Completely Automated CNN Architecture Design Based on Blocks

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

***Abstract*— The performance of convolutional neural networks (CNNs) highly relies on their architectures. In order to design a CNN with promising performance, extensive expertise in both CNNs and the investigated problem domain is required, which is not necessarily available to every interested user. To address this problem, we propose to automatically evolve CNN architectures by using a genetic algorithm (GA) based on ResNet and DenseNet blocks. The proposed algorithm is completely automatic in designing CNN architectures. In particular, neither preprocessing before it starts nor postprocessing in terms of CNNs is needed. Furthermore, the proposed algorithm does not require users with domain knowledge on CNNs, the investigated problem, or even GAs. The proposed algorithm is evaluated on the CIFAR10 and CIFAR100 benchmark data sets against 18 state-of-the-art peer competitors. Experimental results show that the proposed algorithm outperforms the state-of-the-art CNNs hand-crafted and the CNNs designed by automatic peer competitors in terms of the classification performance and achieves a competitive classification accuracy against semiautomatic peer competitors. In addition, the proposed algorithm consumes much less computational resource than most peer competitors in finding the best CNN architectures.**

***Index Terms*— Automatic architecture design, convolutional neural networks (CNNs), evolutionary deep learning, genetic algorithms (GAs), neural networks.**

I. Introduction

CONVOLUTIONAL neural networks (CNNs) [1] have been showcasing their promising performance on various real-world applications [2]–[5]. It has been known that the performance of CNNs highly depends on their architectures, such as how many building-block layers (e.g., the convolutional and pooling layers) are used, how the used building-block layers are composed, and how the parameters related to the used building-block layers are specified.

Generally, given a CNN, denoted by $\mathcal{A}$, having n architecture related parameters $\lambda_1, \ldots, \lambda_n$ whose decision spaces are $\Lambda_1, \ldots, \Lambda_n$, respectively, the CNN architecture design is to optimize the problem formulated as follows:

$$\begin{cases} \arg\min_{\boldsymbol{\lambda}} \quad \mathcal{L}(\mathcal{A}_{\boldsymbol{\lambda}}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}}) \\ \text{s.t. } \boldsymbol{\lambda} \in \boldsymbol{\Lambda} \end{cases} \tag{1}$$

where $\boldsymbol{\lambda} = \{\lambda_1, \ldots, \lambda_n\}$, $\boldsymbol{\Lambda} = \Lambda_1 \times \cdots \times \Lambda_n$, $\mathcal{A}_{\boldsymbol{\lambda}}$ denotes the CNN $\mathcal{A}$ adopting the architecture parameter setting $\boldsymbol{\lambda}$, and $\mathcal{L}(\cdot)$ measures the performance of $\mathcal{A}_{\boldsymbol{\lambda}}$ on the validation data $\mathcal{D}_{\text{valid}}$ after $\mathcal{A}_{\boldsymbol{\lambda}}$ has been trained on the training data $\mathcal{D}_{\text{train}}$. In the case of classification tasks, $\mathcal{L}(\cdot)$ measures the classification error of the tasks to which $\mathcal{A}$ is applied. Typically, the gradient-based algorithms, such as stochastic gradient descent (SGD) [6], are employed to train the weights of $\mathcal{A}_{\boldsymbol{\lambda}}$, as $\mathcal{L}(\cdot)$ is differentiable (or approximately differentiable) with respect to the weights.

Unfortunately, with respect to the architecture related parameters, $\mathcal{L}(\cdot)$ is often nonconvex and nondifferentiable because these parameters usually have discrete values, e.g., the feature map sizes of convolutional layers are generally specified as integers. To this end, the exact optimization algorithms (e.g., the gradient-based algorithms) are incapable of or ineffective in solving the architecture optimization problem [7], [8]. As a result, researchers have proposed various architecture optimization algorithms based on the heuristic computational paradigms [9], such as random search [10], Bayesian-based Gaussian process [11], [12], tree-structured Parzen estimators [13], sequential model-based global optimization [14], neuroevolution of augmenting topologies [15], and evolutionary unsupervised deep learning [8]. However, in CNN architecture optimization, it is impossible to know the optimal numbers of built layers in advance, e.g., the particular value of n in $\boldsymbol{\lambda}$, to compose the best CNN architecture, i.e., the number of decision variables for an optimal CNN architecture is also unknown before the best CNN architecture is found. This makes the aforementioned architecture optimization methods also unable to be effectively and efficiently used for CNN architecture design because they work under the assumption where the number of optimized parameters is fixed. Although we could enumerate each potential value of n and, then, perform these methods for each of the different n,

Manuscript received December 5, 2018; revised May 6, 2019; accepted May 24, 2019. Date of publication June 20, 2019; date of current version April 3, 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, in part by the Marsden Fund of New Zealand Government under Contract VUW1209, Contract VUW1509, and Contract VUW1615, in part by the Huawei Industry Fund under Grant E2880/3663, and in part by the University Research Fund at the Victoria University of Wellington under Grant 209862/3580 and Grant 213150/3662. *(Corresponding author: Gary G. Yen.)*

Y. Sun is with the College of Computer Science, Sichuan University, Chengdu 610065, China, and also with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand (e-mail: ysun@scu.edu.cn).

B. Xue and M. 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).

G. G. Yen is with the School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK 74078 USA (e-mail: gyen@okstate.edu).

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/TNNLS.2019.2919608

the run-time computational complexity will increase in an order of magnitude as n grows, and the satisfactory solutions may not be obtained within the acceptable time [16].

Due to this, the state-of-the-art CNNs, such as ResNet [17] and DenseNet [18], are primarily hand-crafted. Designing CNNs manually requires considerable expertise in the CNN architecture, as well as in the problem domain. This is often not available in practice. For example, a medical doctor could find a CNN extremely useful in evaluating the results of a magnetic resonance imaging (MRI) scan. While the doctor clearly has expertise in the problem domain, they are very unlikely to have comparable experience in the CNN architectures. This barrier has prevented CNNs from being utilized in a variety of image classification tasks. There is a significant demand for algorithms that are able to effectively and efficiently design[1] CNN architectures without requiring such expertise.

Fortunately, in the last two years, multiple algorithms developed for designing CNN architectures have been proposed. Based on whether the preprocessing or postprocessing in terms of CNNs is required when these algorithms are used, they can be divided into two different categories: the semiautomatic CNN architecture design algorithms and the completely automatic ones. Particularly, the semiautomatic algorithms cover the genetic CNN method (Genetic CNN) [19], the hierarchical representation-based method (Hierarchical Evolution) [20], the efficient architecture search method (EAS) [21], and the block design method (Block-QNN-S) [22], to name a few. The automatic algorithms include the large-scale evolution method (Large-scale Evolution) [23], the Cartesian genetic programming method (CGP-CNN) [24], the neural architecture search method (NAS) [25], and the metamodeling method (MetaQNN) [26]. These algorithms are mainly based on evolutionary algorithms [27] or reinforcement learning [28]. For example, Genetic CNN, Large-scale Evolution, Hierarchical Evolution, and CGP-CNN are based on evolutionary algorithms, while NSA, MetaQNN, EAS, and Block-QNN-S are built on reinforcement learning.

