To be specific, the proposed DECNN ranks the fifth on both the CONVEX and MB benchmark datasets; for the MBI benchmark, DECNN beats all of the state-of-the-art methods; for the MDRBI dataset, the mean error rate of DECNN is the fourth best, but the P-value of One Sample T-Test between DECNN and the third best is 0.0871, which indicates that the significance difference is not supported from the statistical point of view, so DECNN ties the third with PGBM+DN-1; for the MRB benchmark, the mean error rate of DECNN is smaller than all other methods, but the difference between DECNN and the second best algorithm is not significant given the calculated P-value of 0.1053, so DECNN ties the first with PGBM+DN-1; for the MRD benchmark, DECNN outruns the state-of-the-arts method apart from TIRBM. In addition, by looking at the best results of DECNN, DECNN achieves the smallest error rates on five out of the six datasets compared with the 12 state-of-the-art methods, which are 1.03% on MB, 5.67% on MBI, 32.85% on MDRBI, 3.46% on MRB and 4.07% on MRD. This shows that DECNN has the potential to improve the state-of-the-art results.

Table 4.1: The classification errors of DECNN against the peer competitors

| classier | CONVEX | | MB | | MBI | | MDRBI | | MRB | | MRD | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CAE-2 | | – | 2.48 | (+) | 15.50 | (+) | 45.23 | (+) | 10.90 | (+) | 9.66 | (+) |
| TIRBM | | – | | – | | – | 35.50 | (-) | | – | 4.20 | (-) |
| PGBM+DN-1 | | – | | – | 12.15 | (+) | 36.76 | (=) | 6.08 | (=) | | – |
| ScatNet-2 | 6.50 | (-) | 1.27 | (-) | 18.40 | (+) | 50.48 | (+) | 12.30 | (+) | 7.48 | (+) |
| RandNet-2 | 5.45 | (-) | 1.25 | (-) | 11.65 | (+) | 43.69 | (+) | 13.47 | (+) | 8.47 | (+) |
| PCANet-2 (softmax) | 4.19 | (-) | 1.40 | (-) | 11.55 | (+) | 35.86 | (-) | 6.85 | (+) | 8.52 | (+) |
| LDANet-2 | 7.22 | (-) | 1.05 | (-) | 12.42 | (+) | 38.54 | (+) | 6.81 | (+) | 7.52 | (+) |
| SVM+RBF | 19.13 | (+) | 30.03 | (+) | 22.61 | (+) | 55.18 | (+) | 14.58 | (+) | 11.11 | (+) |
| SVM+Poly | 19.82 | (+) | 3.69 | (+) | 24.01 | (+) | 54.41 | (+) | 16.62 | (+) | 15.42 | (+) |
| NNet | 32.25 | (+) | 4.69 | (+) | 27.41 | (+) | 62.16 | (+) | 20.04 | (+) | 18.11 | (+) |
| SAA-3 | 18.41 | (+) | 3.46 | (+) | 23 | (+) | 51.93 | (+) | 11.28 | (+) | 10.30 | (+) |
| DBN-3 | 18.63 | (+) | 3.11 | (+) | 16.31 | (+) | 47.39 | (+) | 6.73 | (+) | 10.30 | (+) |
| DECNN(best) | 7.99 | | 1.03 | | 5.67 | | 32.85 | | 3.46 | | 4.07 | |
| DECNN(mean) | 11.19 | | 1.46 | | 8.69 | | 37.55 | | 5.56 | | 5.53 | |
| DECNN(standard deviation) | 1.94 | | 0.11 | | 1.41 | | 2.45 | | 1.71 | | 0.45 | |

### 4.4.2 DECNN vs. IPPSO

As none of IPPSO, IPDE and IPGA outperforms the other in first part of this project from statistical point of view, IPPSO is used as a representative of the three EC algorithms, which is compared with the proposed DECNN. In Table 4.2, it can be observed that the mean error rates of DECNN are smaller across all of the six benchmark datasets, and the standard deviations of DECNN is less than those of IPPSO on five datasets out of the six, so the overall performance of DECNN is superior to IPPSO. The second crossover operator improves the performance of DECNN because it performs a kind of local search between the two children and their parents both in terms of the depth of CNN architectures and their parameters.

Table 4.2: Classification rates of DECNN and IPPSO

| | CONVEX | MB | MBI | MDRBI | MRB | MRD |
|---|---|---|---|---|---|---|
| DECNN(mean) | 11.19 | 1.46 | 8.69 | 37.55 | 5.56 | 5.53 |
| DECNN(standard deviation) | 1.94 | 0.11 | 1.41 | 2.45 | 1.71 | 0.45 |
| IPPSO(mean) | 12.65 | 1.56 | 9.86 | 38.79 | 6.26 | 6.07 |
| IPPSO(standard deviation) | 2.13 | 0.17 | 1.84 | 5.38 | 1.54 | 0.71 |
| P-value | **0.01** | **0.01** | **0.01** | 0.26 | 0.10 | **0.001** |

### 4.4.3 Evolved CNN Architectures

After examining the evolved CNN architectures, it is found that DECNN demonstrates its capability of evolving the length of the architectures. When the evolutionary process starts, the lengths of individuals are around 10; while the lengths of evolved CNN architectures drop to 3 to 5 depending on the complexity of the datasets, which proves that DECNN has the ability of effectively evolving CNN architectures of various lengths. Table 4.3 shows an example of the evolved CNN architecture.

Table 4.3: An example of the evolved architectures on MB dataset

| Layer type | Configuration |
|---|---|
| convolutional | Filter size: 2, Stride size: 1, feature maps: 23 |
| convolutional | Filter size: 4, Stride size: 2, feature maps: 49 |
| full | Neurons: 1583 |
| full | Neurons: 10 |

## 4.5 Conclusions

The goal of this part is to develop a novel DE-based algorithm to automatically evolve the architecture of CNNs for image classification without any constraint of the depth of CNN architectures. This has been accomplished by designing and developing the proposed hybrid differential evolution method. More specifically, three major contributions are made by the proposed DECNN algorithm. First of all, the IP-Based Encoding Strategy has been improved by removing the maximum length of the encoded vector and the unnecessary disabled layer in order to achieve a real variable-length vector of any length; Secondly, the new DE operations - mutation, crossover are developed, which can be applied to candidate vectors of variable lengths; Last but not least, a novel second crossover is designed and added to DE to produce children having different lengths from their parents. The second crossover plays an important role to search the optimal depth of the CNN architectures because the two children created through the second crossover have different length from their parents - one is longer and the other is shorter, and during the selection from the two children and the two parents, the candidate with a better fitness survives to the next generation, which indicates that the length of the remaining candidate tends to be better than the other three.

