Automatically Designing CNN Architectures Using Genetic Algorithm for Image Classification

Yanan Sun, *Member, IEEE,* Bing Xue, *Member, IEEE,* Mengjie Zhang, *Senior Member, IEEE,* and Gary G. Yen, *Fellow, IEEE*

Abstract—Convolutional Neural Networks (CNNs) have gained a remarkable success on many real-world problems in recent years. However, the performance of CNNs is highly relied on their architectures. For some state-of-the-art CNNs, their architectures are hand-crafted with expertise in both CNNs and the investigated problems. To this end, it is difficult for researchers, who have no extended expertise in CNNs, to explore CNNs for their own problems of interest. In this paper, we propose an automatic architecture design method for CNNs by using genetic algorithms, which is capable of discovering a promising architecture of a CNN on handling image classification tasks. The proposed algorithm does not need any pre-processing before it works, nor any post-processing on the discovered CNN, which means it is completely automatic. The proposed algorithm is validated on widely used benchmark datasets, by comparing to the state-of-the-art peer competitors covering eight manually designed CNNs, four semi-automatically designed CNNs and additional four automatically designed CNNs. The experimental results indicate that the proposed algorithm achieves the best classification accuracy consistently among manually and automatically designed CNNs. Furthermore, the proposed algorithm also shows the competitive classification accuracy to the semi-automatic peer competitors, while reducing 10 times of the parameters. In addition, on the average the proposed algorithm takes only one percentage of computational resource compared to that of all the other architecture discovering algorithms. Experimental codes and the discovered architectures along with the trained weights are made public to the interested readers.

Index Terms—Convolutional neural network, genetic algorithm, neural network architecture optimization, evolutionary deep learning.

1 Introduction

Convolutional Neural Networks (CNNs), as the dominant technique of deep learning [1], have shown remarkable superiority in various real-world applications over most machine learning approaches [2], [3], [4], [5]. Since the year of 1998 when the first version of the CNN (i.e., LeNet5 [6]) was proposed, diverse CNN variants have been developed, such as AlexNet [2], VGG [7], GoogleNet [8], ResNet [9] and DenseNet [10], to name a few. There is a trend among the CNN variants that their architectures become increasingly deeper. For example, the depth of LeNet5 is six, VGG is 16, while ResNet [11] achieves at a depth of $1,202$. The principle behind such designs is that a deeper CNN typically has a more powerful capability to address much complex and large-scale data.

The state-of-the-art CNNs are typically designed by experts who have rich domain knowledge from both investigated data and CNNs. Because the performance of the CNNs strongly relies on the investigated data, it is expected that there is a major limitation to this design manner. For example, researchers who are familiar with the data at hand do not necessarily have the experience in designing the architectures of CNNs, and vice versa. To this end, there is a great demand for developing algorithms which allow researchers without any expertise to automatically derive the best performing CNN for the given data. Indeed, multiple algorithms for this purpose have been proposed in recent years.

In general, the algorithms of designing architectures for CNNs can be divided into two different categories according to their base techniques. The first covers methods using evolutionary algorithms [12], such as the genetic CNN method (Genetic CNN) [13], the large-scale evolution method (Large-scale Evolution) [14], the hierarchical representation method (Hierarchical Evolution) [15] and the Cartesian genetic programming method (CGP-CNN) [16]. These algorithms follow the standard flow of an evolutionary algorithm to heuristically discover the optimal solution. The second refers to algorithms based on reinforcement learning [17], such as the neural architecture search method (NAS) [18], the meta-modelling method (MetaQNN) [19], the efficient architecture search method (EAS) [20] and the block design method (Block-QNN-S) [21]. The algorithms in the second category resemble those in the first category, in addition to the employed heuristic nature that algorithms in the second category utilize the reward-penalty principle of reinforcement learning.

Experimental results of these algorithms have demonstrated promising classification accuracy in the challenging benchmark datasets, such as CIFAR10 and CIFAR100 [22], but limitations also exist. Firstly, the algorithms in the first category do not make full use of the advantages of evolutionary algorithms, which results in consuming extensive computational resource and not promising classification accuracy. Secondly, the algorithms in the second category require even more computational resources than the algo-

- *Yanan Sun, Bing Xue, and Mengjie Zhang are with the School of Engineering and Computer Science, Victoria University of Wellington, PO Box 600, Wellington 6140, New Zealand (e-mails: yanan.sun@ecs.vuw.ac.nz; bing.xue@ecs.vuw.ac.nz; and mengjie.zhang@ecs.vuw.ac.nz).*
- *Gary G. Yen is with the School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK 74078 USA (e-mail:gyen@okstate.edu).*

rithms in the first category due to the nature of reinforcement learning. Thirdly, manual assistances based on domain expertise are required for most algorithms in both categories. To this end, the development of algorithms automatically 1) discovering the best CNN architectures for given data, 2) relying on the limited computational resources and 3) being directly used without any manual refinement or re-composition regarding their discovered CNNs, is still in its infancy. Note that, depending on whether expertise in CNNs is required or not in using these algorithms, they can also be classified into the automatic and the semi-automatic categories. The first includes Large-scale Evolution, CGP-CNN, NAS and Meta-CNN, while the second is composed of Genetic CNN, Hierarchical Evolution, EAS and Block-QNN-S.

Evolutionary algorithm [12] is a class of population-based meta-heuristic optimization paradigm inspired by the biological evolution. Typical evolutionary algorithms include genetic algorithms (GAs) [23], genetic programming [24], evolutionary strategy [25], etc., among which GAs are the most popular one mainly because of their theoretical evidences [26] and promising performance in solving different optimization problems [27], [28], [29], [30], [31]. It has also been recognized that GAs are capable of generating high-quality optimal solutions by using bio-inspired operators, i.e., mutation, crossover and selection [32]. Our goal in this paper is to develop an effective and efficient algorithm by using GA, in short, termed as CNN-GA, to automatically discover the best architectures of CNNs for given image classification tasks, so that the discovered CNN can be directly used without any manual refinement or re-composition. The contributions of the proposed CNN-GA method are summarized as follows:

1) The depth of CNNs is not limited to a predefined number in the proposed algorithm, but instead the best depth is discovered during the evolutionary process, by finding the CNN with the best classification accuracy for the given data. Although this design could produce the optimal CNN, the crossover operation, which plays the role of exploitation search (i.e., the local search), cannot work in such a situation due to individuals with unequal (variable) lengths. To address this need, we also design a crossover operator to adapt for these individuals during the evolutionary progress.
2) The skip connection, of which the superiority has been theoretically and experimentally proven in effectively training deep architectures, is directly incorporated into the proposed algorithm. In such a way, the evolved CNNs are capable of dealing with complex data by using deep architectures, by avoiding the Gradient Vanishing (GV) problems [33]. Furthermore, this design can also reduce the search space so that the best performance can be achieved within the limited time. In addition, compared to other models with similar performance, the architectures evolved by the proposed algorithm have a much smaller number of parameters.
3) The proposed algorithm is completely automated in discovering the best CNN architectures, and does not require any manual intervention during the evolutionary search. When evolution is finished, the obtained CNNs can be directly used to process the data, and do not need further refinement, such as adding more convolutional or pooling layers. Furthermore, the proposed algorithm can be directly used by other researchers who do not need to do any preparations such as providing a manually tuned network in advance.
4) An asynchronous computational component is designed to make full use of the given computational resources to accelerate the evaluation of fitness for the individuals in the same generation of the proposed algorithm. In addition, a cache component is also developed which is expected to further reduce the fitness evaluation time for the whole population.