Experimental results from these algorithms have shown their promising performance in finding the best CNN architectures on the given data. However, major limitations remain. First, the expertise in the investigated data and CNNs is still needed by the semiautomatic CNN architecture design algorithms. For example, EAS takes effect on a base network, which already has a fairly good performance on the investigated problem. However, the base network is manually designed based on expertise. Block-QNN-S only designs several small networks, and these networks are then integrated into a larger CNN framework. However, the other types of layers, such as the pooling layers, need to be properly assimilated into the CNN framework with expertise. Second, the CNN architecture design algorithms based on reinforcement learning typically consume much more computational resource. For instance, NAS consumes 28 days on 800 graphics process unit (GPU) cards for the CIFAR10 data set [29]. However, sufficient computation resource is not necessarily available to every interested user. Finally, the CNN architecture design algorithms based on the evolutionary algorithm use only partial principled merit of the evolutionary algorithms, which inadvertently results in the found CNNs usually without the promising performance for the investigated problems. For example, Genetic CNN employs a fixed-length encoding scheme to represent CNNs. However, we never know the best depth of the CNN in solving a new problem. To this end, Large-scale Evolution utilizes a variable-length encoding scheme where the CNNs can adaptively change their depths for the problems. However, Large-scale Evolution uses only the mutation operator but not any crossover operator during the search process. In evolutionary algorithms, the crossover operator and the mutation operator play complementary roles of local search and global search. Without using the crossover operator, the mutation operator works just like a random search at different start positions. Nevertheless, it is not surprising that Large-scale Evolution does not use the crossover operator since the crossover operator is originally designed for the fixed-length encoding scheme.

To this end, the development of CNN architecture design algorithms, especially for the completely automatic ones with promising performance and relying on the limited computational resource, is still in its infancy. The aim of this paper is to design and develop a new genetic algorithm (GA)-based algorithm to automatically design CNN architectures by addressing the above-discussed limitations. To achieve this goal, the objectives have been specified in the following.

1) The proposed algorithm does mandate any prerequisite knowledge from the users in base CNN design, investigated data set, and GAs. The CNN whose architecture is designed by the proposed algorithm can be directly used without any recomposition, preprocessing, or postprocessing.
2) The variable-length encoding scheme is employed for searching the optimal depth of the CNN. To adopt the variable-length encoding, a new crossover operator and a mutation operator are designed and incorporated into the proposed algorithm to collectively exploit and explore the search space in finding the best CNN architectures.
3) An efficient encoding strategy is designed based on the ResNet block (RB) and DenseNet block (DB) for speeding up the architecture design, and the limited computational resource is utilized, while the promising performance can be achieved by the proposed algorithm. Noting that, although the RB and DB are used in the proposed algorithm, the users are not required to have expertise in these blocks when they are using the proposed algorithm.

The remainder of this paper is organized as follows. The background related to the base knowledge of the proposed algorithm is introduced in Section II. Then, the details of the proposed algorithm are documented in Section III. To evaluate the performance of the proposed algorithm, the experiment design and the numerical results are shown in Sections IV and V, respectively. Finally, the conclusions and future work are summarized in Section VI.

[1]In this paper, the terms "design," "find," "learn," and "evolve" have identical meaning when used to describe the "CNN architectures."

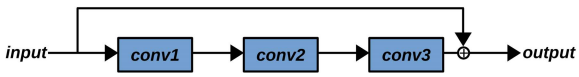

Fig. 1. Example of the RB.

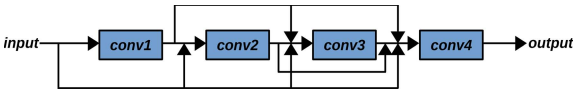

Fig. 2. Example of the DB including four convolutional layers.

II. Background

As have highlighted in Section I, the proposed algorithm is to design a novel GA, to automatically design the CNN architectures, by using the blocks of ResNet and DenseNet that are the state-of-the-art CNNs manually designed. In order to help readers easily understand the details of the proposed algorithm to be shown in Section III, the fundamentals to GAs, RBs, and DBs are discussed in this section.

A. Genetic Algorithms

GAs [30] are a class of heuristic population-based computational paradigm. They are also the most popular type of evolutionary algorithms [evolutionary algorithms broadly include genetic programming (GP) [31], evolutionary strategy [32], and so on, in addition to GAs]. Because of the nature of gradient-free and insensitiveness to the local minimum, GAs are preferred, especially in engineering fields where the optimization problems are commonly nonconvex and nondifferentiable [33], [34]. GAs address optimization problems by imitating the biological evolution through a series of bio-inspired operators, such as crossover, mutation, and selection [35], [36]. Generally, a GA works as follows.

Step 1: Initialization of a population of individuals each of which represents a candidate solution of the problem through the employed encoding strategy.

Step 2: Evaluation of the fitness of each individual in the population based on the encoded information and the fitness function.

Step 3: Mating selection of promising parent individuals from the current population, and then, generate offspring with crossover and mutation operators.

Step 4: Evaluation of the fitness of the generated offspring.

Step 5: Environmental selection of a population of individuals with a promising performance from the current population, and then, replace the current population by the selected population.

Step 6: Go to Step 3 if the termination criterion is not met; otherwise, return the individual with the best fitness as the best solution for the problem.

Commonly, a maximal generation number is predefined as the termination criterion.

B. ResNet and DenseNet Blocks

ResNet [17] and DenseNet [18] are two state-of-the-art CNNs proposed in recent years. The success of ResNet and DenseNet largely owes to their building blocks, i.e., RBs and DBs, respectively.

Fig. 1 shows an example of an RB, which is composed of three convolutional layers[2] and one skip connection. In this example, the convolutional layers are denoted as $conv1$, $conv2$, and $conv3$. On $conv1$, the spatial size of the input is reduced by a smaller number of filters with a size of 1×1, to lower the computational complexity of $conv2$. On $conv2$, filters with a larger size, such as 3×3, are used to learn features with the same spatial size. On $conv3$, filters with a size of 1×1 are used again, and the spatial size is increased for generating more features. The input is added, denoted by $\oplus$, to the output of $conv3$ as the final output of the RD. Noting that if the spatial sizes of the input and $conv3$'s output are unequal, a group of convolutional operations with the filters of 1×1 size is applied on the input, to achieve the same spatial size as that of $conv3$'s output, for the addition.

Fig. 2 exhibits an example of a DB. For the convenience of the introduction, we give only four convolutional layers in the DB. In practice, the DB can have a different number of convolutional layers, which is tuned by users. In the DB, each convolutional layer receives inputs from not only the input data but also the output of all the previous convolutional layers. In addition, there is a parameter k for controlling the spatial size of the input and output of the same convolutional layer. If the spatial size of the input is a, then the spatial size of the output is $a + k$, which is achieved by the convolutional operation using the corresponding number of filters.

Efforts in [37] and [38] have been put on investigating the mechanism behind the success of RBs and DBs and revealed that RBs and DBs are able to mitigate the adverse impact of the gradient vanishing problem [39], based on which a deep architecture is capable of effectively learning the hierarchical representations of the input data and then improving the final classification accuracy in turn. In addition, the dense connections in DBs have also been claimed to be able to reuse the low-level features to increase the discrimination of features learned at the top layers of CNNs [18]. Mainly based on these good characteristics, RBs and DBs are chosen as the building blocks in the proposed algorithm.

