Automatically Designing CNN Architectures Using the Genetic Algorithm for Image Classification

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

***Abstract*—Convolutional neural networks (CNNs) have gained remarkable success on many image classification tasks in recent years. However, the performance of CNNs highly relies upon their architectures. For the most state-of-the-art CNNs, their architectures are often manually designed with expertise in both CNNs and the investigated problems. Therefore, it is difficult for users, who have no extended expertise in CNNs, to design optimal CNN architectures for their own image classification problems of interest. In this article, we propose an automatic CNN architecture design method by using genetic algorithms, to effectively address the image classification tasks. The most merit of the proposed algorithm remains in its "automatic" characteristic that users do not need domain knowledge of CNNs when using the proposed algorithm, while they can still obtain a promising CNN architecture for the given images. The proposed algorithm is validated on widely used benchmark image classification datasets, compared to the state-of-the-art peer competitors covering eight manually designed CNNs, seven automatic + manually tuning, and five automatic CNN architecture design algorithms. The experimental results indicate the proposed algorithm outperforms the existing automatic CNN architecture design algorithms in terms of classification accuracy, parameter numbers, and consumed computational resources. The proposed algorithm also shows the very comparable classification accuracy to the best one from manually designed and automatic + manually tuning CNNs, while consuming fewer computational resources.**

***Index Terms*—Convolutional neural networks (CNNs), evolutionary deep learning, genetic algorithms (GAs), neural-network architecture optimization.**

Manuscript received May 12, 2019; revised January 20, 2020; accepted March 25, 2020. Date of publication April 21, 2020; date of current version August 18, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61803277, in part by the Fundamental Research Funds for the Central Universities, in part by the National Natural Science Fund of China for Distinguished Young Scholar under Grant 61625204, and in part by the National Natural Science Fund under Grant 61806135 and Grant 61836011. This article was recommended by Associate Editor Y.-M. Cheung. *(Corresponding author: Gary G. Yen.)*

Yanan Sun and Jiancheng Lv are with the College of Computer Science, Sichuan University, Chengdu 610065, China (e-mail: ysun@scu.edu.cn; lvjiancheng@scu.edu.cn).

Bing Xue and Mengjie Zhang are with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand (e-mail: bing.xue@ecs.vuw.ac.nz; 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).

This article has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the author.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCYB.2020.2983860

I. 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]–[5]. It has been known that the performance of CNNs highly relies upon their architectures [2], [6]. To achieve promising performance, the architectures of state-of-the-art CNNs, such as GoogleNet [7], ResNet [8], and DenseNet [9], are all manually designed by experts who have rich domain knowledge from both investigated data and CNNs. Unfortunately, such domain knowledge is not necessarily held by each interested user. For example, users who are familiar with the data at hand do not necessarily have the experience in designing the architectures of CNNs, and vice versa. As a result, there is a surge of interest in automating the design of CNN architectures, allowing tuning skills of the CNN architectures to be transparent to the users who have no domain knowledge on CNNs. On the other hand, CNN architecture design algorithms can also spread the wide adoption of CNNs which, in turn, promotes the development of machine intelligence.

Existing CNN architecture design algorithms can be divided into two different categories, based on whether domain knowledge is required or not when using them. The first is the "automatic + manually tuning" CNN architecture designs, which imply that the manual tuning based on the expertise in designing CNN architectures is still required. This category covers the genetic CNN method (Genetic CNN) [10], the hierarchical representation method (Hierarchical Evolution) [11], the efficient architecture search method (EAS) [12], the block design method (Block-QNN-S) [13], and the advanced neural architecture search method (NSANet) [14]. The other is the automatic CNN architecture designs, which do not require any manual tuning from users when using them. The large-scale evolution method (Large-scale Evolution) [15], the Cartesian genetic programming method (CGP-CNN) [16], the neural architecture search method (NAS) [17], and the metamodeling method (MetaQNN) [18] belong to this category. Due to the extra benefits of manual expertise in CNNs, it is natural that the automatic + manually tuning designs often show slightly better performance than the automatic designs. However, the significant superiority of the automatic designs is their manually tuning-free characteristic, which is much preferred by users without any domain knowledge of CNNs. For example, the NASNet and the Large-scale Evolution algorithms,

which are from the automatic + manually tuning and the automatic categories, respectively, achieved the classification accuracies of 96.27% and 94.60% on the CIFAR10 benchmark dataset [19], respectively. However, a CNN framework architecture must be provided to NSANet because NSANet only automated partial cells of the framework. Obviously, if the framework architecture is not well designed, the resulting CNN will not have promising performance. When using the Large-scale Evolution method, the users just directly perform it on the given data, and finally, a promising CNN architecture is obtained. To this end, the automatic CNN architecture designs should be more welcomed because the majority of CNN users have no extensive domain knowledge of the CNN architecture design.

On the other hand, based on the adopted techniques, CNN architecture designs can also be classified into the evolutionary algorithm-based ones and the reinforcement learning-based ones. Specifically, Genetic CNN, Large-scale Evolution, Hierarchical Evolution, and CGP-CNN are based on evolutionary algorithms [20], following the standard flow of an evolutionary algorithm to heuristically discover the optimal solution while NAS, MetaQNN, EAS, Block-QNN-S, and NASNet are based on reinforcement learning [21], resembling those based on evolutionary algorithms, in addition to the employed heuristic nature utilizing the reward–penalty principle of reinforcement learning. Experimental evidence shows that the reinforcement learning-based designs often require more extensive computational resources than the ones based on the evolutionary algorithms [22]. For example, the NAS method consumed 800 graphic processing units (GPUs) in 28 days to find the promising CNN architecture on the CIFAR10 dataset, while the Genetic CNN consumed only 17 GPUs in one day on the same dataset providing similar performance. To this end, the evolutionary algorithms-based CNN architecture designs are much preferred because extensive computational resources are not necessarily available to every interested user.

The evolutionary algorithm [20] is a class of population-based metaheuristic optimization paradigms 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]–[31]. It has also been recognized that GAs are capable of generating high-quality optimal solutions by using bioinspired operators, that is, mutation, crossover, and selection [32]. Commonly, the operators need to be carefully designed for the specific problems at hand. For example, Liu *et al.* [33] designed a one-point crossover operator and a novel mutation operator for wideband code-division multiple access (WCDMA) network planning. In addition, Nag and Pal [34] developed a new mutation operator to exploit the fitness and the unfitness for the feature selection via an ensemble method for the feature selection and classification. Considering both characteristics of automatic and evolutionary algorithm-based CNN architecture designs, we propose an effective and efficient algorithm by using GA, in short, called as CNN-GA, to automatically discover the best architectures of CNNs, so that the discovered CNN can be directly used without any manual tunings. Obviously, CNN-GA is an automatic CNN architecture design algorithm. Please note that the automatic and automatic + manually tuning are discussed from the users' view but not the developers' view. In contrast, adequate domain knowledge should be encouraged in developing high-performance CNN architecture design algorithms. This effort can be easily understood by the analogy to the design of Windows operating system by scientists from Microsoft: the scientists should use their professional knowledge as much as possible in designing a user-friendly operating system so that the users are allowed to effectively work on the computers, even without extensive knowledge on the operating system.