The remainder of this paper is organized as follows. Firstly, the background is presented in Section 2. Then, the details of the proposed algorithm are documented in Section 3. Next, the experimental designs and experimental results are shown in Sections 4 and 5, respectively. Finally, conclusions and future works are outlined in Section 6.

2 Background

In this section, CNNs, skip connections and GAs, which are the background of the proposed algorithm, are introduced to help readers better understand the related works and the proposed algorithm.

2.1 CNNs

In this subsection, we mainly introduce the building blocks of CNNs, i.e., the convolutional and pooling layers, which are the basic objects encoded by GAs to represent CNNs.

Specifically, the convolutional layer employs filters to perform convolutional operations on the input data. One filter can be viewed as a matrix. In this paper, we focus on 2-dimension convolution operators (with a 2-dimension filter) because the proposed algorithm aims at processing image data. During the convolutional operation, the filter horizontally slides (with a given step size), then vertically moves (with another step size) for the next horizontal slide, until the whole image has been scanned. At each position, the filter is applied to the image by multiplying each filter value with the corresponding pixel value, and summing the results to give the output of the filter. The set of filter outputs form a new matrix called the feature map. The horizontal and vertical step sizes are called the width and height of a stride. In a convolutional layer, multiple filters (typically with the same sizes and using the same stride) are allowed to coexist, producing a set of feature maps. The exact number of feature maps used is a parameter in the architecture of the corresponding CNN. The number of filters is derived from the number of resulting feature maps and the spatial size of the input data. In addition, two convolutional operations are applied: the *same* convolutional operation which pads zeros to the input data when there is no enough area for the filter to overlap, and the *valid* convolutional operation which does not pad anything. Hence, the parameters of a convolutional layer

are the number of feature maps, the filter size, the stride size and the convolutional operation type.

A pooling layer has common components of a convolutional layer except that 1) the filter is called the kernel which has no value, 2) the output of a kernel is the maximal or mean value of the area it stops, and 3) the spatial size of the input data is not changed through a pooling layer. When the maximal value is returned, it is a pooling layer with the type of *max*, otherwise of *mean*. Hence, the parameters of a pooling layer are the kernel size, the stride size and the pooling type used.

In addition, the fully-connected layers are usually incorporated into the tail of a CNN. Because it is very common and also not the building blocks of CNNs, we will not detail it here. But note that, the number of fully-connected layers and the number of neurons in each fully-connected layer are also the parameters of a CNN's architecture, if the fully-connected layers are used in the CNN. In the proposed algorithm, the fully-connected layers are discarded, and the justifications are given in Subsection 3.2.

2.2 Skip Connections

Commonly, the connections in CNNs exist between the neurons of two adjacent layers. Analogously, the skip connections refer to those connecting the neurons of the layers that are not adjacent. The skip connection was firstly introduced in [33] as a gate mechanism, effectively training a recurrent neural network with long and short-term memory [34] and avoiding the GV problems [33]. Specifically, the GV problems refer to the gradient becoming very small or explosion during back propagation training in a deep neural network. Because gradient-based algorithms are the dominant learning algorithms of neural networks, the GV problems are the main obstacle to effectively training *deep* neural networks. The skip connections were experimentally proven to be able to train very deep neural networks [35]. Indeed, the promising performance of ResNet [9], which was proposed very recently, also benefits from the skip connections. A simple example of using a skip connection is shown in Fig. 1, where the dashed line denotes the skip connection from the input X to the output of the N-th building block, and the symbol "$\oplus$" refers to the element-wise addition.

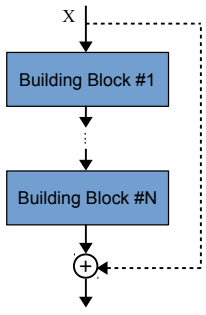

Fig. 1. An example of using the skip connection.

Over the past few years, an increasing number of researchers has attempted to theoretically reveal the mechanisms behind the skip connections. For example, the skip connections have also been claimed to be able to eliminate singularities [36]. However, a completely satisfactory explanation is still elusive. Among the existing theoretical evidence, skip connections defying the GV problems receives the most recognition [33], [34]. The GV problems frequently occur when the error has been back-propagated over many layers of a given neural network. Because the skip connections shorten the number of layers of back-propagation, the GV problems should be alleviated. As discussed above, the deeper the CNN is, the more powerful capability it would have to process complex data. Combined with the connections that are not skipped, a CNN with the skip connections can have the capability that a deep architecture has and can also be effectively trained.

Owning to the promising performance of the skip connections, it is naturally incorporated into the proposed algorithm to discover the architectures of an optimal CNN. In addition, the search space can also be reduced if these skip connections are directly used. Consequently, the required computational resources can be minimized, and the promising architectures of CNNs can be discovered within a limited time.

2.3 Genetic Algorithms

The flowchart of a GA is shown in Fig. 2. Specifically, a population of individuals (i.e., CNNs with architecture variants) is randomly initialized first, and then the fitness of each individual is evaluated. The fitness is measured by a deterministic function which is known as the fitness function, based on the context of the problem to be optimized (i.e., performance of CNNs on specific image classification tasks). The input of the function is the decision variables encoded in the individuals. After that, the individuals who have better fitness will be chosen by the selection operation, to hopefully generate offspring with better fitness. Specifically, new offspring are generated by exchanging or varying the encoded information of the selected parent individuals, i.e. by the crossover and mutation operations. These generated offspring are evaluated for the fitness, and then the population surviving into the next generation are selected from the current population, which is composed of the parent population and generated offspring, by the environmental selection. Through repeating a series of these operations, it is expected to find the optimal solution from the population when the GA terminates. Note that, the typical criterion to terminate a GA is a predefined maximal generation number, say 20 in our experiments.

3 The Proposed Algorithm

In this section, we firstly present the framework of the proposed algorithm in Subsection 3.1, and then detail the main steps in Subsections 3.2 to 3.5. To help readers better understand the proposed algorithm, we will not only document the details of each main step, but also provide the justifications for such designs.