III. Proposed Algorithm

In this section, the framework of the proposed algorithm and its main components are discussed in detail. For the convenience of the development, the proposed algorithm is named AE-CNN (automatically evolving CNN) in short, and the evolved CNN is used solely for image classification tasks.

A. Algorithm Overview

Algorithm 1 shows the framework of AE-CNN, which is composed of three parts. First, the population is randomly initialized with a predefined size of N (see line 1). Then, the individuals are evaluated for the fitness (see line 2). Next, all individuals in the population take part in the evolutionary process of GA with the maximal generation number of

[2]Here, we only detail this type of blocks, which is used to build deeper networks. Indeed, ResNet also has another type of block, which is typically used for building networks with no more than 34 layers.

Algorithm 1 Framework of AE-CNN

Input: The population size N, the maximal generation number T, the crossover propability μ, the mutation probability ν.
Output: The best CNN.

1 $P_0 \leftarrow$Initialize a population with the size of N by using the proposed encoding strategy;
2 Evaluate the fitness of individuals in P_0;
3 $t \leftarrow 0$;
4 **while** $t < T$ **do**
5 $\quad Q_t \leftarrow \emptyset$;
6 $\quad$ **while** $|Q_t| < N$ **do**
7 $\quad\quad p_1, p_2 \leftarrow$ Select two parent individuals from P_t by using binary tournament selection;
8 $\quad\quad q_1, q_2 \leftarrow$ Generate two offspring by p_1 and p_2 by crossover operation with the probability of μ and mutation operation with the probability of ν;
9 $\quad\quad Q_t \leftarrow Q_t \cup q_1 \cup q_2$;
10 $\quad$ **end**
11 $\quad$ Evaluate the fitness of individuals in Q_t;
12 $\quad P_{t+1} \leftarrow$ Select N individuals from $P_t \cup Q_t$ by environmental selection;
13 $\quad t \leftarrow t + 1$;
14 **end**
15 Select the best individual from P_t and decode it to the corresponding CNN.

T (see lines 3–14). Finally, the best CNN architecture is decoded from the best individual that is chosen from the final population based on the fitness (see line 15). During the evolutionary process, an empty population is initialized for including offspring (see line 5), and then, a new offspring is generated from selected parents with the crossover and mutation operations, while the parents are selected by the binary tournament selection (see lines 6–10); after the fitness of the generated offspring has been evaluated (see line 11), a new population is selected with the environmental selection operation (see line 12) from the current population (containing the current individuals and the generated offspring) as the parent solutions surviving into the next evolutionary process (i.e., the next generation). Noting that the symbol of $|\cdot|$ shown in line 6 is a cardinality operator. The phases of "population initialization," "fitness evaluation," "offspring generation," and "environmental selection" are documented in Sections III-B–III-E, respectively.

B. Population Initialization

Population initialization provides a base population containing multiple individuals for the following evolutionary process. Generally, all the individuals are initialized in a random manner with a uniform distribution, as have introduced in Section II-A that each individual in GAs represents a candidate solution of the problem to be solved. Because GAs in the proposed algorithm are employed to find the best CNN architecture, each individual in the proposed algorithm should represent a CNN architecture. Generally, the architecture of a CNN is constructed by multiple convolutional layers, pooling layers, and fully connected layers in a particular order, as well as their parameter settings. In the proposed algorithm, CNNs are constructed based on RBs, DBs, and pooling layers, which is motivated by the remarkable success of ResNet [17] and DenseNet [18], while the fully connected layers are not considered in the proposed algorithm. The main reason is that the fully connected layers easily cause the overfitting phenomenon [40] due to their full-connection nature. To reduce this phenomenon, other techniques must be adopted, such as dropout [41]. However, these techniques will also give rise to extra parameters that need to be carefully tuned, which will increase the computational complexity of the proposed algorithm. The experimental results shown in Section V will justify that the promising performance of the proposed algorithm can still be achieved without using the fully connected layers. The details of initializing the population of AE-CNN are summarized in Algorithm 2.

Algorithm 2 Initialize Population

Input: The population size N, the training instance dimension $d \times d$.
Output: The initialized population P_0.

1 $P_0 \leftarrow \emptyset$;
2 $m_p \leftarrow$ Calculate the maximal number of pooling layers by $\lfloor log_2(d) \rfloor$;
3 **for** $i \leftarrow 1$ ***to*** N **do**
4 $\quad k \leftarrow$ randomly initialize a positive integer;
5 $\quad a \leftarrow$ initialize an empty array with the size of k;
6 $\quad$ **for** $j \leftarrow 1$ ***to*** k **do**
7 $\quad\quad u \leftarrow$ Randomly choose one from {RBU, DBU, PU};
8 $\quad\quad$ **if** *u is a PU and the number of used PU is not less than m_p* **then**
9 $\quad\quad\quad u \leftarrow$ Randomly choose one from {RBU, DBU};
10 $\quad\quad$ **end**
11 $\quad\quad$ Encode u and put the encoded information into the j-th position of a;
12 $\quad$ **end**
13 $\quad P_0 \leftarrow P_0 \cup a$;
14 **end**
15 **Return** P_0.

Next, we will explain the details of lines 8 and 11 because other parts of Algorithm 2 are straightforward. Specifically, the pooling layers in CNNs perform the dimension reduction on their input data, and the most commonly used pooling operation is to halve the input size, which can be seen from the state-of-the-art CNNs [2]–[5], [17], [18]. To this end, the employed pooling layers cannot be arbitrarily specified but following the constraint that has been calculated, as shown in line 2. For example, if the input size is 32×32, the number of used pooling layers cannot be larger than six because six pooling layers will reduce the dimension of the input data to 1×1, and one extra pooling layer on the dimension of 1×1 will lead to the logic error.

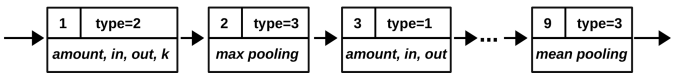

Fig. 3. Example of the proposed encoding strategy.

Encoding enables GAs with the ability to model real-world problems, and then, the problems can be solved by the GAs directly. The encoding is achieved by the corresponding encoding strategy, which is the first step of employing GAs. There is not a unified encoding strategy that can be used for all the problems. In the proposed algorithm, we design a new encoding strategy aiming at effectively modeling CNNs with different architectures. For the used RBs, based on the configuration of the state-of-the-art CNNs [17], [42], we set the filter size of $conv2$ to 3×3, which is also used for the convolutional layers in the used DBs. For the used pooling layers, we set the same stride as the step size to 2×2 based on the conventions, which means that such a single pooling layer in the evolved CNN halves the input dimension for one time. To this end, the unknown parameter settings for RBs are the spatial sizes of input and output, those for DBs are the spatial sizes of input and output, as well as k, and that for pooling layers are only their types, i.e., the max or $mean$ pooling type. Note that the number of convolutional layers in the DB is known because it can be derived by the spatial sizes of input and output, as well as k. Accordingly, the proposed encoding strategy is based on three different types of units and their positions in the CNNs. The units are the RB unit (RBU), the DB unit (DBU), and the pooling layer unit (PU). Specifically, an RBU and a DBU contain multiple RBs and DBs, respectively, while a PU is composed of only a single pooling layer. Our justifications are that: 1) by putting multiple of RBs or DBs into an RBU or a DBU, the depth of the CNN can be significantly changed compared with stacking RBs or DBs one-by-one, which will speed up the heuristic search of the proposed algorithm by easily changing the depth of the CNN and 2) one PU consisting of a single pooling layer is more flexible than consisting of multiple pooling layers because the effect of multiple consequent pooling layers can be achieved by stacking multiple PUs. In addition, we also add one parameter to represent the unit type for the convenience of the algorithm implementation. In summary, the encoded information for an RBU is the type, the number of RBs, the input spatial size, and the output spatial size, which are denoted as type, amount, in, and out, respectively. On the other hand, the encoded information for a DBU is the same as those of an RBU, in addition to the additional parameter k. Only one parameter is needed in a PU for encoding the pooling type.