The proposed DECNN has achieved encouraging performance. By comparing the performance of DECNN with the 12 state-of-the-art competitors on the six benchmark datasets, it can be observed that DECNN obtains a very competitive accuracy by ranking the first on the MBI and MRB datasets, the second and the third on the MRD and MDRBI datasets, respectively, and the fifth on the MB and CONVEX datasets. In a further comparison with the peer EC competitor, the best results are achieved by DECNN on five out of the six datasets.

However, it can be observed that there are still some room to improve the performance in terms of the accuracy, especially on MB, CONVEX and MDRBI benchmark datasets as there are a few peer competitors outperforming DECNN on these three datasets, so in the final part of the project, more advanced CNN architectures are evolved to boost the accuracy on these three datasets.

# Chapter 5

# The Proposed Hybrid Two-level EC Method

## 5.1 Introduction

DECNN proposed in Chapter 4 has achieved a promising results across all of the six benchmark datasets, but the performance is not ideal because it does not outperform all of the 12 peer competitors. Especially for the CONVEX benchmark dataset, four of nine peer competitors obtain better accuracy than that of DECNN, which means there is still room to explore to improve the performance of using EC methods to evolve CNN architectures.

In recent years, deep CNNs have achieved better and better accuracy on image classification tasks, and the architectures of CNNs grow deeper and deeper, which on the other hand, brings up the optimisation difficulty. Fortunately, additional connections added to connect the current layer and the forward layers apart from the next layer have been discovered and proved to be effective to conquer the optimisation difficulty, which therefore improves the performance of the deep CNNs, e.g. ResNet [9] shown in Fig. 5.1 and DenseNet [10] illustrated in Fig. 5.2. From the figures of ResNet and DenseNet, it can be seen that in ResNet, along with the direct forward connections between the current layer and the next layer, there are jump connections, which connect the current layer to the layer after the next layer; in DenseNet, it divides the CNN architecture into a number of blocks, and each layer is connected to all of the forward layers, which is called densely-connected structure. The intuition behind the jump connections in ResNet and the densely-connected structure in DenseNet is to pass the useful features learned from the current layer to the future layers. However, the features learned from the current layer might not be useful, so it might introduce some noises by passing them to the future layers. In this part of the project, instead of just using the human-designed connections between the current layer and the forward layers as shown in DenseNet and ResNet, a hybrid algorithm is proposed to evolve the structure of the CNN which defines the layers of the CNN architecture, and also to decide whether the connection exists between the current layer and the forward layers apart from the next layer because the current layer is always connected to the next layer. This kind of connection is called the shortcut connection. As shortcut connections of the evolved CNN architectures may be densely-connected or sparsely-connected, the evolved CNN architecture is called Dynamically-connected Network (DynamicNet).

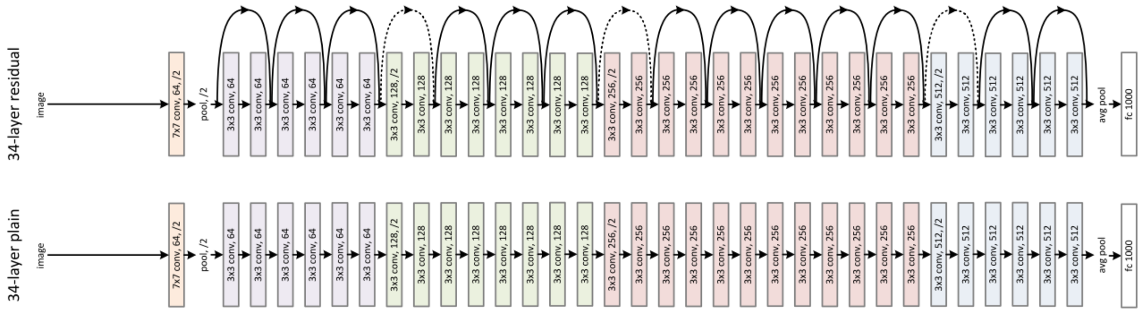


Figure 5.1: ResNet architecture [9]

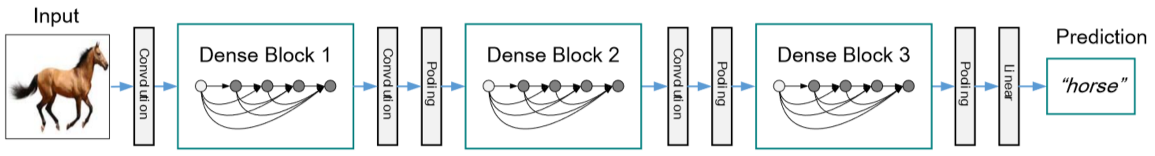


Figure 5.2: DenseNet architecture [10]

**Goals:** The goal of this chapter is to propose a hybrid algorithm with two-level evolution using GA and PSO (HGAPSO) to evolve more advanced and deeper CNN architectures called DynamicNet in order to further improve the classification accuracy from the previous EC methods. The proposed method will be examined and compared with 12 state-of-the-art methods on three of the widely-used datasets having different levels of difficulties. The specific objectives are

- Design and develop a new encoding strategy to encode the structures of CNNs and the corresponding shortcut connections. The CNN structure is encoded into a vector with decimal values, while the shortcut connections are encoded into a vector of binary values, which will be specified in Section 5.2.2;

- Desing and develop the hybrid two-level evolution algorithm. Since the CNN structure carries the capacity of the model, which is decisive to the classification accuracy, and the shortcut connections impact how well the CNN will be trained, it is reasonable to evolve the CNN structure at the first-level of evolution, and evolve the shortcut connections at the second-level based on the evolved CNN architecture. Based on the types of values of the encoded vectors, PSO is chosen to evolve the CNN architectures as the effectiveness of PSO optimising tasks with decimal values has been proven, and GA is used to evolve the shortcut connections because GA works well on optimisation tasks with binary values;

- Design and develop a new fitness evaluation method. As the individual vector representing a CNN need to be trained in order to obtain the fitness of the individual, and the number of training epochs is crucial to the computational cost of the fitness evaluation, the number of epochs is fixed to a very small number, e.g. 5. An automation method is developed to search for the best learning rate among a sequence of learning rates, which can achieve the best accuracy on DenseNet with the same number of layers of the DynamicNet that needs to be evaluated.

## 5.2 The Proposed Method

A hybrid algorithm with two-level evolution called HGAPSO is proposed here, and the details of the algorithms are illustrated in the following sub-sections. Firstly, the new CNN architecture named DynamicNet, which will be evolved by the new algorithm, is introduced, and the HGAPSO encoding strategy used to encode DynamicNet is designed as the rest of the algorithm is dependent them; Secondly, the overview of the algorithm is given; Thirdly, the first-level PSO evolution and the second-level GA evolution are described; Lastly, the fitness evaluation is defined.