3.1 Algorithm Overview

Algorithm 1 shows the framework of the proposed algorithm. Specifically, by giving a set of predefined building

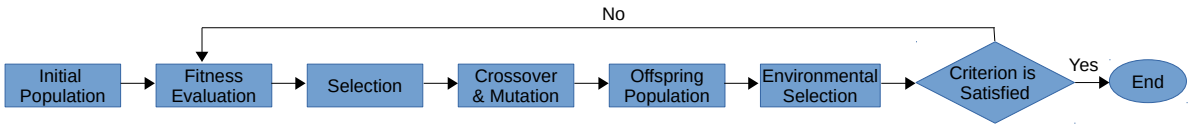

Fig. 2. The flowchart of genetic algorithm.

Algorithm 1: Framework of The Proposed Algorithm

Input: A set of predefined building blocks, the population size, the maximal generation number, the image dataset for classification.
Output: The discovered best architecture of CNN.

1 $P_0 \leftarrow$ Initialize a population with the given population size;
2 $t \leftarrow 0$;
3 **while** $t <$ *the maximal generation number* **do**
4 Evaluate the fitness of each individual in P_t;
5 $Q_t \leftarrow$ Generate offspring by genetic operators from the selected parent individuals;
6 $P_{t+1} \leftarrow$ Environmental selection from $P_t \cup Q_t$;
7 $t \leftarrow t + 1$;
8 **end**
9 **Return** the individual which has the best fitness in P_t.

blocks of CNNs, the population size as well as the maximal generation number for the GA and the image classification dataset, the proposed algorithm begins to work, through a series of evolutionary processes, and finally discovers the best architecture of the CNN to classify the given image dataset. During evolution, a population is randomly initialized with the predefined population size, using the proposed encoding strategy to encode the predefined building blocks (line 1). Then, a counter for the current generation is initialized to zero (line 2). During evolution, the fitness of each individual, which encodes a particular architecture of the CNN, is evaluated on the given dataset (line 4). After that, the parent individuals are selected based on the fitness, and then generate new offspring by the genetic operators (line 5). Then, a population of individuals surviving into the next generation are selected by the environmental selection from the current population (line 6). Specifically, the current population is composed of the parent population and the generated offspring population. Finally, the counter is increased by one, and the evolution continues until the counter exceeds the predefined maximal generation. As shown in Fig. 2, the proposed algorithm follows the standard pipeline of a GA (the phases of selection, crossover and mutation, and the offspring population shown in Fig. 2 are collectively described in line 5 of Algorithm 1). Note that, the genetic operator is composed of the crossover and mutation operations.

3.2 Population Initialization

As introduced in Section 2, a CNN is composed of the convolutional layers, pooling layers and occasionally fully-connected layers. The performance of a CNN highly relies on its depth, and the skip connections could turn the *deep depth* to be a reality. In the proposed encoding strategy, we design a new building block by directly using the skip connections, named the skip layer, to replace the convolutional layer when forming a CNN. In addition, the fully-connected layers are discarded in the proposed encoding strategy (the reason will be given later in this subsection). In summary, only the skip layers and the pooling layers are used to construct a CNN in the proposed encoding strategy.

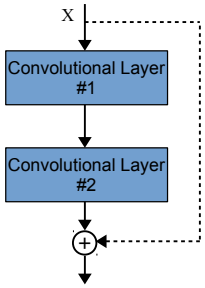

Fig. 3. An example of the skip layer in the proposed encoding strategy.

Specifically, a skip layer is composed of two convolutional layers and one skip connection. The skip connection connects from the input of the first convolutional layer to the output of the second convolutional layer. An example of the skip layer is shown in Fig. 3. As introduced above, the parameters of a convolutional layer are the number of feature maps, the filter size, the stride size and the convolutional operation type. In the proposed encoding strategy, we use the same settings for the filter sizes, stride sizes and convolutional operations. Particularly, the filter and stride sizes are set to 3×3 and 1×1, respectively, and only the *same* convolutional operation type is used. To this end, the parameters encoded for a skip layer are the numbers of the feature maps for the two convolutional layers (denoted as $F1$ and $F2$, respectively). In addition, the pooling layers used in the proposed encoding strategy are set to be 2×2 for both the kernel sizes and the stride sizes. To this end, the parameter encoded for a pooling layer is only the pooling type (denoted as $P1$). Note that, the reasons for this design and adopting the settings for such a design will be explained later in this subsection.

Algorithm 2 shows the details of the population initialization. Briefly, T individuals are initialized in the same manners, and then they are stored into P_0. During the individual initialization process, the length (denoted as L) of an individual, representing the depth of the corresponding CNN, is randomly initialized at first (line 3). Then, a linked list containing L nodes is created (line 4). After that, each node is configured (lines 5-20), and then the linked list is stored into P_0 (line 21). During the configuration of each node, a number, r, is randomly generated from $(0, 1)$ (line 6). If $r < 0.5$, the type of this node is marked as a skip

Algorithm 2: Population Initialization

Input: The population size T.
Output: The initialized population P_0.

```
1  P_0 ← ∅ ;
2  while |P_0| < T do
3      L ← Randomly generate an integer greater than zero;
4      list ← Create a linked list contains L nodes;
5      foreach node in the linked list do
6          r ← Uniformly generate a number from (0, 1);
7          if r < 0.5 then
8              node.type ← 1;
9              node.F1 ← Randomly generate an integer greater than zero;
10             node.F2 ← Randomly generate an integer greater than zero;
11         else
12             node.type ← 2;
13             q ← Uniformly generate a number from (0, 1);
14             if q < 0.5 then
15                 node.P1 ← max;
16             else
17                 node.P1 ← mean;
18             end
19         end
20     end
21     P_0 ← P_0 ∪ list;
22 end
23 Return P_0.
```

layer by setting its $type$ property to 1. Otherwise, this node represents a pooling layer by setting its $type$ to 2. In the case of a skip connection layer, the numbers of the feature maps are randomly generated and then assigned to $node.F1$ and $node.F2$, respectively (lines 7-10). Otherwise, the pooling type is determined by the probability of flipping a coin. Particularly, the pooling type, $node.P1$, is set to *max* when the probability is below 0.5, and *mean* otherwise (lines 11-19).

Next, we will detail the reasons why discarding the fully-connected layers, using two convolutional layers in a skip layer and the settings for the skip and pooling layers in the proposed algorithm. Typically, multiple fully-connected layers are added to the tail of a CNN. However, the fully-connected layer easily results into the over-fitting phenomenon [40] due to its dense connection [41]. To reduce the over-fitting, the dropout [41] randomly removing a part of the connections is commonly used. However, each dropout will introduce one additional parameter. Only a properly specified parameter can lead to the promising performance of the corresponding CNN. Meanwhile, the number of fully-connected layers and the number of neurons in each fully-connected layer are extra parameters hard to tune. If the fully-connected layers are incorporated into the proposed encoding strategy, the search space will be substantially enlarged, increasing the difficulty of searching for the best CNN architecture. The use of two convolutional layers in a skip layer is inspired by the design of ResNet, and the effectiveness of such skip layers have been experimentally proven in literature [10], [11], [14], [15]. However, the sizes of feature maps in each skip layer of ResNet are set to be equal. In our proposed encoding strategy, the sizes of feature maps can be different, which is believed to be more flexible. Furthermore, setting the convolutional operation type to *same* and using the 1×1 stride are to make the dimension of the input data remain the same, which is more flexible for such an automatic design. Please note 1×1 stride does not change the image size. As far as the settings of filter and kernel sizes as well as the stride size in the pooling layers are concerned, they are all based on the designs of existing hand-crafted CNNs [10], [11]. Moreover, another important reason for specifying such settings is based on our expertise in manually tuning the architectures of CNNs. The effectiveness of such settings will be shown in Section 5.