Fig. 3 shows an example of the proposed algorithm in encoding a CNN containing nine units. Specifically, each number in the top-left corner of the block denotes the position of the unit in the CNN. The unit is an RBU, a DBU, or a PU if the type is 1, 2, or 3, respectively. Noting that the proposed encoding strategy does not constrain the maximal length of each individual, which means that the proposed algorithm can adaptively find the best CNN architecture with a proper depth through the designed variable-length encoding strategy.

C. Fitness Evaluation

The fitness of the individuals provides a quantitative measurement indicating how well they adapt to the environment and is calculated based on the information that these individuals encode and the task at hand. In AE-CNN, the fitness of an individual is the classification accuracy based on the architecture encoded by the individual and the corresponding validation data. According to the principle of evolutionary algorithms, an individual with a higher fitness has a higher probability to generate an offspring hopefully with even higher fitness than itself. For evaluating the fitness, each individual in AE-CNN is decoded to the corresponding CNN and then added to a classifier to be trained like that of a common CNN. Typically, the widely used classifier is the logistic regression for binary classification and the softmax regression for multiple classifications. As formulated by (1), in AE-CNN, the decoded CNN is trained on the training data, and the fitness is the best classification accuracy on the validation data after the CNN training.

Algorithm 3 Evaluate Fitness

Input: The population P_t for fitness evaluation, traing data $\mathcal{D}_{\text{train}}$, validation data $\mathcal{D}_{\text{valid}}$.
Output: The population P_t with fitness.

1 **for** *each individual in* P_t **do**
2 $cnn \leftarrow$ Transform the information encoded in $individual$ to a CNN with the corresponding architecture;
3 Initialize the weights of cnn;
4 Train cnn on $\mathcal{D}_{\text{train}}$;
5 $acc \leftarrow$ Evaluate the classification accuracy of the trained cnn on $\mathcal{D}_{\text{valid}}$;
6 Assign acc as the fitness of $individual$;
7 **end**
8 **Return** P_t.

The fitness evaluation of the proposed algorithm is shown in Algorithm 3, where each individual in the population is evaluated in the same manner. First, the architecture information encoded in the individual is transformed into a CNN with the corresponding architecture (see line 2), which is an inverse of the encoding strategy introduced in Section III-B. Second, the CNN is initialized with weights (see lines 3) like that of a hand-crafted CNN and then trained on the provided training data (see line 4). Noting that the weight initialize method and the training method are the Xavier initializer [43] and the SGD with momentum, respectively, which are commonly used in the deep learning community. Finally, the trained CNN is evaluated on the validation data (see line 5), and the evaluated classification accuracy is considered as the fitness of the individual (see line 6).

D. Offspring Generation

In order to generate a population of offspring, parent individuals need to be chosen in advance. Based on the principle of evolutionary algorithms, the generated offspring are expected

to have higher fitness than their parents, through inheriting the quality traits from both parents. To this end, the individuals having the best fitness should be chosen as the parent individuals. However, adopting the best ones as the parents could easily cause the loss of diversity in the population, which in turn leads to the premature convergence [44], [45], and as a result, the best performance of the population cannot be achieved [46], [47] due to trapping into the local minima [48], [49]. To address this problem, a general way is to select promising parents via a random way. In the proposed AE-CNN algorithm, the binary tournament selection [50] is used for this purpose [50], [51] based on the conventions of the GA community. The binary tournament selection randomly selects two individuals from the population, and the one with a higher fitness is chosen as one parent individual. By repeating this process again, another parent individual is chosen, and then, these two parent individuals perform the crossover operation. Noting that two offspring are generated after each crossover operation, and N offspring are generated in each generation, i.e., the crossover operation is performed $N/2$ times during each generation where N stands for the population size.

In traditional GAs, the crossover operation is performed on two individuals with the same length, which is biologically evident. Based on the proposed encoding strategy, individuals in the proposed algorithm have different lengths, i.e., the corresponding CNNs are with different depths. In this regard, the traditional crossover operator cannot be used. However, the crossover operator often refers to the local search ability of GAs, exploiting the search space for a promising performance. The performance of the final solution may be deteriorated due to the lacking of the crossover operation in GAs. In the proposed algorithm, we employ the one-point crossover operator. The reason is that the one-point crossover has been widely used in GP [31]. GP is another important class of evolutionary algorithms, and the individuals in GP are common with different lengths. Algorithm 4 shows the crossover operation in the proposed algorithm.

Noting that some necessary changes are automatically made on the generated offspring if required. For example, the in of the current unit should be equal to the out of the previous unit, and other cascade adjustments are caused by this change. For a better understanding of the crossover operation, an example is shown in Fig. 4, where Fig. 4(a) shows the two parent individuals. Supposing the separation positions of these two parent individuals are the third and fourth units, respectively, and then, Fig. 4(b) shows the corresponding generated offspring, and the red numbers imply the corresponding changes needed after the crossover operation for the logic representing a valid CNN.

The mutation operation typically performs the global search in GAs, exploring the search space for promising performance. It works on one generated offspring with a predefined probability and the allowed mutation types. Available mutation types are designed based on the proposed encoding strategy. In the proposed algorithm, the available mutation types are as follows:

Algorithm 4 Crossover Operation of AE-CNN

Input: Two parent individuals, p_1 and p_2, selected by the binary tournament selection, crossover propability μ.
Output: Two offspring.

```
r ← Uniformly generate a number from [0, 1];
if r < μ then
    Randomly choose a position from p1 and p2, respectively;
    Separate p1 and p2 based on the chosen positions;
    q1 ← Combine the first part of p1 and the second part of p2;
    q2 ← Combine the first part of p2 and the first part of p1;
else
    q1 ← p1;
    q2 ← p2;
end
Return q1 and q2.
```

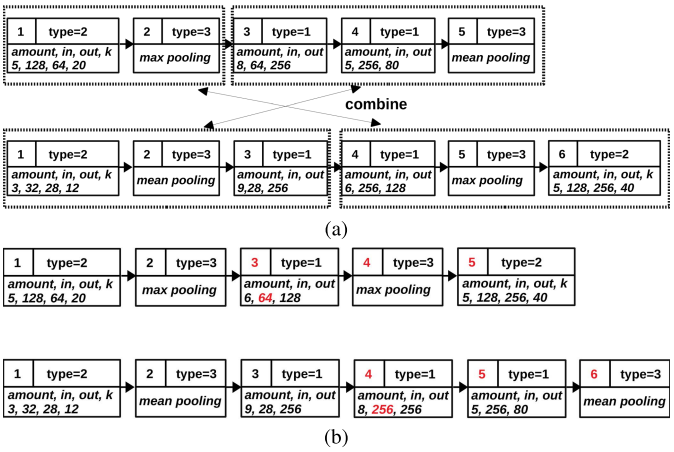

Fig. 4. (a) Two selected parent individuals for the crossover operation and (b) generated offspring. The numbers in each block denote the corresponding configuration, and the red numbers in Fig. 4(b) denote the necessary changes after the crossover operation.