The novelty of the proposed algorithm lies in its complete automation in discovering the best CNN architectures, and not requiring any manual intervention during the evolutionary search. The contributions of the proposed CNN-GA algorithm are summarized as follows.

1) GAs often employ the fixed-length encoding strategy because of the crossover operators primitively designed for the individuals having the same lengths. In this case, the length of the encoding must be specified beforehand. Ideally, the length should be the optimal CNN depth that is mostly unknown in advance. As a result, the specified number may be incorrectly estimated, resulting in an ineffective architecture design. Although many researchers have developed the variable-length encoding strategy independently, the resulting CNN architecture is not optimal because the crossover operator is not redesigned accordingly. In this article, we have proposed a variable-length encoding strategy and the corresponding crossover operator to address both aforementioned issues efficiently and effectively.
2) Most existing CNN architecture algorithms are designed based on the basic components or the well-constructed blocks of CNNs. In practice, both designs often generate the ineffective CNN architectures and the complex CNN architectures having poor generalization ability, respectively. In this article, the skip connections are incorporated into the proposed algorithm to deal with complex data by avoiding the vanishing gradient (VG) problems [35]. On the one hand, this design can reduce the search space so that the best performance can be easily identified. On the other hand, compared to other algorithms with similar performance, the architectures evolved by the proposed algorithm are much simpler.
3) In order to speed up the CNN architecture design and provide the users with an optimal CNN architecture within an acceptable time, most of the existing algorithms employed extensive computational resources. Particularly, the algorithms employed the data parallelism strategy which is not efficient. The main reason is that most architectures are median scale and do not need to run on the extensive computational resource. Otherwise, the communication consumption between different computational nodes will take up most of the computational cost. In the proposed algorithm, an asynchronous computational component is developed to

make full use of the given computational resources to accelerate the evaluation of the fitness of the individuals, while a cache component is employed to further reduce the fitness evaluation time for the entire population.

The remainder of this article is organized as follows. First, related works and background are presented in Section II. Then, the details of the proposed algorithm are documented in Section III. Next, the experimental designs and experimental results as well as the analysis are shown in Sections IV and V, respectively. Finally, conclusions and future works are outlined in Section VI.

II. Literature Review

In this section, CNNs and skip connections, which are considered the background of the proposed algorithm, are introduced to help readers better understand the related works and the proposed algorithm. Then, the relevant work in discovering the architectures of CNNs is reviewed.

A. Background

1) Convolutional Neural Networks: In this section, we mainly introduce the building blocks of CNNs, that is, 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. 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 entire image has been scanned. The set of filter outputs forms a new matrix called the feature map. The horizontal and vertical step sizes are called the width and height of a stride. The exact number of feature maps used is a parameter in the architecture of the corresponding CNN. In addition, two convolutional operations are applied: 1) the *same* convolutional operation which pads zeros to the input data when there is not enough area for the filter to overlap and 2) 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. In the proposed algorithm, the fully connected layers are discarded, and the justifications are given in Section III-B.

2) Skip Connections: The skip connections refer to those connecting the neurons of the layers that are not adjacent. The skip connection was first introduced in [35] as a gate mechanism, effectively training a recurrent neural network with long- and short-term memory [36] and avoiding the VG problems [35]. Specifically, the VG problems refer to the gradient becoming either very small or an explosion during backpropagation training in a deep neural network, and are the main obstacle to effectively train *deep* neural networks. The skip connections were experimentally proven to be able to train very deep neural networks [37]. Indeed, the promising performance of ResNet [8], 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 Nth building block, and the symbol "$\oplus$" refers to the elementwise addition.

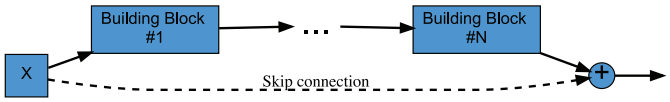

Fig. 1. Example of using the skip connection.

Over the past few years, an increasing number of researchers have 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 [38]. Among the existing theoretical evidence, skip connections defying the VG problems receive the most recognition [35], [36]. Because the skip connections shorten the number of layers of backpropagation, the VG 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.

B. Related Work

As the based work of the proposed algorithm, the Genetic CNN method [10] is discussed in this section.

In Genetic CNN, encoding the entire architecture of a CNN consists of multiple stages. In each stage, a small architecture is encoded, and then multiple, different small architectures are stacked to give the entire architecture. Just like the NSANet method, a CNN architecture framework is provided to Genetic CNN, while the small architectures replace the convolutional layers in the provided framework to form the final CNN. These small architectures are called cell in Genetic CNN, and each cell is designed in each stage of Genetic CNN. In each stage, multiple predefined building blocks of CNNs are ordered, and their connections are encoded. For ordering these building blocks, the first and the last building blocks are manually specified, and the remainings are all convolutional layers with the same settings. In order to encode the connections between these ordered building blocks, a binary string encoding method [39] is used. Suppose there are four building blocks, a string of "111-10-0" implies that the first building block has the connections to all other building blocks. The second one has no connection to the fourth building block, which is the same as the third one. Furthermore, the number of stages is manually predefined. Clearly observed from this

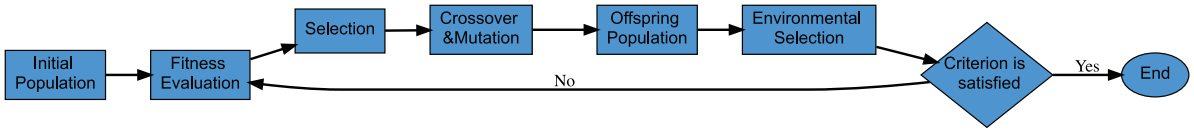

Fig. 2. Flowchart of a GA.

encoding strategy, multiple manual interventions are required, such as the numbers of building blocks in each stage and total stages. Because these numbers are related to the depth of the discovered CNN, domain expertise in CNNs is strongly needed to predefine these numbers for proper depth of the CNN. In addition, the parameters of the building blocks in each stage cannot be changed in Genetic CNN. Indeed, this encoding strategy can be viewed at only discovering the connections of a given CNN, and cannot change the depth of the given CNN. Note that no speeding up technique was designed in Genetic CNN, resulting in the experiments not applicable to complex datasets such as CIFAR100.

III. Proposed Algorithm

In this section, we first present the framework of the proposed algorithm in Section III-A and then detail the main steps in Sections III-B–III-E. To help readers better understand the proposed algorithm, we will not only document the details of each main step but also provide the analysis for given designs.

A. Algorithm Overview

Algorithm 1 shows the framework of the proposed algorithm. Specifically, by giving a set of predefined building 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 generated a new offspring by the genetic operators, including the crossover and mutation operators (line 5). Then, a population of individuals surviving into the next generation is selected by the environmental selection from the current population (line 6). Specifically, the current population consists 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).