3.3 Fitness Evaluation

Algorithm 3: Fitness Evaluation

Input: The population P_t of the individuals to be evauated, the image dataset for clasification.
Output: The population P_t of the individuals with fitness values.

```
1  if t == 0 then
2      Cache ← ∅;
3      Set Cache to a global variable;
4  end
5  foreach individual in P_t do
6      if the identifier of individual in Cache then
7          v ← Query the fitness by identifier from Cache;
8          Set v to individual;
9      else
10         while there is available GPU do
11             asynchronously evaluate individual in an availabel GPU (details shown in Algorithm 4);
12         end
13     end
14 end
15 Return P_t.
```

Algorithm 3 details the fitness evaluation of the individuals in the population P_t. Briefly, given the population, P_t, containing all the individuals for evaluating the fitness, and the image classification dataset on which the best architecture of a CNN is to be discovered, Algorithm 3 evaluates each individual of P_t in the same manner, and finally returns P_t containing the individuals whose fitness have been evaluated. Specifically, if the fitness evaluation is for the initialized population, i.e., P_0, a global cache system (denoted as $Cache$) is created, storing the fitness of the individuals with unseen architectures (lines 1-4). For each individual (denoted by $individual$) in P_t, if $individual$ is found in $Cache$, its fitness is directly retrieved from $Cache$ (lines 6-8). Otherwise, $individual$ is asynchronously placed on an available GPU for its fitness evaluation (lines 9-13). Note that, querying an individual from $Cache$ is based on

the individual's identifier. Theoretically, arbitrary identifiers can be used, as long as they can distinguish individuals encoding different architectures. In the proposed algorithm, the 224-hash code [42], which has been implemented by most programming languages, in terms of the encoded architecture is used as the corresponding identifier. Furthermore, the individual is asynchronously placed on an available GPU, which implies that we don't need to wait for the fitness evaluation for the next individual until the fitness evaluation of the current one finishes, but simply place the next individual on an available GPU immediately.

Algorithm 4: Individual Fitness Evaluation

Input: The individual $individual$, the available GPU, the number of training epochs, the global cache $Cache$, the training data D_{train} and the validation data D_{valid} from the given image classification dataset.
Output: The individual $individual$ with its fitness.

```
1  Construct a CNN with a classifier based on the
     information encoded in individual and the given
     image classification dataset;
2  v_best ← 0;
3  foreach epoch in the given training epochs do
4      Train the CNN on D_train by using the given GPU;
5      v ← Calculate the classification accuracy on
         D_valid;
6      if v > v_best then
7          v_best ← v;
8      end
9  end
10 Set v_best as the fitness of individual;
11 Put the identifier of individual and v_best into Cache;
12 Return individual.
```

The details of evaluating the fitness of one individual are shown in Algorithm 4. Firstly, a CNN is decoded from $individual$, and a classifier is added to this CNN (line 1) based on the given image classification dataset. In the proposed algorithm, a softmax classifier [43] is used, and the particular number of classes is determined by the given image dataset. When decoding a CNN, a rectifier activation function [44] followed by a batch normalization [45] operation is added to the output of the convolutional layer, which is based on the conventions of modern CNNs [9], [10]. In addition, when the spatial number of the skip layer differs from that of the input data, a convolutional layer, which is with the unit filter and the unit stride but the corresponding number of the feature maps, is added to the input data [9], [10]. After that, the CNN is trained by the Stochastic Gradient Descent (SGD) algorithm [46] on the training data by using the given GPU (line 4), and the classification accuracy is calculated on the validation data (line 5). Note that, the use of the softmax classifier and SGD training method are based on the conventions of the deep learning community. When the training phase is finished, the best classification accuracy on the validation data is set as the fitness of $individual$ (line 10). Finally, the identifier and fitness of $individual$ are associated and put into $Cache$ (line 11).

Next, the reasons for designing such an asynchronous and a cache components are given. In summary, because the training on CNNs is very time-consuming, ranging from several hours to even several months depending on the particular architecture, they are designed to speed up the fitness evaluation in the proposed algorithm. Specifically, the asynchronous component is a parallel computation platform based on GPUs. Due to the computational nature of calculating the gradients, deep learning algorithms are typically placed on GPUs to speed up the training [47]. Indeed, existing deep learning libraries, such as Tensorflow [48] and PyTorch [49], support the calculation on multiple GPUs. However, their parallel calculations are based on the data-parallel and model-parallel pipelines. In the data-parallel pipeline, the input data is divided into several smaller groups, and each group is placed on one GPU for calculation. The reason is that the limited memory of one GPU cannot effectively handle the whole data at the same time. In the model-parallel pipeline, a model is divided into several smaller models, and each GPU carries one smaller model. The obvious reason is the limited computational capability on one GPU cannot run a whole model. However, the designed parallel pipeline obviously does not fall into either of the pipelines, but at a higher level. Hence, such an asynchronous component is designed to make full use of the GPU computational resource, especially for the population-based algorithms. Furthermore, the asynchronous component is widely used in solving a large problem, if the problem can be divided into several independent sub-problems. By parallelly performing these sub-problems in different computational platforms, the total processing time of the whole problem is consequently shortened. In the past, evolutionary algorithms are typically used to solve the problems of which the fitness evaluation is not time-consuming[1], and there is no critical need in developing such asynchronous components. Occasionally, they just use the built-in components based on the adopted programming languages. However, almost all such built-in components are based on CPUs, and they cannot effectively train deep neural networks, mainly because the acceleration platform for neural networks are based on GPUs. Furthermore, the fitness evaluation of each individual is independent, which just satisfies the scenario of using this technique. Motivated by the reasons described above, such an asynchronous component is designed in the proposed algorithm. The cache component is also used to speed up the fitness evaluation, which is based on the considerations of: 1) the individuals surviving into the next generation do not need to evaluate the fitness again if its architecture is not changed, and 2) the architecture, which has been evaluated, could be regenerated with the mutation and crossover operations in another generation.

3.4 Offspring Generating

The details of generating the offspring are shown in Algorithm 5 which is composed of two parts. The first is the crossover (lines 1-18) and the second is the mutation

1. Although there is a type of computationally expensive problems, their fitness is commonly calculated by a surrogate model to bypass the direct fitness evaluation.