1) adding (adding an RBU, adding a DBU, or adding a PU to the selected position);
2) removing (removing the unit at the selected position);
3) modifying (modifying the encoded information of the unit at the selected position).

The mutation operation in the proposed algorithm is detailed in Algorithm 5. Because all the generated offspring use the same routine for the mutation, Algorithm 5 shows only the process of one offspring for the reason of simplicity. Noting that the offspring will be kept the same if it is not mutated. In addition, a series of necessary adjustments will also be automatically performed based on the logic of composing a valid CNN as highlighted in the crossover operation. For better understanding the mutation, an example in terms of the "adding an RBU" is shown in Fig. 5, where Fig. 5(a) shows the selected individual for the mutation and the randomly

Algorithm 5 Mutation Operation of AE-CNN

Input: The offspring q_1, mutation propability v.
Output: The mutated offspring.

$r \leftarrow$ Uniformly generate a number from [0, 1];
if $r < v$ **then**
  Randomly choose a position from q_1;
  $type \leftarrow$ Randomly select one from {Adding, Removing, Modifying};
  if *type is Adding* **then**
    $mu \leftarrow$ Randomly select one from {adding an RBU, adding a DBU, adding a PU}
  else if *type is Removing* **then**
    $mu \leftarrow$ removing a unit;
  else
    $mu \leftarrow$ modifying the encoded information;
  end
  Perform mu at the chosen position;
Return q_1.

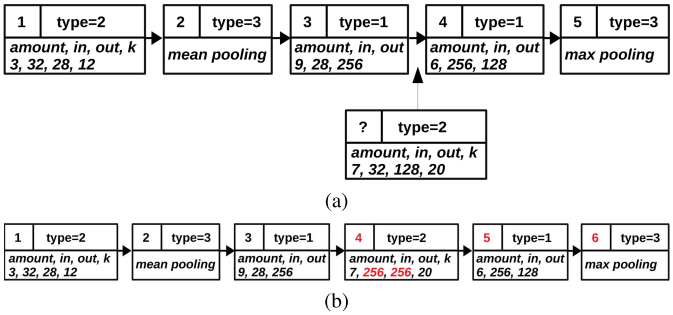

Fig. 5. Example of the "adding an RBU" mutation. (a) First and second rows denote the selected individual for the mutation and the randomly initialized RBU for the "adding and RBU" mutation at the fourth position of the individual to be mutated. (b) Mutated individual. The red numbers denote the necessary changes after the mutation.

initialized RBU, and Fig. 5(b) shows the mutated individual. The red numbers in Fig. 5(b) also mean the necessary changes when the mutation has been performed. In the proposed crossover and mutation operations, all these necessary changes are made automatically.

E. Environmental Selection

In the environmental selection, a population of individuals in the size of N is to be selected from the current population, i.e., $P_t \cup Q_t$, serving as the parent individuals for the next generation. Theoretically, a good population has the characteristics of both convergence and diversity [30] to prevent from trapping into local minima [48], [49] and premature convergence [44], [45]. In practice, the parent individuals should be composed of individuals with the best fitness for the convergence and individuals whose fitness has significant differences from each other for diversity. To this end, we will purposely select the individual with the best fitness, along with $N-1$ individuals which are selected by binary tournament selection [50], [51], as parent individuals to generate offspring for the new population. Explicitly selecting the best one as the parent for the next generation is an implementation of the "elitism" mechanism [52] in GAs, which could prevent the performance of the population from degrading as the evolutionary progresses.

Algorithm 6 Environmental Selection

Input: The population P_t, the generated offspring population Q_t, the population size N.
Output: The population P_{t+1} surviving in the next generation.

1 $P_{t+1} \leftarrow \emptyset$;
2 **for** $j \leftarrow 1$ ***to*** N **do**
3 $p_1, p_2 \leftarrow$ Randomly selected two individuals from $P_t \cup Q_t$;
4 $p \leftarrow$ Select the one with higher fitness from $\{p_1, p_2\}$;
5 $P_{t+1} \leftarrow P_{t+1} \cup p$;
6 **end**
7 $p_{\text{best}} \leftarrow$ Select the one with the highest fitness from $P_t \cup Q_t$;
8 **if** p_{best} *is not in* P_{t+1} **then**
9 Randomly select one from P_{t+1} and then replace it by p_{best};
10 **end**
11 **Return** P_{t+1}.

Algorithm 6 shows the details of the environmental selection in the proposed algorithm. Specifically, given the current population P_t and the generated offspring population Q_t, N individuals are selected with the binary tournament selection that are shown in lines 2–6. After that, the best individual p_{best} (i.e., the individual having the highest fitness) is selected from $P_t \cup Q_t$ (see line 7) and then to check whether p_{best} has been selected into P_{t+1} or not. A random one selected from P_{t+1} will be replaced by p_{best} if it does not exist in P_{t+1} (see lines 8–10). Noting that the offspring in Q_t should have been evaluated for their fitness prior to the environmental selection because the binary tournament selection works based on the fitness.

IV. Experiment Design

The experiment is purposely designed to verify whether the proposed automatic CNN architecture design algorithm is able to achieve a promising performance on image classification tasks. In this section, we will first introduce the chosen peer competitors (in Section IV-A) to which the performance of the proposed algorithm is compared, and then highlight the adopted benchmark data sets (in Section IV-B) and the parameter settings (in Section IV-C).

A. Peer Competitors

In order to demonstrate the superiority of the proposed algorithm, various peer competitors are chosen to perform the comparison. Particularly, the chosen peer competitors can be divided into three different categories.

The first includes the state-of-the-art CNNs whose architectures are hand-crafted with extensive domain expertise: DenseNet [18], ResNet [17], Maxout [53], VGG [54], Network in Network [55], Highway Network [56], All-CNN [57],

and FractalNet [58]. In addition, considering the promising performance of ResNet, we use two different versions in the experiment, they are the ResNet with 101 layers and ResNet with 1202 layers, which are labeled as ResNet (depth = 101) and ResNet (depth = 1,202), respectively. Owing to the promising performance, most peer competitors in this category win the champions of the large-scale vision challenge [59] in recent years. The intention of choosing these state-of-the-art CNNs is to verify if the proposed automatic CNN architecture design algorithm can show competitive performance to the hand-crafted CNNs. The second covers the CNN architecture design algorithms with a semiautomatic means, including Genetic CNN [19], Hierarchical Evolution [20], EAS [21], and Block-QNN-S [22]. The third refers to Large-scale Evolution [23], CGP-CNN [24], NAS [25], and MetaQNN [26], which design CNN architectures in a completely automatic way.