Algorithm 1: 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 using **the proposed variable-length encoding strategy**;
2 $t \leftarrow 0$;
3 **while** $t <$ *the maximal generation number* **do**
4 Evaluate the fitness of each individual in P_t using **the proposed acceleration components**;
5 $Q_t \leftarrow$ Generate offspring from the selected parent individuals using **the proposed mutation and the crossover operators**;
6 $P_{t+1} \leftarrow$ Environmental selection from $P_t \cup Q_t$;
7 $t \leftarrow t + 1$;
8 **end**
9 **Return** the individual having the best fitness in P_t.

Please note that GAs only provide a unified framework to solve optimization problems with their inherent biological mechanism. When GAs are used in practice, their components regarding the biological mechanism must be specifically designed based on the particular problems to be solved. In the proposed algorithm, we carefully design the variable-length encoding strategy, acceleration components, and genetic operators (they are also highlighted in bold in Algorithm 1), to assure the effectiveness and efficiency of the proposed algorithm in designing CNN architectures.

B. Population Initialization

As introduced in Section II, a CNN consists 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, called 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 section). In summary, only the skip layers and the pooling layers are used to construct a CNN in the proposed encoding strategy.

Specifically, a skip layer consists 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. As introduced above, the

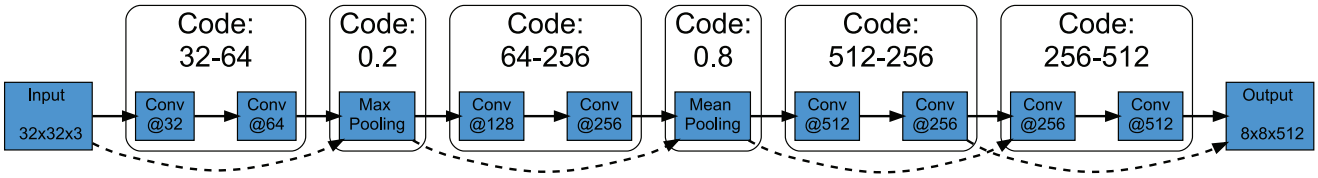

Fig. 3. Example of the proposed encoding strategy representing a CNN. This CNN consists of four skip layers and two pooling layers. The code encoding this CNN consists of the codes representing each layer. For each skip layer, its codes are the number of feature maps of the two convolutional layers within this skip layer; while the code for each pooling layer is the pooling type. We use a number between $(0, 0.5)$ to represent the max-pooling type, and a number between $[0.5, 1)$ to represent the mean-pooling type. The numbers above each layer are the code of the corresponding layer. So the code representing this CNN is "32-64-0.2-64-256-0.8-512-256-256-512."

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, respectively. Particularly, the filter and stride sizes are set to 3×3 and 1×1, respectively, and only the *same* convolutional operation 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 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 section.

Algorithm 2 shows the details of the population initialization. Briefly, T individuals are initialized with the same ways, 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 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 layer by setting its *type* property to 1. Otherwise, this node represents a pooling layer by setting *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 7–11). Otherwise, the pooling type is determined by the probability of tolling 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).

An example of the proposed encoding strategy encoding a CNN is shown in Fig. 3. This CNN consists of four skip layers and two pooling layers. The code representing one skip layer is the string of the feature map numbers of the corresponding convolutional layers within the same skip layer; while that for a pooling layer is a number representing the pooling type. Particularly, a random number between $(0, 0.5)$ represents a max-pooling layer, while that between $[0.5, 1)$ represents a mean-pooling layer. The code representing the entire CNN is the sequential string connection of codes representing the layers. As shown in this example where the code of each layer is listed above itself, the code of the entire CNN is "32-64-0.2-64-256-0.8-512-256-256-512" to represent a CNN with a depth of 10.

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.
```

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. Specifically, multiple fully connected layers are typically added to the tail of a CNN. However, the fully connected layer easily results in the overfitting phenomenon [40] due to its dense connection [41]. To reduce the overfitting, the dropout [41] randomly removing a part of the connections is commonly used. However, each dropout will introduce one parameter. Only a properly specified parameter can lead to the promising performance of the corresponding CNN. Meanwhile, the number of fully connected layers and

Algorithm 3: Fitness Evaluation

Input: The population P_t of the individuals to be evaluated, the image dataset for classification.
Output: The population P_t of the individuals with their 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
               available GPU (details shown in
               Algorithm 4);
12         end
13     end
14 end
15 Return P_t.
```

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 fitness evaluation data $D_{fitness}$ 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_fitness;
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 number of neurons in each fully connected layer are also two parameters hard to tune. If the fully connected layers are incorporated into the proposed encoding strategy, the search space will substantially enlarge, increasing the difficulty of finding 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 proved in [9], [11], [15], and [42]. However, the sizes of feature maps in each skip layer of ResNet are set to be equal. In the proposed encoding strategy, the sizes of feature maps can be unequal, which is more flexible. Furthermore, setting the convolutional operation to *same* and using the 1×1 stride is to make the dimension of the input data remain the same, which is more flexible for such an automatic design because 1×1 stride does not change the image size. As the settings of filter and kernel sizes as well as the stride size in the pooling layers, they are all based on the designs of existing handcrafted CNNs [9], [42]. 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 V.

C. Fitness Evaluation

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 discover, 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, that is, 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 gotten 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 [43], 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 do not need to wait for the fitness evaluation for the next individual until the fitness evaluation of the current one finishes, but place the next individual on an available GPU immediately.

The details of evaluating the fitness of one individual are shown in Algorithm 4. First, 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 [44] is used, and the particular number of classes is determined by the given image dataset. When decoding a CNN, a rectifier activation function [45] followed by a batch normalization [46] operation is added to the output of the convolutional layer, which is based on the conventions of modern CNNs [8], [9]. 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 special number of the feature maps, is added to the input data [8], [9]. After that, the CNN is trained by the stochastic gradient descent (SGD) algorithm [47] on the training data by using the given GPU (line 4), and the classification accuracy is calculated on the fitness evaluation data (line 5). Note that the use of the softmax classifier and

SGD training method is based on the conventions of the deep-learning community. When the training phase is finished, the best classification accuracy on the fitness evaluation 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, varying 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 [48]. Indeed, existing deep-learning libraries, such as Tensorflow [49] and PyTorch [50], 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 are divided into several small groups, and each group is placed on one GPU for the calculation. The reason is that the limited memory of one GPU cannot effectively handle the entire data at the same time. In the model-parallel pipeline, a model is divided into several small models, and each GPU carries one small model. The reason is that the limited computational capability on one GPU cannot run the entire model. However, the designed parallel pipeline obviously does not fall into either of the pipelines, but a higher level based on them. 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 subproblems. By parallel performing these subproblems in different computational platforms, the total processing time of the entire 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 high 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 is based on GPUs. Furthermore, the fitness evaluation of each individual is independent, which just satisfies the scene 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 following considerations: 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. Note that the weight inheriting of Large-scale Evolution cannot work for the second consideration.