Algorithm 5: Offspring Generating

Input: The population P_t containing individuals with fitness, the probabity for crossover operation p_c, the probability for mutation operationp_m, the mutation operation list l_m, the probabilities of selecting different mutation operations p_l.
Output: The offspring population Q_t.

```
1  Q_t ← ∅;
2  while |Q_t| < |P_t| do
3      p_1 ← Randomly select two individuals from P_t, and from the two then select the one with the better fitness;
4      p_2 ← Repeat Line 3;
5      while p_2 == p_1 do
6          Repeat Line 4;
7      end
8      r ← Randomly generate a number from (0, 1);
9      if r < p_c then
10         Randomly choose a point in p_1 and divide it into two parts;
11         Randomly choose a point in p_2 and divide it into two parts;
12         o_1 ← Join the first part of p_1 and the second part of p_2;
13         o_2 ← Join the first part of p_2 and the second part of p_1;
14         Q_t ← Q_t ∪ o_1 ∪ o_2;
15     else
16         Q_t ← Q_t ∪ p_1 ∪ p_2;
17     end
18 end
19 foreach individual p in Q_t do
20     r ← Randomly generate a number from (0, 1);
21     if r < p_m then
22         i ← Randomly choose a point in p;
23         m ← Select one operation from l_m based on the probabilities in p_l;
24         Do the mutation m at the point i of p;
25     end
26 end
27 Return Q_t.
```

(lines 19-26). During the crossover operation, there will be totally $|P_t|$ offspring generated, where $|\cdot|$ measures the size of the collection. Specifically, two parents are selected first, and each is selected from two randomly selected individuals based on the better fitness (lines 3-7). This selection is known as the binary tournament selection [50], which is popularly used in GAs for single-objective optimization. Once the parents are selected, a random number is generated (line 8), determining whether the crossover will be done or not. If the generated number is not below the predefined crossover probability, these two parent individuals are put into Q_t as the offspring (line 16). Otherwise, each parent individual is randomly split into two parts, and the two parts from the two parent individuals are swapped to create two offspring (lines 10-14). During the mutation operation, a random number is generated first (line 20), and the mutation is performed on the current individual if the generated number is below p_m (lines 21-25). When mutating an individual, a position (denoted as i) is randomly selected from the current individual, and one particular mutation operation (denoted as m) is selected from the provided mutation list based on the probabilities defined in p_l. Then, m is performed on the position i. In the proposed algorithm, the available mutation operations defined in the mutation list are:

- Adding a skip layer with random settings;
- Adding a pooling layer with random settings;
- Removing the layer at the selected position;
- Randomly changing the parameter values of the building block at the selected position.

Next, the motivations of designing such a crossover operator and selecting a mutation operation based on a provided probability list are given. Firstly, the designed crossover operator is inspired by the one-point crossover [37] in traditional GAs. However, the one-point crossover was designed only for the individuals with the equal lengths. The designed crossover operator is used for the individuals with variable lengths. Although the designed crossover operator is simple, it dose improve the performance in discovering better architectures of CNNs, which will be experimentally proven in Section 5. Secondly, existing algorithms use an equal probability for choosing the particular mutation operation. In the proposed algorithm, the provided mutation operations are selected with different probabilities. Specifically, we provide a higher probability for the "adding a skip layer" mutation, which will encourage a higher probability to increase the depths of CNNs. For other mutation operations, we still employ the equal probabilities. The motivation behind this design is that a deeper CNN tends to have a more powerful capability as mentioned previously. Although the "adding a pooling layer" is also capable of increasing the depth of CNNs, the dimension of input data will be reduced to half by using one pooling layer, which results in the unavailability of the discovered CNN. To this end, we do not put a higher probability on it.

3.5 Environmental Selection

Algorithm 6 shows the details of the environmental selection. Firstly, $|P_t|$ individuals are selected from the current population ($Q_t \cup P_t$) by using the binary tournament selection, and then these selected individuals are placed into the next population (denoted P_{t+1}) (lines 2-6). Secondly, the best individual is selected to check whether it has been placed into P_{t+1}. If not, it will replace the worst individual in P_{t+1} (lines 7-10).

In principle, only selecting the top $|P_t|$ best individuals for the next generation could cause the premature convergence phenomenon [51], which will lead the algorithm trapped into a local optimum [23], [52]. If we do not explicitly select the best individuals for the next generation, the algorithm will not converge. In principle, a desirable population should contain not only good but also the relatively bad ones for enhancing the diversity [53], [54]. To this end, the binary tournament selection is typically used for such a purpose [50], [55]. However, only using the binary tournament selection may miss the best individual, resulting

Algorithm 6: Environmental Selection

Input: The parent population P_t, the offspring population Q_t
Output: The population for the next generation P_{t+1}.

```
P_{t+1} ← ∅;
while |P_{t+1}| < |P_t| do
    p_1, p_2 ← Randomly select two individuals from Q_t ∪ P_t;
    p ← Select the one who has a better fitness from {p_1, p_2};
    P_{t+1} ← P_{t+1} ∪ p;
end
p_best ← Find the individual with the best fitness from Q_t ∪ P_t;
if p_best is not in P_{t+1} then
    Replace the one who has the worst fitness in P_{t+1} by p_best;
end
Return P_{t+1}.
```

in the algorithm not moving towards a better direction of the evolution. Hence, we explicitly add the best individual into the next population, which is regarded as an elitism strategy in evolutionary algorithms [56].

4 Experimental Designs

In order to evaluate the performance of the proposed algorithm, a series of the experiments has been conducted on image classification tasks. Specifically, the peer competitors chosen to compare with the proposed algorithm are introduced in Subsection 4.1. Then, the used benchmark datasets are detailed in Subsection 4.2. Finally, the parameter settings of the proposed algorithm are shown in Subsection 4.3.

4.1 Peer Competitors

In order to show the effectiveness and efficiency of the proposed algorithm, the state-of-the-art algorithms are selected as the peer competitors to the proposed algorithm. Particularly, the peer competitors are chosen from three different categories.

The first refers to the state-of-the-art CNNs that are manually designed, including ResNet [9], DenseNet [10], VGG [7], Maxout [57], Network in Network [58], Highway Network [59] and All-CNN [60]. Specifically, two versions of ResNet, i.e., the ResNet models with a depth of 101 and $1,202$, are used since both of them have shown promising performance. For the convenience, they are named ResNet (depth=101) and ResNet (depth=$1,202$), respectively. Note that, most algorithms from this category are the champions in the large-scale visual recognition challenges in recent years [61].

The second and the third contain the algorithms discovering the best architectures of CNNs using a semi-automatic way and an automatic manner, respectively. Specifically, Genetic CNN [13], Hierarchical Evolution [15], EAS [18] and Block-QNN-S [21] belong to the second category, while Large-scale Evolution [14], CGP-CNN [16], NAS [18] and MetaQNN [19] fall into the third category. In addition, algorithms from these two categories are all proposed in the last two years.