### 5.2.1 DynamicNet - The Evolved CNN Architecture

As in the proposed HGAPSO algorithm, both the CNN structure and the shortcut connections are evolved, the shortcut connections of the evolved CNN architectures are not fixed, which might be densely-connected or sparsely-connected. This is why the CNN architecture is named DynamicNet.

By comparing the figures of ResNet and DenseNet, it can be observed that in ResNet, each layer has at most two connections from previous layers; however, in DenseNet, the connections from previous layers are the number of previous layers due to the densely-connected structure, so the number of input feature maps is the sum of the numbers of feature maps of all previous layers, which results in the exploding growth of the number of feature maps, particularly for the layers near the output layer. The solution introduced in DenseNet is to divide the whole CNN into multiple blocks called Dense Block as depicted in Fig. 5.2. After each block, a pair of a convolutional layer with the filter size of $3 \times 3$, the stride size of 1 and the number of feature maps of half the number of input feature maps, and a pooling layer with the kernel size of $2 \times 2$ and the stride size of 2 is added to reduce the number of feature maps to half of the number of input feature maps, and the pair is called a transition layer. As the proposed DynamicNet may be densely-connected, it might have the same exploding growth issue of the number of feature maps. Therefore, DynamicNet adopts the block mechanism of DenseNet.

Inside each block, there are a number of convolutional layers with a fixed filter size of $3 \times 3$ and a fixed stride size of 1, and all of them have the same number of feature maps, which is called the *growth rate* of the block because, after each layer, the total number of input feature maps grows by the number of feature maps of the convolutional layer. In DenseNet, *the number of blocks*, *the number of convolutional layers* and *the growth rate* are manually designed, which requires good domain knowledge and a lot of manual trials to find a good architecture. In the proposed HGAPSO algorithm, these three hyperparameters will be automatically designed.

### 5.2.2 HGAPSO Encoding Strategy

DynamicNet is comprised of a number of blocks which are connected by transition layers, and the shortcut connections are built between layers inside the block. Based on the construction pattern of the network, the hyperparameters of the architecture can be split into the structure and the shortcut connections. Regarding the structure of the network, there are various hyperparameters including the number of blocks, the number of conv layers in each block and the growth rate of the conv layer in the block, which need to be evolved. In contrast to the densely-connected structure in DenseNet, different topologies of shortcut connections both densely-connected or sparsely-connected structures, i.e. the different combination of partial shortcut connections in each block, will be explored by the proposed

HGAPSO in order to keep the meaningful features and remove the unmeaningful features learned by previous layers.

Based on the analysis of the architecture and hyperparameters, the encoding process can be divided into two steps. The first step is to encode the hyperparameters of the CNN structure. Each of the hyperparameters is a dimension of the structure encoding, which is shown in Fig. 5.3. The first dimension is the number of blocks, and the two hyperparameters of each block - the number of convolutional layers and the growth rate, as two dimensions are appended to the vector. The first step of the encoding is named the first-level encoding, which will be used by the first-level evolution . Secondly, based on the result of the first-level encoding representing a CNN structure, the shortcut connections can be encoded into a binary vector illustrated in Fig. 5.4, which is named as the second-level encoding. Fig. 5.4 shows an example of one block with 5 layers. Each of the dimension represents a shortcut connection between two layers that are not next to each other, and the two layers next to each other are always connected. Taking the first layer as an example, the three binary digits - [101] represents the shortcut connections between the first layer to the third, fourth and fifth layer, respectively, where 1 means the connection exists; while 0 means there is no connection. A number of similar binary vectors drawn in Fig. 5.4 constitute one whole vector that represents the shortcut connections of the whole block.

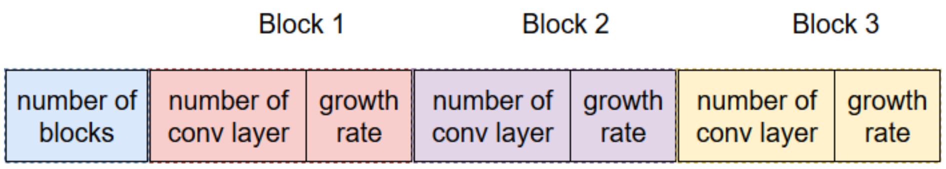


Figure 5.3: HGAPSO first-level encoding

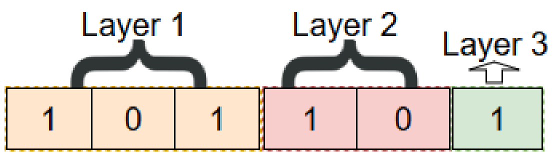


Figure 5.4: HGAPSO second-level encoding

### 5.2.3 HGAPSO Algorithm Overview

Based on the two-level encoding strategy, the algorithm is composed of two levels of evolution described in Algorithm 9. The first-level evolution is designed to evolve the structure of the CNNs encoded by the first-level encoding, and the second-level evolution is performed to search for the best combination of shortcut connections. There are a couple of reasons to separate the structure evolution from the evolution of the shortcut-connection combination. First of all, since the structure and the shortcut connections play different roles in the architectures of CNNs, which are that the structure including the depth and the width of the CNNs represents the capacity of network and the shortcut connections are able to facilitate the training process of the network, the training process is only comparable when the structure is fixed, which inspires the idea of splitting the evolution to two levels. Secondly, as shown in the above algorithms, there are quite a few types of parameters combined into the encoded vector, which brings some uncertainties to the search space, it may deteriorate the complex search space by introducing more disturbance to the search space.

It is arguable that the computational cost of the two-level evolution may be high, but the two-level encoding strategy divides the complex search space to two smaller search space and it also reduces the disturbance in the search space, so the two-level evolution we believe will not perform worse than searching for the optima in a much more complex search space. Other than that, as the second-level evolution of searching for the best combination of shortcut connections only depends on the specific structure evolved in the first-level evolution, the second-level evolution can be done in parallel for each of the individual of the first-level evolution, which can dramatically speed up the process if sufficient hardware is available.