Commonly, a cache system should seriously treat its size and provide the details to discuss the conflicting problem resulted from the duplicate keys. In the proposed algorithm, none is necessary to be investigated. First, the cache component is similar to a map data structure where each record in this component is a string combining the identifier and the fitness value of a CNN. For example, a record such as "identifier1 = 98.12" denotes that the identifier is "identifier1" and its fitness value is "98.12." Second, as we have highlighted, the identifier is calculated by the 224-hash code that can generate 2^{224} different identifiers. In practice, the CNN architecture algorithms, including the proposed algorithm, only evaluate thousands of CNNs. Obviously, we do not need to consider the conflicting problem because the conflicting problem will not happen. Third, the 224-hash code implementation used for the proposed algorithm will generate the identifier having the length of 32, and the fitness value is the classification accuracy that is denoted by a string having the length of 4. Totally, each record in the cache component is a string having the length of 37 occupying 37 B with the UTF-8 file encoding. Obviously, the cache file will only occupy very little disk space even though there are thousands of records. Therefore, we do not need to concern about the size of the cache component.

D. Offspring Generating

The details of generating the offspring are shown in Algorithm 5, which consists of two parts. The first is crossover (lines 1–18) and the second is mutation (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 [51], which is popular 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 formulate two offspring (lines 10–14). During the mutation operation, a random number is generated first (line 20), and the mutation operation is performed on the current individual if the generated number is below than 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 algorithms, the available mutation operations defined in the mutation list are as follows.

1) Adding a skip layer with random settings.
2) Adding a pooling layer with random settings.

[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.

$Q_t \leftarrow \emptyset$;
while $|Q_t| < |P_t|$ **do**
  $p_1 \leftarrow$ Randomly select two individuals from P_t, and from the two then select the one with the better fitness;
  $p_2 \leftarrow$ Repeat Line 3;
  while $p_2 == p_1$ **do**
    Repeat Line 4;
  end
  $r \leftarrow$ Randomly generate a number from (0, 1);
  if $r < p_c$ **then**
    Randomly choose a point in p_1 and divide it into two parts;
    Randomly choose a point in p_2 and divide it into two parts;
    $o_1 \leftarrow$ Join the first part of p_1 and the second part of p_2;
    $o_2 \leftarrow$ Join the first part of p_2 and the second part of p_1;
    $Q_t \leftarrow Q_t \cup o_1 \cup o_2$;
  else
    $Q_t \leftarrow Q_t \cup p_1 \cup p_2$;
  end
end
foreach *individual p in* Q_t **do**
  $r \leftarrow$ Randomly generate a number from (0, 1);
  if $r < p_m$ **then**
    $i \leftarrow$ Randomly choose a point in p;
    $m \leftarrow$ Select one operation from l_m based on the probabilities in p_l;
    Do the mutation m at the point i of p;
  end
end
Return Q_t.

3) Remove the layer at the selected position.
4) Randomly changing the parameter values of the building block at the selected position.

Next, the motivation of designing such a crossover operator and selecting a mutation operation based on a provided probability list is given. First, the designed crossover operator is inspired by the one-point crossover [39] in traditional GAs. However, the one-point crossover was designed only for the individuals with equal lengths. The designed crossover operator is used for the individuals with unequal lengths. Although the designed crossover operator is simple, it does improve the performance in discovering the architectures of CNNs, which will be experimentally proved in Section V. Second, existing algorithms use an equal probability for choosing the particular

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}.

$P_{t+1} \leftarrow \emptyset$;
while $|P_{t+1}| < |P_t|$ **do**
  $p_1, p_2 \leftarrow$ Randomly select two individuals from $Q_t \cup P_t$;
  $p \leftarrow$ Select the one who has a better fitness from $\{p_1, p_2\}$;
  $P_{t+1} \leftarrow P_{t+1} \cup p$;
end
$p_{best} \leftarrow$ Find the individual with the best fitness from $Q_t \cup P_t$;
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}.

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 have 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 will 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 the one pooling layer, which results in the unavailability of the discovered CNN. To this end, we do not put a higher probability on it.

Please note that the optimal depth of the CNN is achieved by the proposed mutation operators. As shown above, there are four different types of mutation operators designed in the proposed algorithm. The optimal depth can be found through the first three mutation operators. Particularly, the first two operators provide a chance to increase the depth of the CNN (denoted both as the "depth-increasing operator"), while the third mutation operator provides a chance to decrease the depth (denoted it as the "depth-decreasing operator"). For example, if the optimal depth of the CNN is N, after the random initialization, some individuals may have the length (denoting the depth of the CNNs) smaller than N, while others are with length greater than N. During evolution, each individual has a chance to be mutated. If the individual whose length is greater than N and the depth-decreasing operator is selected, the individuals' length will be decreased toward N, and vice versa. In summary, the optimal depth is obtained by performing the mutation operations on the individuals randomly initialized with different lengths.

E. Environmental Selection

Algorithm 6 shows the details of the environmental selection. First, $|P_t|$ 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). Second, the best individual is selected and 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 easily causes the premature phenomenon [52], which will lead the algorithm trap into a local optimum [23], [53]. If we do not explicitly select the best individuals for the next generation, the algorithm will not convergence. In principle, a desirable population should contain not only good but also the relatively bad ones for enhancing the diversity [54], [55]. To this end, the binary tournament selection is typically used for such a purpose [51], [56]. However, only using the binary tournament selection may miss the best individual, resulting in the algorithm, not toward a better direction of the evolution. Hence, we explicitly add the best individual to the next population, which is a particular elitism strategy in evolutionary algorithms [57].

Note that the binary tournament selection, used in the proposed algorithm (they are used in the offspring generating and environmental selection phases), is used with replacement. Based on our previous work [58], the tournament selection with or without replacement almost has no different bias to the final performance, and the replacement mechanism used in the proposed algorithm is just to follow the common practice [27], [28], [30].

IV. Experimental Design

In order to evaluate the performance of the proposed algorithm, a series of experiments have been conducted on image classification tasks. Specifically, the peer competitors chosen to compare with the proposed algorithm are introduced in Section IV-A. Then, the used benchmark datasets are detailed in Section IV-B. Finally, the parameter settings of the proposed algorithm are shown in Section IV-C.

A. 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 [8], DenseNet [9], VGGNet [6], Maxout [59], Network in Network [60], Highway Network [61], and All-CNN [62]. Specifically, two versions of ResNet, that is, the ResNet models with a depth of 110 and 1202, are used for the comparison. For the convenience, they are called ResNet (depth = 110) and ResNet (depth = 1202), respectively. Among all the ResNet versions investigated in the seminal paper [8], ResNet (depth = 110) is the winner in terms of the classification accuracy on CIFAR10, and ResNet (depth = 1202) is the most complex version, which is the main reason for selecting them for the comparison. For the DenseNet, we use the version called "DenseNet-BC" that achieved the promising classification accuracy but also has the least number of parameters among its all variants [9]. Note that most algorithms from this category are the champions in the large-scale visual-recognition challenges in recent years [63].