Note that, the term "semi-automatic" means that expertise in CNNs must be provided before we use the corresponding architecture discovering algorithms. For example, EAS works on a base CNN which has already shown a good classification accuracy for the given data and QNN presumes a predefined big CNN where its discovered CNNs are embedded. This is opposite to the term "automatic" that no expertise in CNNs is needed when using the algorithms from this category. Obviously, the proposed algorithm in this paper is automatic.

4.2 Benchmark Datasets

The CIFAR10 and CIFAR100 benchmark datasets [22] are chosen as the image classification tasks in the experiments. The reasons for choosing them are that 1) both datasets are challenging in terms of the image sizes, categories of classification and the noise as well as rotations in each image; and 2) they are widely used to measure the performance of deep learning algorithms, and most of the chosen compared algorithms have publicly reported their classification accuracy on them.

Specifically, the CIFAR10 dataset is an image classification problem for recognizing 10 classes of natural objects, such as air planes and birds. It consists of $60,000$ RGB images in the dimension of 32×32. In addition, there are $50,000$ images and $10,000$ images in the training dataset and the testing dataset, respectively. Each category has the equal number of images. The CIFAR100 dataset is similar to CIFAR10, except it has 100 classes. The objects to be classified in these benchmark datasets typically occupy different areas of the whole image, and their positions are not the same in different images. Their variations are also challenging for the classification algorithms.

In the experiments, the training images are split into two parts. The first part accounts for 90% to serve as the training dataset for training the individuals, while the remaining images serve as the validation dataset for the fitness evaluation. In addition, the images are augmented during the training phases. In order to do a fair comparison, we employ the same augmentation routine as those often used in peer competitors, i.e., each direction of one image is padded by four zeros pixels, and then an image with the dimension of 32×32 is randomly cropped, finally, a horizontal flip is randomly performed on the cropped image with a probability of 0.5 [9], [10].

To date, most architecture discovering algorithms do not perform experiments on the CIFAR100 dataset due to its large number of classes. In order to show the superiority of our proposed algorithm over the peer competitors, we perform experiments on CIFAR100 and report the results.

4.3 Parameter Settings

As have been discussed, the main objective in this paper is to design an automatic architecture discovering algorithm for researchers without domain expertise in CNNs. To further improve applicability of the proposed algorithm, we design it in a way that potential users are not required to have expertise in evolutionary algorithms either. Hence,

TABLE 1
The comparisons between the proposed algorithm and the state-of-the-art peer competitors in terms of the classification accuracy (%), number of parameters and the taken GPU days on the CIFAR10 and CIFAR100 benchmark datasets.

		CIFAR10	CIFAR100	# Parameters	GPU days	Manual assistance?
state-of-the-art CNNs	ResNet (depth=101)	93.57	74.84	1.7M	–	completely need
	ResNet (depth=1,202)	92.07	72.18	10.2M	–	completely need
	DenseNet	94.17	76.58	27.2M	–	completely need
	VGG	93.34	71.95	20.04M	–	completely need
	Maxout	90.70	61.40	–	–	completely need
	Network in Network	91.19	64.32	–	–	completely need
	Highway Network	92.40	67.66	–	–	completely need
	All-CNN	92.75	66.29	1.3M	–	completely need
semi-automatic algorithms	Genetic CNN	92.90	70.97	–	17	partially need
	Hierarchical Evolution	96.37	–	–	300	partially need
	EAS	95.77	–	23.4M	10	partially need
	Block-QNN-S	95.62	79.35	6.1M	90	partially need
automatic algorithms	Large-scale Evolution	94.60	–	5.4M	2,750	completely not need
	Large-scale Evolution	–	77.00	40.4M	2,750	completely not need
	CGP-CNN	94.02	–	1.68M	27	completely not need
	NAS	93.99	–	2.5 M	22,400	completely not need
	Meta-QNN	93.08	72.86	–	100	completely not need
	CNN-GA	95.22	–	2.9M	35	completely not need
	CNN-GA	–	77.97	4.1M	40	completely not need

we simply set the parameters of the proposed algorithm based on the conventions. Specifically, the probabilities of crossover and mutation are set to 0.9 and 0.2, respectively, as suggested in [12]. During the training of each individual, the routine in [9] is employed, i.e., the SGD with the learning rate of 0.1 and the momentum of 0.9 are used to train 350 epochs and the learning rate is decayed by a factor of 0.1 at the 1-st, 149-th and 249-th epochs. Indeed, most peer competitors are also based on this training routine. When the proposed algorithm terminates, we choose the one with the best fitness value, and then train it up to 350 epochs on the original training dataset. At last, the classification accuracy on the test dataset is tabulated to compare with peer competitors. In addition, the available numbers of feature maps are set to $\{64, 128, 256\}$ based on the settings employed by the state-of-the-art CNNs. As for the probabilities of the four mutation operations shown in Subsection 3.4, we set the normalized probability of increasing the depth to 0.7, while others share the equal probabilities. Theoretically, any probability can be set for the adding mutation by keeping it higher than that of others. In addition, the population size and the number of generations are all set to 20, and the similar settings are also employed by the peer competitors. In theory, the larger population size and larger maximal generation number should result in a better performance, but also cost more computational resources. However, larger settings are not investigated in this paper since the current settings have easily outperformed most of the peer competitors based on the results shown in Table 1.

5 Experimental Results

Because the state-of-the-art CNNs in the first category of peer competitors are hand-crafted, we mainly compare the classification accuracy and the number of parameters. For the algorithms in other two categories, we compare the "GPU days" used to discover the corresponding CNN, in addition to the classification accuracy and the number of parameters. Particularly, the unit GPU day means the algorithm has performed one day on one GPU when the algorithm terminates, which reflects the computational resource consumed by these algorithms. For the convenience of summarizing the comparison results, we use the name of the architecture discovering algorithm as the name of the discovered CNN when comparing the classification accuracy and the number of parameters between peer competitors. For example, the proposed algorithm is named CNN-GA which is an architecture discovering algorithm, so its discovered CNN is titled as CNN-GA when comparing it to the chosen state-of-the-art CNNs.

The comparison results between the proposed algorithm and the peer competitors are shown in Table 1. In Table 1, the peer competitors are grouped into three different blocks based on the categories defined in Subsection 4.1, and the first column shows the category names. As a supplement, the last column also provides the information regarding how much manual assistance the corresponding CNN requires during discovering the architectures of CNNs. Additionally, the second column denotes the names of the peer competitors. The third and fourth columns refer to the classification accuracy on the CIFAR10 and CIFAR100 datasets, while the fifth column displays the numbers of parameters in the corresponding CNNs. The sixth column shows the used GPU days which are only applicable to the semi-automatic and automatic algorithms. The symbol "–" implies there is no result publicly reported by the corresponding algorithm. For ResNet (depth=101), ResNet (depth=1,202), DenseNet, VGG, All-CNN, EAS and Block-QNN-S, they have the same number of parameters on both the CIFAR10 and CIFAR100 datasets, respectively, which is caused by the similar CNNs achieving the best classification accuracy on both datasets. Note that, the results of the peer competitors shown in Table 1 are all from their respective

seminal papers[2]. For the proposed algorithm, the best CNN discovered by CNN-GA is selected from the population in the last generation, and then trained independently for five runs. The best classification accuracy is selected from the five results to show in Table 1, which follows the conventions of its peer competitors [13], [14], [15], [16], [18], [19], [21].