**Algorithm 9:** Framework of HGAPSO

```
$P \leftarrow$ Initialize the population with first-level encoding elaborated in Section 5.2.2;
$P_{best} \leftarrow EmptyPSOPersonalBest$;
$G_{best} \leftarrow EmptyPSOglobalbest$;
**while** first-level termination criterion is not satisfied **do**
    $P \leftarrow$ Update the population with first-level PSO evolution described in Section 5.2.4;
    **for** particle $ind$ in population $P$ **do**
        $P\_sub \leftarrow$ Initialize the population with second-level encoding based on the value of
        $ind$ illustrated in Section 5.2.2;
        **while** second-level termination criterion is not satisfied **do**
            $P\_sub \leftarrow$ Update the population with second-level GA evolution described in
            Section 5.2.5;
            evaluate the fitness value of each individual;
            $P\_sub_{best} \leftarrow$ retrieve the best individual in $P\_sub$;
        **end while**
        Update $P_{best}$ if $P\_sub_{best}$ is better than $P_{best}$;
    **end for**
    $G_{best} \leftarrow$ retrieve the best individual in $P$;
**end while**
```

### 5.2.4 HGAPSO First-level PSO evolution

Algorithm 10 shows the pseudo code for the PSO evolution, which briefly describes the evolution process. Based on the encoded vector from the first-level encoding, the value of each dimension is a decimal value, and PSO is proved to be effective and efficient to solve the optimisation problem with decimal values, so PSO is chosen as the first-level evolution algorithm. However, the dimensionality of the encoded vector is not fixed, so an adapted variable-length PSO is proposed to solve this variable-length problem. Since the size of the input feature maps to each block is different and the specific block is trained and designed to learn meaningful features given the size of the input feature maps, when applying EC operators on two individuals, it is important to find the match blocks which have the same size of input feature maps and apply the operators on the matched blocks. To be specific with the PSO evolution in HGAPSO, the length of the particle may be different from the length of the personal best and global best, so based on the blocks of the individual, the corresponding blocks in the personal best and the global best need to be matched by selecting the blocks with the same size of the output feature and the PSO algorithm is only applied on the matched blocks

The first dimension of the vector represents the number of blocks. When the number of blocks changes, the depth of the CNN architectures changes, which obtains the ability of evolving the depth of the CNN architecture and keep the diversity of the individuals; however, the change of the number of blocks incurs a dramatic change to the CNN architecture, and if it changes too often, the CNN architecture will be disturbed over and over, which should be avoided, so it is better to leave the evolution algorithm some time to search for the other hyperparameters given the specific number of blocks. In order to keep the diversity of the number of blocks and reduce the disturbance caused by frequently changing the number of blocks, the rate of changing the number of blocks in the vector is introduced, which is a real value between 0 to 1. Therefore, the preference for diversity or stability depending on specific tasks can be controlled by tweaking the rate of changing the number of locks.

When the number of blocks is changed, some blocks need to randomly cut or randomly generated in order to meet the requirement of the number of blocks in the first dimension. For example, suppose the number of blocks is increased from 3 to 4, the hyperparameters of the fourth block need to be randomly generated based on the first-level encoding strategy, which then are appended to the vector of 3 blocks; In the other way around, assume the number of blocks is decreased from 4 to 3, the last block is removed. In the proposed HGAPSO, whenever removing blocks, it always starts from the last layer because it does not affect the feature map sizes of the other blocks.

**Algorithm 10:** HGAPSO first-level PSO evolution

```
Input: The current particle $ind$, the personal best $P_best$, the global best $G_best$, the rate of
  changing the number of blocks $r_{cb}$;
  $rnd \leftarrow$ Generate a random number from a uniform distribution
  Find the matched blocks of the particle $ind$ by comparing the feature map size;
  Update the velocity and position of the matched blocks of the particle $ind$ according to
  Equation 2.5 and 2.6;
  if $rnd < r_{cb}$ then
    Update the velocity and position of the dimension of number of blocks of the particle
    $ind$ according to Equation 2.5 and 2.6;
    Randomly cut or generate the blocks to the value of the number of blocks
  end if
```

### 5.2.5 HGAPSO Second-level GA evolution

According to the second-level encoding depicted in Section 5.2.2, once the particle is obtained from the first-level evolution, the dimensionality of the second-level encoding will be fixed, so the encoded vector can be represented by a fixed-length binary vector. Since GA is an efficient algorithm to optimise the problems that can be encoded into binary vectors, GA is chosen as the algorithm to perform the second-level evolution, which becomes a standard GA problem.

### 5.2.6 HGAPSO Fitness Evaluation

The fitness evaluation is done by using backpropagation with Adam Optimiser [17] to train the network for a number of epochs on the training part of the training data and then obtaining an accuracy of the trained network used as the fitness value on the test part of the training data. It can be observed that there are two hyperparameters for the fitness evaluation, which are the number of epochs and the initial learning rate of Adam Optimiser. In our experiment, 5 epochs are fixed by considering our hardware and a fairly-short experimental time. After the number of epochs is chosen, DenseNet is used as a benchmark to determine an initial learning rate for optimising a CNN with the given depth and width, i.e. after the structure of the CNN obtained, the network with fully-connected blocks as shown in Fig. 5.2 are used to find a best initial learning rate among 0.9, 0.1 and 0.01.

In order to speed up the evolution process, a partial dataset is used for the second-level evolution because the second-level evolution consume most the computation; while for the first-level evolution, as the computational cost is not that high, and in order to achieve a more stable performance given the structure of a CNN, the full dataset is used to achieve the fitness value of the individuals of the first-level evolution.

## 5.3 Experiment Design

### 5.3.1 Benchmark Datasets and State-of-the-art Competitors

In order to perform a fair comparison between the proposed hybrid two-level EC method and the other algorithms proposed in the first two parts of this project, the same benchmark datasets and the same peer competitors are utilised, which are described in Section 3.3.1 and 3.3.2, respectively. However, due to the computational cost and the time constraint of the final part of this project, only MB, MDRBI and CONVEX datasets are chosen to test the proposed algorithm because the EC algorithms developed in the first two parts of this project does not perform very well on CONVEX dataset by comparing to the other benchmark datasets, and the simplest variant and the most difficult variant of MNIST dataset are chosen in order to evaluate proposed HGAPSO with different datasets in various difficulties.

As it would be more convincing to evaluate the proposed EC algorithms on larger datasets such as CIFAR-10, but the computational cost is too high, e.g. one run of IPPSO on CIFAR-10 takes more than a week, which makes it not feasible to test all of the algorithms on CIFAR-10, it is more reasonable to choose the best proposed algorithm to be tested on CIFAR-10. The proposed HGAPSO method is expected to be the best, so if HGAPSO outperforms the other proposed EC algorithms on the six smaller datasets, a further experiments of running HGAPSO on CIFAR-10 will be performed. However, the experiment will not be ran by 30 times due to the very high computational cost, our limited GPU resource and the time constraint of the project. Instead, only one run of the experiment will be performed in order to obtain an initial result, which can be used to decide whether it is worth continuing the experiments for 30 runs in the future when more GPU resources are ready.