The second and the third contain the CNN architecture design algorithms from the automatic + manually tuning and the automatic categories, respectively. Specifically, Genetic CNN [10], Hierarchical Evolution [11], EAS [17], BlockQNN-S [13], DARTS [64], and NSANet [14] belong to the second category, while Large-scale Evolution [15], CGP-CNN [16], NAS [17], and MetaQNN [18] fall into the third category. In the comparison, we choose two versions of NASNet, that is, NASNet-B and NASNet-A+cutout, which are the ones having the least number of parameters and the best classification accuracy, respectively. Specifically, the "cutout" refers to a regularization method [65] used in the training of CNNs, which could improve the final performance.

Please note that the proposed algorithm mainly focuses on providing an automatic way to design promising CNN architectures for users who have no rich domain knowledge of tuning CNN architectures. Based on the "No Free Lunch Theorems [66]," we should realize that CNNs whose architectures are designed with manual tuning should have better classification accuracy than the automatic ones including the proposed algorithm. Indeed, only the comparison to the CNN architecture designs from the automatic category is fair to the proposed algorithm. In this experiment, we still would like to provide the extensive comparisons to the CNN architectures designed the with domain knowledge, to show the efficiency and effectiveness of the proposed algorithm among all existing state-of-the-art CNNs.

B. Benchmark Datasets

The CIFAR10 and CIFAR100 benchmark datasets [19] 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 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 benchmark for recognizing ten classes of natural objects, such as airplanes 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 set and the testing set, respectively. Each category has an equal number of images. On the other hand, the CIFAR100 dataset is similar to CIFAR10, except it has 100 classes. In order to have a quick glance on both datasets, we randomly select three classes from each benchmark dataset, and then randomly select ten images from each class. These selected images are shown in Fig. 4. Specifically, Fig. 4(a) shows the images from CIFAR10, while Fig. 4(b) displays those from CIFAR100. The left column in Fig. 4 shows the class names of the images in the same rows. It can be observed that the objects to be classified

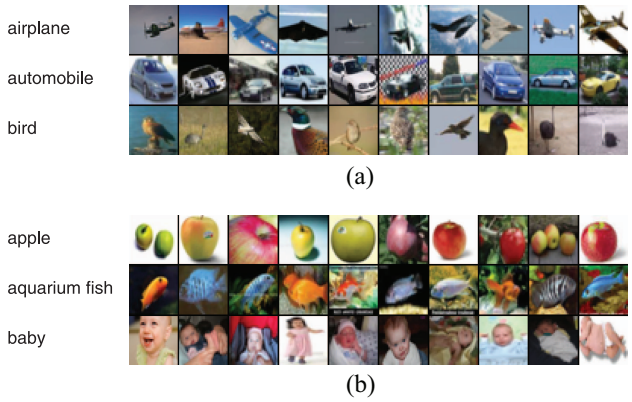

Fig. 4. Examples from CIFAR10 [shown in Fig. 4(a)] and CIFAR100 [shown in Fig. 4(b)]. Each row denotes the images from the same category, and the first column displays the corresponding category label.

in these benchmark datasets typically occupy different areas of the entire 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 set is split into two parts. The first part accounts for 90% to serve as the training set for training the individuals, while the remaining images serve as the fitness evaluation set for evaluating the fitness. 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, that is, 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 [8], [9].

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.

C. Parameter Settings

As have been discussed, the main objective of this article is to design an automatic architecture discovering algorithm for researchers without domain expertise in CNNs. To further improve the 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, 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 [20]. During the training of each individual, the routine in [8] is employed, that is, 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 1st, 149th, and 249th 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 set. Finally, the classification accuracy on the testing set 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 Section III-D, 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. Ideally, a larger population size and a larger maximal generation number should result in a better performance, but expectedly also consume more computational resources. However, such optimal settings are not within the scope of this article since the current settings, as commonly adopted by other papers, have easily outperformed most of the peer competitors based on the results shown in Table I. Noting that the experiments of the proposed algorithm are performed on three GPU cards with the same model of Nvidia GeForce GTX 1080 Ti.

V. Experimental Results and Analysis

In this section, we first show an overview of the comparison results between the proposed algorithm and the chosen peer competitors. Then, the evolutionary trajectories of the proposed algorithm are shown, which helps the readers to better understand the proposed algorithm during the process of discovering the best CNN architecture and the appropriateness of setting to the generation number.

A. Overall Results

Because the state-of-the-art CNNs in the first category of peer competitors are manually designed, we mainly compare the classification accuracy and the number of parameters. For the algorithms in the 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 called CNN-GA which is an architecture discovering algorithm, so it 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 I. In Table I, the peer competitors are grouped into three different blocks based on the categories, 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. In addition, the second column denotes the names of the peer competitors. The third and fourth columns

TABLE I
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?
manually designed	ResNet (depth=110)	93.57	74.84	1.7M	–	completely needed
	ResNet (depth=1,202)	92.07	72.18	10.2M	–	completely needed
	DenseNet-BC	95.49	77.72	0.8M	–	completely needed
	VGGNet	93.34	71.95	20.04M	–	completely needed
	Maxout	90.70	61.40	–	–	completely needed
	Network in Network	91.19	64.32	–	–	completely needed
	Highway Network	92.40	67.66	–	–	completely needed
	All-CNN	92.75	66.29	1.3M	–	completely needed
automatic + manually tuning	Genetic CNN	92.90	70.97	–	17	partially needed
	Hierarchical Evolution	96.37	–	–	300	partially needed
	EAS	95.77	–	23.4M	10	partially needed
	Block-QNN-S	95.62	79.35	6.1M	90	partially needed
	DARTS	97.00	–	3.3M	4	partially needed
	NASNet-B	96.27	–	2.6M	2,000	partially needed
	NASNet-A + cutout	97.60	–	27.6M	2,000	partially needed
automatic	Large-scale Evolution	94.60	–	5.4M	2,750	completely not needed
	Large-scale Evolution	–	77.00	40.4M	2,750	completely not needed
	CGP-CNN	94.02	–	1.68M	27	completely not needed
	NAS	93.99	–	2.5 M	22,400	completely not needed
	Meta-QNN	93.08	72.86	–	100	completely not needed
	CNN-GA	95.22	–	2.9M	35	completely not needed
	CNN-GA	–	77.97	4.1M	40	completely not needed
	CNN-GA + cutout	96.78	–	2.9M	35	completely not needed
	CNN-GA + cutout	–	79.47	4.1M	40	completely not needed

refer to the classification accuracies 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 semiautomatic and automatic algorithms. The symbol "–" implies there is no result publicly reported by the corresponding algorithm. For ResNet (depth = 110), ResNet (depth = 1202), DenseNet, VGG, All-CNN, EAS, and Block-QNN-S, they have the same number of parameters on both the CIFAR10 and CIFAR100 datasets, 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 I 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 times. The best classification accuracy is selected from the five results to show in Table I, which follows the conventions of its peer competitors [10], [11], [13], [15]–[18]. Inspired by the performance improvement of NASNet given by the cutout regularization, we also report the results by performing CNN-GA with the cutout, which is denoted as "CNN-GA+cutout" in Table I. Specifically, we first choose the best CNN-GA architectures found in CIFAR10 and CIFAR100, respectively, then retrain these architectures with the cutout on the training set, and then report the classification accuracy on the testing set.