B. Benchmark Data Sets

The CNNs typically perform image classification tasks to compare their performance by looking at the classification performance. For the state-of-the-art CNNs, the most used image classification benchmark data sets are CIFAR10 and CIFAR100 [29], while for the CNN architecture design algorithms, the widely used benchmark data set is only CIFAR10 because CIFAR100 is much more challenging due to its large number of classes for the classification tasks at hand. Considering the adopted peer competitors covering the state-of-the-art CNNs and architecture design algorithms, both CIFAR10 and CIFAR100 are chosen as the benchmark data sets in the experiment.

CIFAR10 and CIFAR100 are two widely used image classification benchmark data sets for recognizing nature objects, such as bird, boat, and airplane. Each set has 50 000 training images and 10 000 test images. The differences between CIFAR10 and CIFAR100 are that CIFAR10 is 10-class classification, while CIFAR100 is 100-class classification. However, each benchmark has nearly the same number of training images for each class, i.e., each category of CIFAR10 has 5000 training images, while that of CIFAR100 has 500 training images.

Fig. 6 shows the images from each benchmark for reference, where images in each row denote the ones from the same class, and the words in the left column refer to the corresponding class name. As can be seen from Fig. 6, the object to be recognized in each image has the different resolution to each other, mixes with the background, and occupies the different position, which generally increases the difficulty in correctly recognizing the objects. Based on the conventions of the chosen peer competitors [17]–[26], CIFAR10 and CIFAR100 are augmented by padding four zeros to each side of one image and then randomly cropped to the original size followed by a randomly horizontal flip, prior to be input to the proposed algorithm.

C. Parameter Settings

In comparison, we extract the results of the peer competitors reported in their seminal papers rather than performing them by ourselves. The reason is that the results reported are usually the best. In doing so, there is no need to set the parameters of the peer competitors. For the proposed algorithm, we follow the principle that all the parameters are set based on their commonly used values, to lower the difficulty to researchers, who would like to use the proposed algorithm in finding the best CNN architectures for their investigated data, even though they have no expertise in GAs. Particularly, the population size and the maximal generation number are set to be 20, and the probabilities of crossover and mutation are set to 0.9 and 0.2, respectively. Based on the conventions of the machine learning community, the validation data are randomly split from the training data with the proportion of 1/5. Finally, all the classification error rates are evaluated on the same test data for the comparison.

Fig. 6. Randomly selected examples from each of the three categories of (a) CIFAR10 and (b) CIFAR100, and each category has ten examples.

In evaluating the fitness, each individual is trained by SGD with a batch size of 128. The parameter settings for SGD are also based on the conventions from the peer competitors. Specifically, the momentum is set to 0.9. The learning rate is initialized to 0.01, but with a warming up setting of 0.1 during the second to the 150th epoch, and scaled by dividing 10 at the 250th epoch. The weight decay is set to 5×10^{-4}. In addition, the fitness of the individual is set to zero if it is out of memory during the training. When the evolutionary process terminates, the best individual is retrained on the original training data with the same SGD settings, and the error rate on the test data is reported for the comparison. Considering the heuristic nature of the proposed algorithm as well as the expensive computational cost, the best individual is trained for five independent runs. Because all the peer competitors chosen for the comparisons only show their best results no matter how many times they have performed, the best result of the proposed algorithm among the five independent trials is presented here for a fair comparison.

In addition, the available choices of k in the DB are 12, 20, and 40 based on the design of DenseNet, and the maximal convolutional layers in the DB are specified as 10 (when $k = 12$ and $k = 20$) and 5 (when $k = 40$). Both the maximal numbers of RBUs and DBUs in a CNN are set to 4. Both the numbers of DBs and RBs in a DBU and an RBU, respectively, are set from 3 to 10. Noting that these settings are mainly based on our available computational resources

TABLE I

COMPARISONS BETWEEN THE PROPOSED ALGORITHM AND THE STATE-OF-THE-ART PEER COMPETITORS IN TERMS OF THE CLASSIFICATION ERROR (%), THE NUMBER OF PARAMETERS, AND THE CONSUMED GPU DAYS ON THE CIFAR10 AND CIFAR100 BENCHMARK DATA SETS

	CIFAR10	CIFAR100	# of Parameter	GPU Days	
DenseNet (k=12) [18]	5.24	24.42	1.0M	–	hand-crafted architecture
ResNet (depth=101) [17]	6.43	25.16	1.7M	–	hand-crafted architecture
ResNet (depth=1,202) [17]	7.93	27.82	10.2M	–	hand-crafted architecture
Maxout [53]	9.3	38.6	–	–	hand-crafted architecture
VGG [54]	6.66	28.05	20.04M	–	hand-crafted architecture
Network in Network [55]	8.81	35.68	–	–	hand-crafted architecture
Highway Network [56]	7.72	32.39	–	–	hand-crafted architecture
All-CNN [57]	7.25	33.71	–	–	hand-crafted architecture
FractalNet [58]	5.22	22.3	38.6M	–	hand-crafted architecture
Genetic CNN [19]	7.1	29.05	–	17	semi-automatic algorithm
Hierarchical Evolution [20]	3.63	–	–	300	semi-automatic algorithm
EAS [21]	4.23	–	23.4M	10	semi-automatic algorithm
Block-QNN-S [22]	4.38	20.65	6.1M	90	semi-automatic algorithm
Large-scale Evolution [23]	5.4	–	5.4M	2,750	completely automatic algorithm
Large-scale Evolution [23]	–	23	40.4M	2,750	completely automatic algorithm
CGP-CNN [24]	5.98	–	2.64M	27	completely automatic algorithm
NAS [25]	6.01	–	2.5M	22,400	completely automatic algorithm
MetaQNN [26]	6.92	27.14	–	100	completely automatic algorithm
AE-CNN	4.3	–	2.0M	27	completely automatic algorithm
AE-CNN	–	20.85	5.4M	36	completely automatic algorithm

because any number beyond these settings will easily render out of the memory. If the user' computational platform is equipped with more powerful GPUs, they can set the number to an arbitrary one. The proposed algorithm for the experiment is performed on three GPU cards with the model of Nvidia GeForce GTX 1080 Ti, and the codes are implemented based on a GPU-based parallel framework designed in our previous work written by PyTorch [60]. The codes are made available at https://gitlab.ecs.vuw.ac.nz/yanan/ea-cnn.

V. EXPERIMENTAL RESULTS

In the experiments, we investigate the performance of the proposed algorithm in terms of not only the classification error but also the number of parameters, as well as the computational complexity for a comprehensive comparison to the chosen peer competitors (shown in Section V-A). Because it is hard to theoretically analyze the computational complexity of each peer competitor, the consumed "GPU Days" is used as an indicator of the computational complexity. Specifically, the number of GPU Days is calculated by multiplying the number of employed GPU cards and the days the algorithms performed for finding the best architectures. For example, the proposed algorithm performed nine days on three GPU cards for the CIFAR10 data set, and therefore, the corresponding GPU Days is 27 by multiplying nine (days) with three (used GPU cards). Obviously, the state-of-the-art CNNs hand-crafted do not have the data regarding the "GPU days." In addition, we also provide the evolutionary trajectories of the proposed algorithm in finding the best architectures on the chosen benchmark data sets, which could help the readers to know whether the proposed algorithm converges with the adopted parameter settings (shown in Section V-B). Finally, the found best architectures are provided in Section V-C, which may provide useful knowledge to researchers in hand-crafting CNN architectures.