### 5.3.2 Parameter settings of the proposed EC methods

All of the parameters are configured according to the conventions in the communities of PSO [39] and GAs [6] along with taking into account the computational cost and the complexity of the search space. The values of the parameters of the proposed algorithms are listed in Table 5.1.

Table 5.1: Parameter list

| Parameter | Value |
|---|---|
| PSO | |
| acceleration coefficient array for $P_{id}$ | 1.49618 |
| acceleration coefficient array for $P_{gd}$ | 1.49618 |
| inertia weight for updating velocity | 0.7298 |
| GA | |
| mutation rate | 0.01 |
| cross over rate | 0.9 |
| elitism rate | 0.1 |
| HGAPSO parameters | |
| the range of # of layers in each block | [4, 8] |
| the range of growth rate in each block | [8, 32] |
| population size | 20 |
| generation | 10 |

## 5.4 Results and Discussions

Since the proposed EC method is stochastic as the other algorithms proposed in this project, statistical significance test is required to make the comparison result more convincing. When comparing the proposed HGAPSO with the state-of-the-art methods, One Sample T-Test is applied to test whether the results of HGAPSO is better; when the comparison of error rates between HGAPSO and the proposed DECNN is performed, Two Sample T-test is utilised to determine whether the difference is statistically significant or not. Table 5.2 shows the comparison results between the proposed HGAPSO and the state-of-the-art algorithms; Table 5.3 compares HGAPSO with DECNN.

### 5.4.1 HGAPSO vs. State-of-the-Art methods

The experimental results and the comparison between the proposed HGAPSO and the state-of-the-art methods are shown in Table 5.2. In order to clearly show the comparison results, the terms (+) and (-) are provided to indicate the result of HGAPSO is better or worse than the best result obtained by the corresponding peer competitor; The term (=) shows that the mean error rate of HGAPSO are slightly better or worse than the competitor, but the difference is not significant from the statistical point of view; The term – means there are no available results reported from the provider or cannot be counted.

It can be observed that the proposed HGAPSO method achieves a significant improvement in terms of the error rates shown in Table 5.2. HGAPSO significantly outperforms the other peer competitors across all three benchmark datasets. To be specific it reduces the error rate of the best competitor by 5%, 1% and 10% on CONVEX, MB and MDRBI datasets, respectively.

Table 5.2: The classification errors of HGAPSO against the peer competitors

| classier | CONVEX | | MB | | MDRBI | |
|---|---|---|---|---|---|---|
| CAE-2 | | – | 2.48 | (+) | 45.23 | (+) |
| TIRBM | | – | | – | 35.50 | (+) |
| PGBM+DN-1 | | – | | – | 36.76 | |
| ScatNet-2 | 6.50 | (+) | 1.27 | (+) | 50.48 | (+) |
| RandNet-2 | 5.45 | (+) | 1.25 | (+) | 43.69 | (+) |
| PCANet-2 (softmax) | 4.19 | (+) | 1.40 | (+) | 35.86 | (+) |
| LDANet-2 | 7.22 | (+) | 1.05 | (+) | 38.54 | (+) |
| SVM+RBF | 19.13 | (+) | 30.03 | (+) | 55.18 | (+) |
| SVM+Poly | 19.82 | (+) | 3.69 | (+) | 54.41 | (+) |
| NNet | 32.25 | (+) | 4.69 | (+) | 62.16 | (+) |
| SAA-3 | 18.41 | (+) | 3.46 | (+) | 51.93 | (+) |
| DBN-3 | 18.63 | (+) | 3.11 | (+) | 47.39 | (+) |
| HGAPSO(best) | 1.03 | | 0.74 | | 10.53 | |
| HGAPSO(mean) | 1.24 | | 0.84 | | 12.23 | |
| HGAPSO(standard deviation) | 0.10 | | 0.07 | | 0.86 | |

### 5.4.2 HGAPSO vs. DECNN

Since DECNN achieves better performance than the EC algorithms developed in the first part, DECNN is chosen as the peer EC competitor. In Table 5.3, it can be observed that by comparing the results between HGAPSO and DECNN, both the mean error rate and the standard deviation of HGAPSO are smaller than those of DECNN, and from the statistical point of view, HGAPSO has a significant improvement in terms of the classification accuracy.

Table 5.3: Classification rates of HGAPSO and DECNN

| | CONVEX | MB | MDRBI |
|---|---|---|---|
| HGAPSO(mean) | 1.24 | 0.84 | 12.23 |
| HGAPSO(standard deviation) | 0.10 | 0.07 | 0.86 |
| DECNN(mean) | 11.19 | 1.46 | 37.55 |
| DECNN(standard deviation) | 1.94 | 0.11 | 2.45 |
| P-value | **0.0001** | **0.0001** | **0.0001** |

### 5.4.3 Evolved CNN Architecture

After investigating the evolved CNN architectures, it is found that HGAPSO demonstrates its capability of evolving both the structure of CNNs and the shortcut connections between layers. By looking into the evolved CNN architectures, it can be observed that not only the CNN architectures with various number of layers but also different topologies of shortcut connections are evolved. Here is an example of the evolved CNN architecture with 3 blocks. In the first block, there are 4 conv layers, and [0, 0, 0, 0, 1], [0, 1, 0, 1], [0, 0, 1], [0, 0] and [1] represent the connections from the input, the first layer, the second layer, the third layer to the following layers, where 1 indicates the connection exists, and 0 means no connection; The second block is composed of 8 layers with the growth rate of 34, and the corresponding connections are [1, 0, 1, 0, 1, 0, 1, 0], [0, 1, 1, 1, 1, 0, 1], [1, 1, 1, 1, 1, 0], [1, 1, 1, 0, 1], [1, 0, 0, 0], [0, 0, 0], [1, 1] and [0]; In the third block, there are 5 layers with the corresponding connections of [0, 0, 1, 1, 0], [0, 0, 0, 0], [1, 0, 0], [0, 1] and [0], and the growth rate is 39.

### 5.4.4 Initial result on CIFAR-10 dataset

As mentioned earlier, the computational cost of testing HGAPSO is extremely high. For one run of the experiment using one GPU card, it takes more than a week to evolve the CNN architecture, and it spend almost 12 hours to trained the evolved CNN architecture. The classification accuracy of the specific run is 90.08%, which ranks in the middle of the state-of-the-art deep neural networks ranging from 75.86% to 96.53% that are collected by the rodrigob website [1]; However, all of the state-of-the-art deep neural networks require very high specialised domain knowledge and tremendous experiments to manually fine-tune the performance, while HGAPSO has the ability of automatically evolving the CNN architecture without any human interference, which is considered as the biggest advantage.