In this experiment, we try our best to do a fair comparison by using the same training process as well as the same data augmentation method as those of the chosen peer competitors. However, because of multiple peer competitors aiming at finding the best classification accuracy, different training process and data augmentation methods are adopted by them. For example, NASNet employed a revised path-drop technique to improve the classification accuracy. Because the revised path-drop technique is not publicly available, we only keep the proposed algorithm with the same training process and data augmentation methods as those of the most peer competitors. Therefore, we should keep in mind that solely comparing the classification accuracy is not fair to the proposed algorithm either.

For the peer competitors in the first category, CNN-GA obtains 3.15% and 1.88% improvements in terms of the classification accuracy on the CIFAR10 dataset over ResNet (depth = 1202) and VGG, respectively, while using merely 28% and 14% of their respective parameters. The number of parameters in CNN-GA is more than that of ResNet (depth = 110) and All-CNN on the CIFAR10 dataset, but CNN-GA shows the best classification accuracy among them. For DenseNet-BC, which is the most recent state-of-the-art CNN, CNN-GA shows slightly worse classification accuracy on CIFAR10, but on the CIFAR100 that is a more complex benchmark dataset than CIFAR10, CNN-GA shows slightly better classification accuracy. 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 = 110), 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. In addition, 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

[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 I are derived from [16].

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 compared with 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 GPU 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). In addition, NASNet-B shows slightly better classification accuracy on CIFAR10 than CNN-GA (96.20% versus 95.22%), while CNN-GA only consumes about 1/55 GPU days of that consumed by NASNet-B (35 GPU days versus 2000 GPU days). Compared to NASNet-B, NASNet-A+cutout improves a classification accuracy of 1.33% with a CNN having 27.6M parameters, by consuming 2000 GPU days; but the CNN-GA+cutout obtains a 1.76% classification accuracy improvement, while still with the same CNN having 2.9M parameters and consumes 35 GPU days. Although CNN-GA targets at designing promising CNN architectures without any CNN domain knowledge, it still shows comparable classification accuracy among the peer competitors in this automatic + manually tuning category. Compared to CNN-GA, the major limitation of the algorithms in this category is the requirement of 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 a 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. In addition to the CNN framework that is required by all versions of NASNet, NASNet also employed a revised path-dropping technique to improve its performance. In addition, as can be seen in Table I, DARTS achieves a similar performance as well as the number of parameters as CNN-GA on CIFAR10, while consumes much fewer GPU days than CNN-GA dose. However, DARTS requires a manually designed large CNN as a building block, and then using the differential method, that is, the SGD, to choose the best path among the large CNNs. If the provided CNN is smaller than the optimal one, DARTS can never find the optimal CNN architecture. The proposed CNN-GA bears no such limitations, although it consumes more GPU days than DARTS dose.

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 in terms of the classification accuracy. Meanwhile, CNN-GA also shows superiority in 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 [15]. 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 2750 GPU days on the CIFAR10 dataset and another 2750 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 most of the state-of-the-art CNNs manually designed and all the 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 automatic + manually tuning architecture discovering algorithms show similar (to slightly better) classification accuracy to that of CNN-GA, CNN-GA is automatic and does not require users to have any expertise in CNNs when solving real-world tasks, which is the main purpose of our work in this article.

To make the readers intuitively understand the efficacy of the proposed one on the image classification tasks, the representative of the conventional image classification methods [67], that is, the histograms of the oriented gradient (HOG) descriptor with the support vector machine (SVM) (in short, denoted as HOG + SVM), are used to compare the classification accuracy on CIFAR10 and CIFAR100. Particularly, we use the source code of HOG + SVM downloaded from the Github,[4] and the resulted classification accuracies of CIFAR10 and CIFAR100 are 51.22% and 43.90%, respectively. As compared with the results of the proposed algorithm, which are 96.78% and 79.47%, respectively, it is easy to conclude that the proposed algorithm can outperform the HOG + SVM with the significant improvement.

B. Evolutionary Trajectories

In order to better understand the details of the proposed algorithm in discovering the architectures of CNNs and the appropriateness of setting 20 as the generation number, the evolutionary trajectory of the proposed algorithm on the CIFAR10 dataset is shown in Fig. 5. To achieve this, we first collect the individuals selected by the environmental selection in each generation, and then use the boxplot [68] to show the statistics in terms of the classification accuracy. Meanwhile, we also connect the best and median classification accuracy during each generation by using a dashed line and a solid line, respectively. In Fig. 5, the horizon axis denotes the number of

[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 4671. In the same way, the average GPU days took by CNN-GA are found to be 37.5.

[4]https://github.com/subicWang/HOG-SVM-classifer

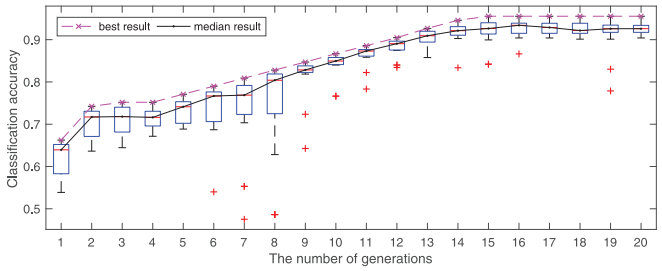

Fig. 5. Evolutionary trajectory of the proposed algorithm in discovering the best architecture of CNN on the CIFAR10 dataset.

generations, while the vertical axis denotes the classification accuracy.

As shown in Fig. 5, both the best and median classification accuracy increase as the evolution progresses. By investigating the height of each box, it can also be observed that the variation of the classification accuracy during each generation becomes smaller and smaller, which implies the evolution toward a steady state in discovering the architectures of CNNs on the CIFAR10 dataset. In addition, the classification accuracy increases sharply from the first generation to the second generation, which is due to the random initialization of the population at the beginning of the evolution. From the second to the fourth generations, the improvement of the classification accuracy becomes much smaller than the previous generations, and it sharply increases immediately after until the 15th generation. Since then, the classification accuracy does not change too much until the evolution terminates, which also implies that the setting of 20 generations in this particular case is reasonable because the proposed algorithm well converges with this setting.