For the peer competitors in the first category, CNN-GA obtains 3.15%, 1.05% and 1.88% improvements in terms of the classification accuracy on the CIFAR10 dataset over ResNet (depth=1,202), DenseNet and VGG, respectively, while using merely 28%, 11% and 14% of their respective parameters. The number of parameters in CNN-GA is more than that of ResNet (depth=101) and All-CNN on the CIFAR10 dataset, but CNN-GA shows the best classification accuracy among them. In addition, CNN-GA achieves the highest classification accuracy among Maxout, Network in Network and Highway Network on the CIFAR10 dataset. On the CIFAR100 dataset, CNN-GA employs 52%, 85% and 40% fewer parameters compared to ResNet (depth=1202), DenseNet and VGG, respectively, but even achieves 5.79%, 1.39% and 6.02% improvements over the respective classification accuracy. CNN-GA also outperforms ResNet (depth=101), Maxout, Network in Network, Highway Network and All-CNN in terms of the classification accuracy, although it uses a larger network than that of RestNet (depth=101) and All-CNN. In summary, CNN-GA achieves the best classification accuracy on both the CIFAR10 and CIFAR100 datasets among the state-of-the-art CNNs manually designed. Additionally, it also employs a much fewer number of parameters than most of the state-of-the-art CNNs in the first category.

For the peer competitors in the second category, the classification accuracy of CNN-GA is better than that of Genetic CNN. CNN-GA shows 1.15% lower classification accuracy than that of Hierarchical Evolution; however, CNN-GA takes only one-tenth of the GPU days that that of Hierarchical Evolution. Compared to Block-QNN-S, CNN-GA shows slightly worse classification accuracy on the CIFAR10 and CIFAR100 datasets, while CNN-GA consumes about half of the number of the parameters and also half of the CPU days compared to those of Block-QNN-S. Furthermore, the classification accuracy of CNN-GA is competitive to that of EAS, and CNN-GA employs a significantly smaller number of parameters than that of EAS (87% fewer parameters used by CNN-GA than that of EAS). Although CNN-GA does not offer the best classification accuracy among the algorithms in this category, the algorithms in this category typically require extended expertise when they are used to solve real-world tasks. For example, EAS requires a manually-tuned CNN on the given dataset, and then EAS is used to refine the tuned CNN. If the tuned CNN is not with promising performance, the developed EAS would perform not well either at the end. In addition, the CNNs discovered by Hierarchical Evolution and Block-QNN-S cannot be directly used. They must be inserted into a larger CNN manually-designed in advance. If the larger network is not designed properly, the final performance of Hierarchical Evolution and Block-QNN-S would also perform poorly. The major advantage of CNN-GA, among the algorithms in this category, is its completely automatic nature.

For the peer competitors in the third category, CNN-GA performs better than Large-scale Evolution and CGP-CNN, and much better than NAS and Meta-QNN on the CIFAR10 dataset regarding the classification accuracy. Meanwhile, CNN-GA also shows a superiority of the classification accuracy over Large-scale Evolution and Meta-QNN on the CIFAR100 dataset. Furthermore, CNN-GA only has 2.9M parameters on the CIFAR10 dataset, which is almost half of those of Large-scale Evolution. On the CIFAR100 dataset, CNN-GA has 4.1M parameters, which saves 90% parameters compared to those of Large-scale evolution which requires 40.4M parameters [14]. Furthermore, CNN-GA takes only 35 GPU days on the CIFAR10 dataset and 40 GPU days on the CIFAR100 dataset, while Large-scale Evolution consumes $2,750$ GPU days on the CIFAR10 dataset and another $2,750$ GPU days on the CIFAR100 dataset. Even more, NAS employs $22,400$ GPU days on the CIFAR10 dataset. Based on the number of parameters and GPU days taken, it can be summarized that CNN-GA shows a promising performance by using 90% simpler architectures on 99% less computational resource than the average numbers of competitors in this category[3].

In summary, CNN-GA outperforms the state-of-art CNNs manually designed and automatic architecture discovering algorithms in terms of not only the classification accuracy, but also the number of parameters and the employed computational resource. Although the state-of-the-art semi-automatic architecture discovering algorithms show similar (to slightly better) classification accuracy to that of CNN-GA, CNN-GA is competitively automatic and does not require users to have any expertise in CNNs when solving real-world tasks.

6 Conclusions and Future Work

The objective of this paper is to propose an automatic architecture design algorithm for CNNs by using the GA (in short named CNN-GA), which is capable of discovering the best CNN architecture in addressing image classification problems. This goal has been successfully achieved by designing a new encoding strategy for the GA to encode arbitrary depths of CNNs, incorporating the skip connections to promote deeper CNNs to be produced during the evolution and developing a parallel as well as a cache component to significantly accelerate the fitness evaluation given a limited computational resource. The proposed algorithm is examined on two challenging benchmark datasets, and compared with 16 state-of-the-art peer competitors, including eight manually designed CNNs, four semi-automatic and four automatic algorithms discovering the architectures of CNNs. The experimental results show that CNN-GA outperforms

2. It is exceptional for VGG because it does not perform experiments on CIFAR10 and CIFAR100 in its seminal paper. Its results displayed in Table 1 is derived from [16].

3. The number is calculated by summing up the corresponding number of each peer competitor in this category, and then normalized by the numbers of available classification results. For example, the total GPU days of the peer competitors are $28,027$, and there are only six available classification results. Hence, the average GPU days are $4,671$. In the same way, the average GPU days took by CNN-GA are found to be 37.5.

all the manually-designed CNNs as well as the automatic peer competitors, and shows competitive performance with respect to semi-automatic peer competitors in terms of the best classification accuracy. The CNN discovered by CNN-GA has a much smaller number of parameters than those of most peer competitors. Furthermore, CNN-GA also employs significantly less computational resource than most automatic and semi-automatic peer competitors. Moreover, CNN-GA is completely automatic, and researchers can directly use it to address their own image classification problems whether or not they have expertise in CNNs or GAs.

In CNN-GA, two components have been designed to speed up the fitness evaluation, and much computational resource has been saved. However, the computational resource employed is still fairly large than those of GAs in solving traditional problems. In the field of solving expensive optimization problems, several algorithms based on evolutionary computation techniques have been developed. In future, we will place efforts on developing effective evolutionary computation methods to significantly speed up the fitness evaluation of CNNs.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[3] T. N. Sainath, A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for lvcsr," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 8614–8618.