## 5.5 Conclusions

It can be concluded that the hybrid two-level EC method outperforms the other proposed algorithms from the first two parts of the project by evolving the more advanced architectures of CNNs instead of the traditional CNN architectures mainly because of two reasons. The first reason is that by introducing shortcut connections, the feature maps learned in previous layers can be reused in further layers, which amplifies the leverage of useful knowledge; Secondly, the shortcut connections makes the training of very deep neural networks more effectively by passing the gradients through shortcut connections, which has been proven by DenseNet [10].

The classification accuracy of HGAPSO on CIFAR-10 is really promising as it is very competitive with the state-of-the-art deep neural networks. In addition, the most advantage

[1]http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#43494641522d3130

of HGAPSO is that it does not require any human efforts to design the architecture of CNNs, which is usually required for the peer state-of-the-art competitors.

# Chapter 6

# Conclusions and Future Work

## 6.1 Major Conclusions

The main objectives of this whole project comprised of three parts have been successfully accomplished, which provides the automatic methods to design CNN architectures with competitive or even better performance than the state-of-the-art algorithms. In the first part, the proposed DE and GA methods using the IP-Based encoding strategy have been developed, and it has proven its competitiveness with the state-of-the-art methods. The IP-Based encoding strategy has shown its flexibility and powerfulness of encoding very complex parameters with various number of parameters and arbitrary ranges of values into a search space with a fixed length, which makes it straightforward to apply EC algorithms on these tasks; In the second part, a new DE algorithm which can be applied on vectors with various length is designed and implemented, and a second crossover is introduced to enhance the variety of the population in terms of the vector length, which grants the ability of evolving the length along with the value of the vector on the proposed algorithm. As a result, the proposed hybrid DE algorithm is able to evolve CNN architectures without any constraints of depths, and the performance has been improved comparing the proposed algorithms in the first part. In the last part of this project, more advanced and recent CNN architectures with shortcut connections are evolved by the proposed hybrid two-level algorithm, which have achieved a significant improvement in terms of the classification accuracy comparing the other proposed algorithms in this project.

## 6.2 Future Work

In regard to the future work, there are two aspects coming up from the experiments and learnt experience of this project. Firstly, due to the hardware limitation, all of the proposed algorithms are tested on relatively small datasets. Even though an initial result of running HGAPSO on CIFAR-10 is achieved, the statistical analysis based on the results from 30 runs needs to be applied in order to make a stronger claim of the proposed HGAPSO. It would be more convincing if the algorithms could be tested on other larger datasets such as ImageNet dataset. Fortunately, more hardware is setting up in our lab, which makes it feasible to do more researches on larger datasets. Secondly, as the CNN architectures with shortcut connections shows dramatical improvement, it would be helpful to investigate more recent CNN architectures and then apply EC methods on searching for a good solution among different types of recent CNN architectures.

# Bibliography

[1] Abdel-Hamid, O., Deng, L., and Yu, D. Exploring convolutional neural network structures and optimization techniques for speech recognition. In *Interspeech 2013* (August 2013), ISCA.

[2] Abdel-Hamid, O., Mohamed, A.-r., Jiang, H., and Penn, G. Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on* (2012), IEEE, pp. 4277–4280.

[3] Bruna, J., and Mallat, S. Invariant scattering convolution networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence 35*, 8 (2013), 18721886.

[4] Chan, T.-H., Jia, K., Gao, S., Lu, J., Zeng, Z., and Ma, Y. Pcanet: A simple deep learning baseline for image classification? *IEEE Transactions on Image Processing 24*, 12 (2015), 50175032.

[5] Chollet, F. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint* (2017), 1610–02357.

[6] Digalakis, J., and Margaritis, K. An experimental study of benchmarking functions for genetic algorithms. *Proceedings of 2000 IEEE International Conference on Systems, Man and Cybernetics.* (2000).

[7] Eberhart, R., and Kennedy, J. A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on* (Oct 1995), pp. 39–43.

[8] Gamperle, R., D Muller, S., and Koumoutsakos, A. A parameter study for differential evolution. *NNA-FSFS-EC 2002 10* (08 2002), 293–298.

[9] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR abs/1512.03385* (2015).

[10] Huang, G., Liu, Z., and Weinberger, K. Q. Densely connected convolutional networks. *CoRR abs/1608.06993* (2016).

[11] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360* (2016).

[12] Jones, M. T. Deep learning architectures, Sep 2017.

[13] K., S., G., Z., C., L., and H., L. Learning and selecting features jointly with point-wise gated boltzmann machines, June 2013.

[14] KALCHBRENNER, N., GREFENSTETTE, E., AND BLUNSOM, P. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188* (2014).

[15] KENNEDY, J., AND EBERHART, R. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on* (Nov 1995), vol. 4, pp. 1942–1948 vol.4.

[16] KIM, Y. Convolutional neural networks for sentence classification. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (08 2014).

[17] KINGMA, D. P., AND BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[18] KRIZHEVSKY, A., SUTSKEVER, I., AND HINTON, G. E. Imagenet classification with deep convolutional neural networks. *Communications of the ACM 60*, 6 (2017), 8490.

[19] LAROCHELLE, H., ERHAN, D., COURVILLE, A., BERGSTRA, J., AND BENGIO, Y. An empirical evaluation of deep architectures on problems with many factors of variation. *Proceedings of the 24th international conference on Machine learning - ICML 07* (2007).

[20] LAWRENCE, S., GILES, C. L., TSOI, A. C., AND BACK, A. D. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks 8*, 1 (1997), 98–113.

[21] LECUN, Y., BOSER, B., DENKER, J. S., HENDERSON, D., HOWARD, R. E., HUBBARD, W., AND JACKEL, L. D. Backpropagation applied to handwritten zip code recognition. *Neural Computation 1*, 4 (Dec 1989), 541–551.

[22] LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (Nov 1998), 2278–2324.

[23] MILLER, J., AND TURNER, A. Cartesian genetic programming. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation* (New York, NY, USA, 2015), GECCO Companion '15, ACM, pp. 179–198.

[24] MITCHELL, M. *An introduction to genetic algorithms*. MIT Press, 1996.

[25] POSTEL, J. Dod standard internet protocol, Jan 1980.

[26] PRICE, K. V., STORN, R. M., LAMPINEN, J. A., AND UNDEFINED, U. U. *Differential evolution: a practical approach to global optimization*. Springer, 2005, ch. 2, pp. 37–42.

[27] S., R., P., V., X., M., X., G., AND Y., B. Contractive auto-encoders: explicit invariance during feature extraction, June 2011.