A. Performance Overview

Table I shows the experimental results of the proposed algorithm and the chosen peer competitors. In order to conveniently investigate the comparisons, Table I is divided into five "rows" by six horizontal lines. The first denotes the title of each column, and the second, third, and fourth rows refer to the state-of-the-art peer competitors whose architectures are manually designed, semiautomatic and automatic CNN architecture design algorithms, respectively. The fifth row shows the results of the proposed algorithm, which is an automatic algorithm in designing CNN architectures. In addition, the symbol "–" in Table I implies there is no result publicly reported by the corresponding peer competitor.

As shown in Table I, AE-CNN outperforms all the state-of-the-art peer competitors manually designed for CIFAR10. Specifically, AE-CNN achieves the classification error of approximately 1.0% lower than DenseNet ($k = 12$) and FractalNet, 2.1% lower than ResNet (depth = 101), VGG, and All-CNN, 3.5% lower than ResNet (depth = 1202) and Highway Network, and even 5.0% lower than Maxout and Network in Network. On CIFAR100, AE-CNN shows significantly lower classification error than Maxout, Network in Network, Highway Network, and All-CNN, slightly lower classification error than DenseNet ($k = 12$), ResNet (depth = 101), ResNet (depth = 1202), and VGG, and similar to but still better than the performance of FractalNet. The number of parameters of the CNN evolved by AE-CNN on both CIFAR10 and CIFAR100 is larger than DenseNet ($k = 12$) and ResNet (depth = 101) but much smaller than that of ResNet (depth = 1202), VGG, and FractalNet.

Compared with the semiautomatic peer competitors, AE-CNN performs much better than Genetic CNN on both CIFAR10 and CIFAR100. Although Hierarchical Evolution shows better performance than AE-CNN on CIFAR10, AE-CNN consumes only 1/10 GPU days as that consumed

by Hierarchical Evolution on CIFAR10. Block-QNN-S shows a bit worse performance on CIFAR10 but slightly better performance on CIFAR100 compared with AE-CNN, while AE-CNN consumes 1/3 of the GPU days as that consumed by Block-QNN-S, and also the best CNN found by AE-CNN has a smaller number of parameters than that of Block-QNN-S. In addition, EAS and AE-CNN perform nearly the same classification error on CIFAR10, while the best CNN evolved by AE-CNN only has 2.0M parameters, which is only 1/11 of that from EAS. In summary, compared with the semiautomatic peer competitors, AE-CNN shows the competitive performance but has a significantly lesser number of parameters. It is important to note that domain expertise is still required when using the algorithms from this category. For example, EAS only consumes ten GPU Days for the best CNN on CIFAR10, which is based on a base CNN with known fairly good performance. Therefore, the comparison in terms of the consumed GPU days is not fair to the proposed AE-CNN algorithm, which is completely automatic without using any human expertise and/or extra resources.

Among the automatic peer competitors, on both the CIFAR10 and the CIFAR100 data sets, AE-CNN shows the best performance in terms of the classification error, the number of parameters, and the consumed GPU days. Specifically, AE-CNN achieves 4.3% classification error on CIFAR10, while the best and worst classification errors from the peer competitors are 5.4% and 6.92%, respectively. In addition, AE-CNN also shows the lower classification error than that of MetaQNN. On CIFA100, AE-CNN shows 2.15% lower classification error than that of Large-scale Evolution and has a 5.4M number of parameters, which is much smaller than that of Large-scale Evolution (40.4M). Furthermore, AE-CNN also consumes much less GPU Days than that of Large-scale Evolution, NAS, and MetaQNN on both CIFAR10 and CIFAR100. The comparison shows that the proposed algorithm achieves the best performance among the automatic peer competitors to which the proposed algorithm belongs.

The rationale for AE-CNN outperforming Large-scale Evolution, CGP-CNN, NAS, and Meta-QNN can be justified as follows. First, Large-scale Evolution does not apply the crossover operator, which provides the local search ability. The GA-based design consequently deteriorates its performance. Second, CGP-CNN employs a fixed-length encoding strategy to design the best CNN architecture. In order to make the encoding strategy work, CGP-CNN must predefine a maximal length of CNNs during the architecture design. As can be seen from [24], the predefined maximal length of CGP-CNN is smaller than the best one identified by AE-CNN. Finally, NAS and Meta-QNN are designed based on reinforcement learning. Because the fitness value is not computed when the reinforcement learning methods are used, the reinforcement learning-based methods often consume more computational resources than GA does for the same performance [7]. Expectedly, NAS and Meta-QNN perform worse than AE-CNN given the available computational resources.

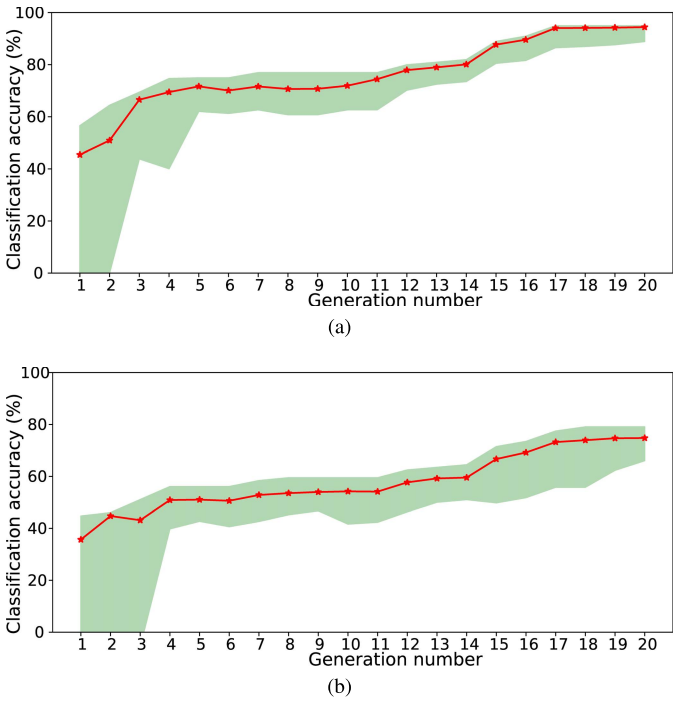

Fig. 7. Evolution trajectories of the proposed algorithm in (a) CIFAR10 and (b) CIFAR100.

B. Evolution Trajectory

When the evolutionary algorithms are used to address real-world problems, we usually like to know whether they have converged or not when they terminate. A better way to observe this is to plot the evolutionary trajectories. In this section, the evolutionary trajectories of the proposed algorithm in terms of the investigated benchmark data sets are provided and analyzed. To achieve this, we first collect the classification accuracy of each individual in every generation and then plot the statistical results.

The evolutionary trajectories of the proposed algorithm are shown in Fig. 7, where Fig. 7(a) and (b) show those on CIFAR10 and CIFAR100, respectively. In Fig. 7, the horizontal axis denotes the generation number, and the vertical axis denotes the classification accuracy; the red line denotes the mean classification accuracy of the individuals in the same generation, while the light-green area is contoured by the best and worst classification accuracy of the individuals in each generation.