[4] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, 2014, pp. 3104–3112.

[5] C. Clark and A. Storkey, "Training deep convolutional neural networks to play go," in *Proceedings of the 32nd International Conference on Machine Learning*, Lille, France, 2015, pp. 1766–1774.

[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of the 32nd International Conference on Machine Learning*, Lille, France, 2015.

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich *et al.*, "Going deeper with convolutions," in *Proceedings of 2015 IEEE Conference on Computer Vision and Pattern Recognition*, Boston, MA, USA, 2015, pp. 1–9.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition*, Las Vegas, NV, USA, 2016, pp. 770–778.

[10] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," in *Proceedings of 2017 IEEE Conference on Computer Vision and Pattern Recognition*, Honolulu, HI, USA, 2017, pp. 2261–2269.

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

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

[13] L. Xie and A. Yuille, "Genetic CNN," in *Proceedings of 2017 IEEE International Conference on Computer Vision*, Venice, Italy, 2017, pp. 1388–1397.

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

[15] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *Proceedings of 2018 Machine Learning Research*, Stockholm, Sweden, 2018.

[16] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the 2017 Genetic and Evolutionary Computation Conference*. Berlin, Germany: ACM, 2017, pp. 497–504.

[17] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[18] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proceedings of the 2017 International Conference on Learning Representations*, Toulon, France, 2017.

[19] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proceedings of the 2017 International Conference on Learning Representations*, Toulon, France, 2017.

[20] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proceedings of the 2018 AAAI Conference on Artificial Intelligence*, Louisiana, USA, 2018.

[21] Z. Zhong, J. Yan, and C.-L. Liu, "Practical network blocks design with q-learning," in *Proceedings of the 2018 AAAI Conference on Artificial Intelligence*, Louisiana, USA, 2018.

[22] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *online: http://www.cs.toronto.edu/kriz/cifar.html*, 2009.

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

[24] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic programming: an introduction*. Morgan Kaufmann San Francisco, 1998, vol. 1.

[25] C. Janis, "The evolutionary strategy of the equidae and the origins of rumen and cecal digestion," *Evolution*, vol. 30, no. 4, pp. 757–774, 1976.

[26] L. M. Schmitt, "Theory of genetic algorithms," *Theoretical Computer Science*, vol. 259, no. 1-2, pp. 1–61, 2001.

[27] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[28] Y. Sun, G. G. Yen, and Z. Yi, "IGD indicator-based evolutionary algorithm for many-objective optimization problems," *IEEE Transactions on Evolutionary Computation*, 2018, DOI:10.1109/TEVC.2018.2791283.

[29] ——, "Reference line-based estimation of distribution algorithm for many-objective optimization," *Knowledge-Based Systems*, vol. 132, pp. 129–143, 2017.

[30] ——, "Improved regularity model-based EDA for many-objective optimization," *IEEE Transactions on Evolutionary Computation*, 2018, DOI:10.1109/TEVC.2018.2794319.

[31] M. Jiang, Z. Huang, L. Qiu, W. Huang, and G. G. Yen, "Transfer learning based dynamic multiobjective optimization algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 4, pp. 501–514, 2018.

[32] M. Mitchell, *An introduction to genetic algorithms*. Cambridge, Massachusetts, USA: MIT press, 1998.

[33] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[34] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: continual prediction with lstm," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[35] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Advances in Neural Information Processing Systems*, Montral, Canada, 2015, pp. 2377–2385.

[36] A. E. Orhan and X. Pitkow, "Skip connections eliminate singularities," in *Proceedings of 2018 Machine Learning Research*, Stockholm, Sweden, 2018.

[37] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *Computer*, vol. 27, no. 6, pp. 17–26, 1994.

[38] Y. Sun, G. G. Yen, and Z. Yi, "Evolving unsupervised deep neural networks for learning meaningful representations," *IEEE Transactions on Evolutionary Computation*, 2018, DOI:10.1109/TEVC.2018.2808689.

[39] J. F. Miller and P. Thomson, "Cartesian genetic programming," in *European Conference on Genetic Programming*. Springer, 2000, pp. 121–132.

[40] D. M. Hawkins, "The problem of overfitting," *Journal of Chemical Information and Computer Sciences*, vol. 44, no. 1, pp. 1–12, 2004.

[41] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[42] R. Housley, "A 224-bit one-way hash function: Sha-224," RFC 3874, September 2004.

[43] N. M. Nasrabadi, "Pattern recognition and machine learning," *Journal of Electronic Imaging*, vol. 16, no. 4, p. 049901, 2007.

[44] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, FL, USA, 2011, pp. 315–323.

[45] S. Ioffe, "Batch renormalization: towards reducing minibatch dependence in match-normalized models," in *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017, pp. 1945–1953.

[46] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: tricks of the trade*. Springer, 2012, pp. 421–436.

[47] R. Helfenstein and J. Koko, "Parallel preconditioned conjugate gradient algorithm on gpu," *Journal of Computational and Applied Mathematics*, vol. 236, no. 15, pp. 3584–3590, 2012.

[48] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning." in *Operating Systems Design and Implementation*, vol. 16, 2016, pp. 265–283.

[49] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017. [Online]. Available: https://openreview.net/forum?id=BJJsrmfCZ

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

[51] Z. Michalewicz and S. J. Hartley, "Genetic algorithms + data structures = evolution programs," *Mathematical Intelligencer*, vol. 18, no. 3, p. 71, 1996.

[52] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine Learning*, vol. 3, no. 2, pp. 95–99, 1988.

[53] C. M. Anderson-Cook, "Practical genetic algorithms," p. 1099, 2005.

[54] S. Malik and S. Wadhwa, "Preventing premature convergence in genetic algorithm using dgca and elitist technique," *nternational Journal of Advanced Research in Computer Science and Software Engineering*, vol. 4, no. 6, 2014.

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

[56] D. Bhandari, C. Murthy, and S. K. Pal, "Genetic algorithm with elitist model and its convergence," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 10, no. 06, pp. 731–747, 1996.

[57] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *Proceedings of the 30th International Conference on Machine Learning*, Atlanta, Georgia, USA, Jun 2013, pp. 1319–1327.

[58] M. Lin, Q. Chen, and S. Yan, "Network in network," in *Proceedings of the 2014 International Conference on Learning Representations*, Banff, Canada, 2014.

[59] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," in *Proceedings of the 2015 International Conference on Learning Representations Workshop*, San Diego, CA, 2015.

[60] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: the all convolutional net," in *Proceedings of the 2015 International Conference on Learning Representations*, San Diego, CA, 2015.

[61] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[62] D. F. Williamson, R. A. Parker, and J. S. Kendrick, "The box plot: a simple visual method to interpret data," *Annals of Internal Medicine*, vol. 110, no. 11, pp. 916–921, 1989.