[28] SCHUMER, M., AND STEIGLITZ, K. Adaptive step size random search. *IEEE Transactions on Automatic Control 13*, 3 (1968), 270276.

[29] SIMONYAN, KAREN, ZISSERMAN, AND ANDREW. Very deep convolutional networks for large-scale image recognition, Apr 2015.

[30] SOHN, KIHYUK, LEE, AND HONGLAK. Learning invariant representations with local transformations, Jun 2012.

[31] STANLEY, K. O. Neuroevolution: A different kind of deep learning, Jul 2017.

[32] Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life 15*, 2 (2009), 185–212.

[33] Stanley, K. O., and Miikkulainen, R. Evolving neural networks through augmenting topologies. *Evol. Comput. 10*, 2 (June 2002), 99–127.

[34] Storn, R. On the usage of differential evolution for function optimization. In *Proceedings of North American Fuzzy Information Processing* (June 1996), pp. 519–523.

[35] Storn, R., and Price, K. Differential evolution  a simple and efficient heuristic for global optimization over continuous spaces, 1997.

[36] Suganuma, M., Shirakawa, S., and Nagao, T. A genetic programming approach to designing convolutional neural network architectures. *CoRR abs/1704.00764* (2017).

[37] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015), pp. 1–9.

[38] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 2818–2826.

[39] Vandenbergh, F., and Engelbrecht, A. A study of particle swarm optimization particle trajectories. *Information Sciences 176*, 8 (2006), 937971.

[40] 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)* (July 2018), pp. 1–8.

[41] Wang, B., Sun, Y., Xue, B., and Zhang, M. A hybrid differential evolution approach to designing deep convolutional neural networks for image classification. *arXiv preprint arXiv:1808.06661* (2018).

[42] Xie, L., and Yuille, A. Genetic CNN. In *2017 IEEE International Conference on Computer Vision (ICCV)* (Oct 2017), pp. 1388–1397.

# Appendix A

# IP-Based EC algorithms

## A.1 IP-Based Random Search

A fixed step size random search (FSSRS) [28] is implemented as it is an easy and effective random search algorithm. In order to make a fair comparison with other IPEC methods, instead of using one candidate to perform the random search task, a population of numerous candidates is used to complete the search process. As the IP-Based Encoding Strategy is used to encode the architecture of CNNs into 8-byte vectors which constitute the search space, the objective of the IP-Based Random Search (IPRS) method is to optimise the accuracy in the constructed search space.

### A.1.1 IPRS Algorithm Overview

The framework of IPRS is mainly comprised of four key steps - initialise the population, update the position of each candidate of the population, retrieve the best candidate in the whole population, and repeat step 2 and 3 until the stop criteria are met. More details are depicted in the pseudo-code in Algorithm 11.

**Algorithm 11:** Framework of IPRS

```
$P \leftarrow$ Initialize the population with IP-Based Encoding Strategy elaborated in Section 3.2.3;
$P\_best \leftarrow empty$;
while termination criterion is not satisfied do
    update the position of each candidate solution as shown in Algorithm 12;
    evaluate the fitness value of each candidate;
    $P\_best \leftarrow$ find the best candidate in the population;
end while
```

### A.1.2 IPRS Position Update

The candidate contains a series of interfaces each of which represents a CNN layer, and each interface carries a 2-byte IP address, so in order to update the position of the candidate, the IP addresses in the candidate need to be flattened into an integer vector which is composed of 1-byte integers, and each byte is one dimension of the position of random search. When randomly searching the space, a random point on the hypersphere with a certain radius and the current point as the origin is selected as the new position. Instead of directly moving to the new position, the candidate only moves to the new position if the fitness value of the new position is better.

**Algorithm 12:** Update The Position of A Candidate Solution

```
Input: candidate vector $c\_vector$, radius for random search $r$;
Output: updated candidate vector $c\_vector$;
  $bytes\_vector \leftarrow$ extract each byte of the IP addresses from the interfaces of the candidate $c\_vector$ in order;
  $sphere\_vector \leftarrow$ randomly generate a vector with the radius $r$ and the size of $bytes\_vector$ as the dimension
  $i \leftarrow 0$;
  for $i <$ the size of $bytes\_vector$ do
    $bytes\_vector[i] \leftarrow bytes\_vector[i] + sphere\_vector[i]$;
    if $bytes\_vector[i] > 255$ then
      $bytes\_vector[i] \leftarrow bytes\_vector[i] - 255$
    end if
    $i \leftarrow i + 1$;
  end for
  $new\_c\_vector \leftarrow$ convert $bytes\_vector$ to a new candidate vector by generating IP addresses from $bytes\_vector$ which then are stored as interfaces in the new candidate vector
  $new\_fitness \leftarrow$ evaluate the new candidate vector $new\_c\_vector$;
  if $new\_fitness >$ the fitness of $c\_vector$ then
    $c\_vector \leftarrow new\_c\_vector$
  end if
  return $c\_vector$
```

## A.2 Experiments to fine-tune the hyperparameters of fitness evaluation

One hyperparameter of the fitness evaluation is the percentage of the dataset, which can dramatically affect the final results of the proposed methods. If the percentage of the dataset is too small, the partial dataset cannot represent the characteristics of the whole training dataset very well, so the final accuracy won't meet our expectation; however, the computational cost explodes along with the increase of the percentage, which indicates that using the whole training dataset for the fitness evaluation is not ideal. It can be observed that there is a conflict between achieving the best accuracy and the fast convergence, which need to be reconciled by fine-tuning the percentage of the dataset used for the fitness evaluation. To be specific with the experiments, IPDE with 40% and 70% given 5 epochs will be run on the six benchmark datasets, and the results will be compared with IPDE with 10% and the whole training dataset that are done in the experiments in Section 3.3.3.

The other hyperparameter is the number of epochs, which also influences the performance of the IPEC methods. If the number of epochs is not sufficient to train a good CNN architecture, the trend of the CNN architecture cannot be learned, which will produce some noises of the fitness value; however, if the number of epochs is too large, overfitting may occur, which results in noises of the fitness value as well because the accuracy obtained from the fitness evaluation is not able to represent the true accuracy of the CNN architecture on the specific dataset. As a result, the percentage of the dataset is fixed to 10% in order to attain the results in a short period without severely comprising the accuracy, and IPDE with various numbers of epochs - 10 and 15 will be done, whose results will be investigated to learn a pattern of how the number of epochs impact the final accuracy of the learned CNN architecture.