VI. Conclusion

The objective of this article is to propose an automatic architecture design algorithm for CNNs by using the GA (in short called CNN-GA), which is capable of discovering the best CNN architecture in addressing image classification problems for the users who have no expertise in tuning CNN architectures. 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 was examined on two challenging benchmark datasets, and compared with 18 state-of-the-art peer competitors, including eight manually designed CNNs, six automatic + manually tuning, and four automatic algorithms discovering the architectures of CNNs. The experimental results show that CNN-GA outperforms almost all of the manually designed CNNs as well as the automatic peer competitors, and shows competitive performance with respect to automatic + manually tuning 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 resources than most automatic and automatic + manually tuning peer competitors. CNN-GA is also completely automatic, and users can directly use it to address their own image classification problems whether or not they have domain expertise in CNNs or GAs. Moreover, the CNN architecture designed by CNN-GA on CIFAR10 also shows the promising performance when it is transferred to the ImageNet dataset.

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 compared to that used in GAs when solving traditional problems. In the field of solving expensive optimization problems, several algorithms based on evolutionary computation techniques have been developed. In the 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 *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[3] T. N. Sainath, A.-R. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for LVCSR," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2013, pp. 8614–8618.
[4] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3104–3112.
[5] C. Clark and A. J. Storkey, "Training deep convolutional neural networks to play go," in *Proc. 32nd Int. Conf. Mach. Learn.*, Lille, France, 2015, pp. 1766–1774.
[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, San Diego, CA, USA, May 2015. [Online]. Available: http://arxiv.org/abs/1409.1556
[7] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Boston, MA, USA, 2015, pp. 1–9.
[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, 2016, pp. 770–778.
[9] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Honolulu, HI, USA, 2017, pp. 2261–2269.
[10] L. Xie and A. L. Yuille, "Genetic CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, Venice, Italy, 2017, pp. 1388–1397.
[11] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *Proc. Mach. Learn. Res.*, Stockholm, Sweden, 2018, pp. 1–13.
[12] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 2787–2794.
[13] Z. Zhong, J. Yan, and C.-L. Liu, "Practical network blocks design with *Q*-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 1–10.
[14] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8697–8710.
[15] E. Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. Mach. Learn. Res.*, 2017, pp. 2902–2911.
[16] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proc. Genet. Evol. Comput. Conf.*, 2017, pp. 497–504.
[17] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, Toulon, France, 2017, pp. 1–16.
[18] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, Toulon, France, 2017, pp. 1–18.

[19] A. Krizhevsky and G. Hinton. (2009). *Learning Multiple Layers of Features From Tiny Images*. [Online]. Available: http://www.cs.toronto.edu/kriz/cifar.html
[20] T. Back, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. New York, USA: Oxford Univ. Press, 1996.
[21] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, vol. 1. Cambridge, MA, USA: MIT Press, 1998.
[22] Y. Sun, B. Xue, and M. Zhang, "Evolving deep convolutional neural networks for image classification," 2017. [Online]. Available: arXiv:1710.10741.
[23] L. Davis, *Handbook of Genetic Algorithms*. New York, USA: Taylor & Francis, 1991.
[24] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming: An Introduction*, vol. 1. San Francisco, CA, USA: Morgan Kaufmann, 1998.
[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," *Theor. Comput. Sci.*, vol. 259, nos. 1–2, pp. 1–61, 2001.
[27] Y. Sun, G. G. Yen, and Z. Yi, "IGD indicator-based evolutionary algorithm for many-objective optimization problems," *IEEE Trans. Evol. Comput.*, vol. 23, no. 2, pp. 173–187, Apr. 2019.
[28] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[29] Y. Sun, G. G. Yen, and Z. Yi, "Reference line-based estimation of distribution algorithm for many-objective optimization," *Knowl. Based Syst.*, vol. 132, pp. 129–143, Sep. 2017.
[30] M. Jiang, Z. Huang, L. Qiu, W. Huang, and G. G. Yen, "Transfer learning based dynamic multiobjective optimization algorithms," *IEEE Trans. Evol. Comput.*, vol. 22, no. 4, pp. 501–514, Aug. 2018.
[31] Y. Sun, G. G. Yen, and Z. Yi, "Improved regularity model-based EDA for many-objective optimization," *IEEE Trans. Evol. Comput.*, vol. 22, no. 5, pp. 662–678, Oct. 2018.
[32] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.
[33] H. L. Liu, F. Gu, Y.-M. Cheung, S. Xie, and J. Zhang, "On solving WCDMA network planning using iterative power control scheme and evolutionary multiobjective algorithm [application notes]," *IEEE Comput. Intell. Mag.*, vol. 9, no. 1, pp. 44–52, Feb. 2014.
[34] K. Nag and N. R. Pal, "A multiobjective genetic programming-based ensemble for simultaneous feature selection and classification," *IEEE Trans. Cybern.*, vol. 46, no. 2, pp. 499–510, Feb. 2016.
[35] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[36] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Comput.*, vol. 12, no. 10, pp. 2451–2471, 2000.
[37] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2377–2385.
[38] A. E. Orhan and X. Pitkow, "Skip connections eliminate singularities," in *Proc. Mach. Learn. Res.*, Stockholm, Sweden, 2018, pp. 1–22.
[39] M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *Computer*, vol. 27, no. 6, pp. 17–26, 1994.
[40] D. M. Hawkins, "The problem of overfitting," *J. Chem. Inf. Comput. Sci.*, 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," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
[42] K. He, X. Zhang, S. Ren, and J. Sun, *Identity Mappings in Deep Residual Networks* (Lecture Notes in Computer Science) Amsterdam, the Netherlands: Springer, 2016, pp. 630–645.
[43] R. Housley, "A 224-bit one-way hash function: SHA-224," IETF, RFC 3874, Sep. 2004.
[44] N. M. Nasrabadi, "Pattern recognition and machine learning," *J. Electron. Imag.*, vol. 16, no. 4, 2007, Art. no. 049901.
[45] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. 14th Int. Conf. Artif. Intell. Stat.*, 2011, pp. 315–323.
[46] S. Ioffe, "Batch renormalization: Towards reducing minibatch dependence in match-normalized models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 1945–1953.
[47] L. Bottou, "Stochastic gradient descent tricks," in *Neural Networks: Tricks of the Trade*. Heidelberg, Germany: Springer, 2012, pp. 421–436.
[48] R. Helfenstein and J. Koko, "Parallel preconditioned conjugate gradient algorithm on GPU," *J. Comput. Appl. Math.*, vol. 236, no. 15, pp. 3584–3590, 2012.
[49] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. Oper. Syst. Design Implement.*, vol. 16, 2016, pp. 265–283.
[50] A. Paszke *et al.* (2017). *Automatic Differentiation in Pytorch*. [Online]. Available: https://openreview.net/forum?id=BJJsrmfCZ
[51] B. L. Miller and D. E. Goldberg, "Genetic algorithms, tournament selection, and the effects of noise," *Complex Syst.*, vol. 9, no. 3, pp. 193–212, 1995.
[52] Z. Michalewicz and S. J. Hartley, "Genetic algorithms + data structures = evolution programs," *Math. Intell.*, vol. 18, no. 3, p. 71, 1996.
[53] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Mach. Learn.*, vol. 3, no. 2, pp. 95–99, 1988.
[54] C. M. Anderson-Cook, *Practical Genetic Algorithms*. Hoboken, NJ, USA: Wiley-Intersci., 2005, p. 1099.
[55] S. Malik and S. Wadhwa, "Preventing premature convergence in genetic algorithm using DGCA and elitist technique," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 4, no. 6, pp. 117–128, 2014.
[56] G. Zhang, Y. Gu, L. Hu, and W. Jin, "A novel genetic algorithm and its application to digital filter design," in *Proc. IEEE Intell. Transp. Syst.*, vol. 2, 2003, pp. 1600–1605.
[57] D. Bhandari, C. A. Murthy, and S. K. Pal, "Genetic algorithm with elitist model and its convergence," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 10, no. 6, pp. 731–747, 1996.
[58] H. Xie and M. Zhang, "Impacts of sampling strategies in tournament selection for genetic programming," *Soft Comput.*, vol. 16, no. 4, pp. 615–633, 2012.
[59] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *Proc. 30th Int. Conf. Mach. Learn.*, Atlanta, GA, USA, Jun. 2013, pp. 1319–1327.
[60] M. Lin, Q. Chen, and S. Yan, "Network in network," in *Proc. Int. Conf. Learn. Represent.*, Banff, AB, Canada, Apr. 2014. [Online]. Available: http://arxiv.org/abs/1312.4400
[61] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," in *Proc. Int. Conf. Mach. Learn. Workshop*, Lille, France, Jul. 2015. [Online]. Available: http://arxiv.org/abs/1505.00387
[62] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," in *Proc. Int. Conf. Learn. Represent. Workshop*, San Diego, CA, USA, May 2015. [Online]. Available: http://arxiv.org/abs/1412.6806
[63] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.
[64] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," 2018. [Online]. Available: arXiv:1806.09055.
[65] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," 2017. [Online]. Available: arXiv:1708.04552.
[66] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.
[67] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2005, pp. 886–893.
[68] D. F. Williamson, R. A. Parker, and J. S. Kendrick, "The box plot: A simple visual method to interpret data," *Ann. Internal Med.*, vol. 110, no. 11, pp. 916–921, 1989.

Yanan Sun (Member, IEEE) received the Ph.D. degree in engineering from Sichuan University, Chengdu, China, in 2017.

He is currently a Professor (research) with the College of Computer Science, Sichuan University. Prior to that, he was a Research Fellow with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. His research topics are evolutionary algorithms, deep learning, and evolutionary deep learning.

Prof. Sun is the Leading Organizer of the First Workshop on Evolutionary Deep Learning, and Special Session on Evolutionary Deep Learning and Applications in CEC19, and the Founding Chair of the IEEE CIS Task Force on Evolutionary Deep Learning and Applications.

Bing Xue (Member, IEEE) received the B.Sc. degree from the Henan University of Economics and Law, Zhengzhou, China, in 2007, the M.Sc. degree in management from Shenzhen University, Shenzhen, China, in 2010, and the Ph.D. degree in computer science from the Victoria University of Wellington, Wellington, New Zealand, in 2014.

She is currently an Associate Professor with the School of Engineering and Computer Science, Victoria University of Wellington. She has over 100 papers published in fully refereed international journals and conferences and most of them are on evolutionary feature selection and construction. Her research focuses mainly on evolutionary computation, feature selection, feature construction, multiobjective optimization, image analysis, transfer learning, data mining, and machine learning.

Dr. Xue is currently the Chair of the IEEE Task Force on Evolutionary Feature Selection and Construction and IEEE Computational Intelligence Society (CIS), and the Vice-Chair of the IEEE CIS Data Mining and Big Data Analytics Technical Committee and IEEE CIS Task Force on Transfer Learning and Transfer Optimization. She is also an Associate Editor/member of the Editorial Board for five international journals and a Reviewer of over 50 international journals. She is the Finance Chair of the IEEE Congress on Evolutionary Computation 2019.

Mengjie Zhang (Fellow, IEEE) received the B.E. and M.E. degrees from the Artificial Intelligence Research Center, Agricultural University of Hebei, Hebei, China, in 1989 and 1992, respectively, and the Ph.D. degree in computer science from RMIT University, Melbourne, VIC, Australia, in 2000.

He is currently a Professor of computer science, the Head of the Evolutionary Computation Research Group, and the Associate Dean (Research and Innovation) with the Faculty of Engineering, Victoria University of Wellington, Wellington, New Zealand. He has published over 350 research papers in refereed international journals and conferences. His current research interests include evolutionary computation, particularly genetic programming, particle swarm optimization, and learning classifier systems with application areas of image analysis, multiobjective optimization, feature selection and reduction, job-shop scheduling, and transfer learning.

Prof. Zhang is currently chairing the IEEE CIS Intelligent Systems and Applications Technical Committee, and the immediate Past Chair for the IEEE CIS Emergent Technologies Technical Committee and the Evolutionary Computation Technical Committee, and a member of the IEEE CIS Award Committee. He is a Vice Chair of the IEEE CIS Task Force on Evolutionary Feature Selection and Construction, and the Task Force on Evolutionary Computer Vision and Image Processing, and the Founding Chair of the IEEE Computational Intelligence Chapter in New Zealand. He is also a Committee Member of the IEEE NZ Central Section. He is a Fellow of the Royal Society of New Zealand and has been a Panel Member of the Marsden Fund (New Zealand Government Funding). He is a member of ACM.

Gary G. Yen (Fellow, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Notre Dame, Notre Dame, IN, USA, in 1992.

In 1997, he was with the Structure Control Division, Air Force Research Laboratory, Albuquerque, NM, USA. He is currently a Regents Professor with the School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK, USA. His research interests includes intelligent control, computational intelligence, conditional health monitoring, and signal processing and their industrial/defense applications.

Prof. Yen received the Andrew P Sage Best Transactions Paper Award from IEEE Systems, Man and Cybernetics Society in 2011 and the Meritorious Service Award from IEEE Computational Intelligence Society in 2014. He served as a Vice President for the Technical Activities in 2005–2006 and then as a President in 2010–2011 of the IEEE Computational intelligence Society. He was the Founding Editor-in-Chief of the *IEEE Computational Intelligence Magazine*; in 2006–2009. He was an Associate Editor of *IEEE Control Systems Magazine*; the IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY; *Automatica*; *Mechatronics*; IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART A: SYSTEMS AND HUMANS; the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART C: APPLICATIONS AND REVIEWS; and the IEEE TRANSACTIONS ON NEURAL NETWORKS. He is currently serving as an Associate Editor for the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and the IEEE TRANSACTIONS ON CYBERNETICS. He served as the General Chair for the 2003 IEEE International Symposium on Intelligent Control held in Houston, TX, USA, and the 2006 IEEE World Congress on Computational Intelligence held in Vancouver, BC, Canada.

Jiancheng Lv (Member, IEEE) received the Ph.D. degree in computer science and engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 2006.

He was a Research Fellow with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore. He is currently a Professor with the Data Intelligence and Computing Art Laboratory, College of Computer Science, Sichuan University, Chengdu. His research interests include neural networks, machine learning, and big data.