## A.3 Results and Discussions of Tuning Fitness Evaluation Hyper-parameters

### A.3.1 Performance Comparison by tuning the percentage of dataset for fitness evaluation

The results of the mean error rates, standard deviations and best error rates are reported in Table A.1. Even though there are no significant differences between any two of IPDE-10%-5, IPDE-40%-5, IPDE-70%-5 and IPDE-100%-5 by performing Two Sample T-Test on the results, there is still an implicit trend on the mean error rate along with the increase of the percentage of the dataset used. As shown in Fig. A.3.1, for the MBI, MDRBI and MRB benchmark datasets, the mean error rate continuously plunges when the percentage of dataset goes up; while, for the CONVEX, MB AND MRD datasets, the mean error rate fluctuates during the increase of the percentage. As the CNNs are trained on partial datasets, the more data are used the more accurate it can represent the distribution of the dataset, which is supposed to produce a better accuracy, and it is perfectly matched by the MBI, MDRBI and MRB benchmark datasets; while, for the other three datasets, the error rate doesn't have a clear trend of rising or falling, and it slightly goes up and down. Since the CONVEX, MB AND MRD datasets are relatively simple compared to the other three, 10% dataset may be able to represent the correct distribution of the whole dataset, so it achieves similar error rates with a bit fluctuation during the increase of the percentage of datasets.

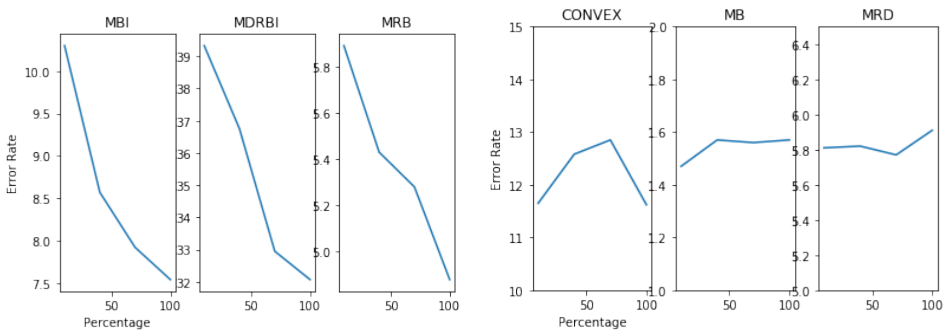
Table A.1: The classification errors of IPDE by tuning the percentage of data used for fitness evaluation

| percentage | CONVEX | MB | MBI | MDRBI | MRB | MRD |
|---|---|---|---|---|---|---|
| Best Error Rate of State of Art method | | | | | | |
| | 4.19 | 1.05 | 11.55 | 35.50 | 6.08 | 4.45 |
| Mean Error Rate | | | | | | |
| 10% | 11.65 | 1.47 | 10.30 | 39.33 | 5.89 | 5.81 |
| 40% | 12.58 | 1.57 | 8.57 | 36.75 | 5.43 | 5.82 |
| 70% | 12.85 | 1.56 | 7.92 | 32.95 | 5.28 | 5.77 |
| 100% | 11.62 | 1.57 | 7.54 | 32.07 | 4.88 | 5.91 |
| Standard Deviation | | | | | | |
| 10% | 1.94 | 0.15 | 1.58 | 5.87 | 1.97 | 1.17 |
| 40% | 2.32 | 0.25 | 1.61 | 6.13 | 1.72 | 0.88 |
| 70% | 3.45 | 0.21 | 1.22 | 2.29 | 1.06 | 0.95 |
| 100% | 3.87 | 0.18 | 1.03 | 2.65 | 0.69 | 1.34 |
| Best Error Rate | | | | | | |
| 10% | 8.56 | 1.16 | 6.63 | 32.20 | 3.88 | 3.84 |
| 40% | 7.54 | 1.26 | 5.88 | 26.38 | 3.22 | 4.98 |
| 70% | 7.70 | 1.31 | 5.50 | 27.02 | 3.09 | 4.35 |
| 100% | 7.25 | 1.23 | 5.45 | 26.83 | 3.54 | 4.45 |

### A.3.2 Performance Comparison by tuning the training epochs for fitness evaluation

Along with the growth of epochs by fixing the percentage of the dataset, the performance of the trained CNN climbs, which can reflect the correct trend of the corresponding fully-trained CNN better, but when the epochs exceeds some point, the performance of the trained

Figure A.1: The trend of the mean error rate along with increasing the percentage of dataset



CNN plunges due to over-fitting, so it can not reflect the trend of the fully-trained CNN anymore, which adds noises to the fitness value. This assumption is supported by the changes in the mean error rates of different datasets. For the CONVEX, MB, MRD benchmark datasets, the CNN trained for 5 epochs are enough or more than enough to represent the fully-trained CNN, so the mean error rate for the CONVEX dataset rockets from 5 epoch to 10 epochs, for the MB dataset, the error rate keeps rising, and for the MRD dataset, the error rate receives a little increase from 5 epochs to 10 epochs and jumps sharply from 10 epochs to 15 epochs. With regard to the MBI, MDRBI datasets with more complexity, the mean error rate drops with the growth of the epoch number. The mean error rate of the MRB datasets reduces while epochs changes from 5 to 10, and it hikes during the change of epochs from 10 to 15, which indicates that 10 epochs could be the point where the trained CNN could represent the fully-trained CNN best.

Table A.2: The classification errors of IPDE by tuning the epochs for fitness evaluation

| epochs | CONVEX | MB | MBI | MDRBI | MRB | MRD |
|---|---|---|---|---|---|---|
| Best Error Rate of State of Art method | | | | | | |
| | 4.19 | 1.05 | 11.55 | 35.50 | 6.08 | 4.45 |
| Mean Error Rate | | | | | | |
| 5 | 11.65 | 1.47 | 10.30 | 39.33 | 5.89 | 5.81 |
| 10 | 12.58 | 1.49 | 9.46 | 37.50 | 5.53 | 5.82 |
| 15 | 12.47 | 1.54 | 9.31 | 37.20 | 5.88 | 6.44 |
| Standard Deviation | | | | | | |
| 5 | 1.94 | 0.15 | 1.58 | 5.87 | 1.97 | 1.17 |
| 10 | 1.65 | 0.31 | 2.07 | 5.01 | 1.31 | 0.94 |
| 15 | 2.27 | 0.42 | 2.00 | 6.20 | 1.52 | 1.94 |
| Best Error Rate | | | | | | |
| 5 | 8.56 | 1.16 | 6.63 | 32.20 | 3.88 | 3.84 |
| 10 | 9.03 | 1.00 | 6.49 | 30.30 | 3.54 | 4.41 |
| 15 | 9.14 | 1.13 | 5.16 | 29.63 | 4.07 | 4.75 |