As can be seen from Fig. 7(a), the mean classification accuracy sharply increases from the first generation to the third generation and, then, steadily improves, as the evolution process proceeds until the 14th generation; from then, the mean classification accuracy has a significant increase from about 75% to about 95%. Finally, the proposed algorithm converges when it terminates. As can be seen from the lower boundary of the light-green area, the worst classification accuracy in the first two generations is zero, which is caused because the randomly initialized architecture cannot run on the employed GPUs due to the out-of-memory problem; from the third generation, the individuals with the out-of-memory

architectures are eliminated from the population due to their uncompetitive fitness, and the classification accuracy steadily improves until the algorithm terminates, although there is an exception at the fourth generation. As can be seen from the upper boundary of the light-green area, the best performance almost keeps the same improvement, as the mean classification accuracy with the evolutionary process continues. In addition, the difference between the best classification accuracy and the worst accuracy also becomes smaller, which implies that the population converges to a steady state.

A similar situation can also be seen from Fig. 7(b). Specifically, the mean classification accuracy increased from about 30% to about 45% from the first generation to the fourth generation, although there is a slight drop at the third generation. Since the fourth generation, the mean classification accuracy keeps improving until the 14th generation and, then, increases from about 50% at the 14th generation to about 79% at the 17th generation; after that, the mean classification accuracy converges until the evolutionary process terminates. During the first three generations, the worst classification accuracy stays at zero because of the randomly initialized out-of-memory individuals; from the 4th generation, the worst classification accuracy improves until the 20th generation with the exception at the 10th and 15th generations. As can be seen from the evolutionary trajectories of the best classification accuracy, the best classification accuracy improves almost with the same trend as that of the mean classification accuracy and also archives the converged performance from the 17th generation.

A common trend can both be seen from Fig. 7(a) and (b) that the best classification accuracy (i.e., the upper boundaries of the light-green areas) will not be degraded, which is achieved through the utilized elitism detailed in Section III-E, i.e., the individual with the best fitness is unconditionally kept into the next generation. In summary, the proposed algorithm converges within the default parameter settings in terms of GAs, which could help the users to employ the proposed algorithm to find the best CNN architectures for their own data, even though the users have no expertise in GAs. However, the maximal generation number and the population size can be set to larger numbers if more computational resources are available.

C. Designed CNN Architectures

In this section, the best CNN architectures found by the proposed algorithm on CIFAR10 and CIFAR100 are provided in Tables II and III, respectively.

As can be seen from Tables II and III, the best architecture on CIFAR10 is composed of nine units that are designed in the proposed encoding strategy in Section III-B, and altogether, it has 38 layers that consist of 34 convolutional layers and four pooling layers; while the best architecture on CIFAR100 is composed of six units that consist of 21 layers, containing 17 convolutional layers and four pooling layer.

Compared with the state-of-the-art CNNs that are solely built on DBs or RBs, the automatically found architectures based on both blocks have much simpler architectures and much better performance. This may serve as *a priori* knowledge in hand-crafting CNN architectures in which ensemble blocks may be more effective. In addition, CIFAR100 is commonly viewed as a more complex benchmark than CIFAR10, and researchers usually consider CNN architectures with more layers than that of CIFAR10 when dealing with CIFAR100. However, based on the found architectures shown in Tables II and III, the best architecture for CIFAR100 has surprisingly a smaller number of layers than that of CIFAR10. To this end, finding the best architecture through evolution search may also provide useful domain expertise, which is in contrast to our common sense.

TABLE II
INFORMATION OF THE BEST ARCHITECTURE FOUND ON CIFAR10

id	type	configuration
1	RBU	amount=8, in=3, out=64
2	PU	mean pooling
3	RBU	amount=5, in=64, out=28
4	PU	mean pooling
5	RBU	amount=7. in=128, out=64
6	DBU	amount=7, in=64, out=204, k=20
7	DBU	amount=7, in=204, out=204, k=20
8	PU	mean pooling
9	PU	max pooling

TABLE III
INFORMATION OF THE BEST ARCHITECTURE FOUND ON CIFAR100

id	type	configuration
1	DBU	amount=10, in=3, out=203, k=20
2	PU	max pooling
3	PU	mean pooling
4	RBU	amount=7, in=203, out=256
5	PU	mean pooling
6	PU	mean pooling

VI. CONCLUSION AND FUTURE WORK

The goal of this paper is to develop a CNN architecture design algorithm by using GAs, which is capable of designing/searching/learning/evolving the best CNN architecture for the given task in a completely automatic manner and based on the limited computational resource. This goal has been successfully achieved by the proposed encoding strategy built on the state-of-the-art blocks with a variable-length representation, presenting a crossover operator for the variable-length individuals and the corresponding mutation operators. Building upon the blocks is able to speed up the CNN architecture design. The variable-length of individuals can adaptively evolve the proper depth of a CNN for tasks with different complexities. The presented crossover operator and the designed mutation operators provide the proposed algorithm with effective local search and global search ability, which in turn helps the proposed algorithm to be able to find the best CNN architectures. The proposed algorithm is examined on CIFAR10 and CIFAR100 image classification data sets, against nine state-of-the-art CNNs manually designed, four peer competitors designing CNN architectures with a semiautomatic way, and five peer competitors designing CNN architectures with the completely automatic way. The results show that the proposed algorithm outperforms all the state-of-the-art CNNs

hand-crafted and all the peer competitors from the automatic category in terms of the classification error rate. In addition, the proposed algorithm also consumes a much smaller number of GPU Days than the peer competitors in the same category. Furthermore, the proposed algorithm shows competitive performance against the semiautomatic peer competitors. Our future work will focus on effectively speeding up the fitness evaluation.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, Lake Tahoe, NV, USA., 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.*, Vancouver, BC, Canada, May 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.*, Montreal, QC, Canada, 2014, pp. 3104–3112.

[5] C. Clark and A. Storkey, "Training deep convolutional neural networks to play go," in *Proc. 32nd Int. Conf. Mach. Learn. (ICML)*, Lille, France, Jun. 2015, pp. 1766–1774.

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

[7] Y. Sun, B. Xue, and M. Zhang, "Evolving deep convolutional neural networks for image classification," 2017, *arXiv:1710.10741*. [Online]. Available: https://arxiv.org/abs/1710.10741

[8] Y. Sun, G. G. Yen, and Z. Yi, "Evolving unsupervised deep neural networks for learning meaningful representations," *IEEE Trans. Evol. Comput.*, vol. 23, no. 1, pp. 89–103, Feb. 2018. doi: 10.1109/TEVC.2018.2808689.

[9] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 26, no. 1, pp. 29–41, Feb. 1996.

[10] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.

[11] C. E. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning*, vol. 1. Cambridge, MA, USA: MIT Press, 2006.

[12] J. Močkus, "On Bayesian methods for seeking the extremum," in *Proc. IFIP Tech. Conf. Optim. Techn.* Berlin, Germany: Springer, 1975, pp. 400–404.

[13] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, Granada, Spain, 2011, pp. 2546–2554.

[14] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Proc. Int. Conf. Learn. Intell. Optim.*, vol. 5, Jan. 2011, pp. 507–523.

[15] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.

[16] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "An experimental study on hyper-parameter optimization for stacked auto-encoders," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2018, pp. 1–8.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 770–778.

[18] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, Jul. 2017, pp. 2261–2269.

[19] L. Xie and A. Yuille, "Genetic CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Venice, Italy, Oct. 2017, pp. 1388–1397.

[20] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *Proc. ICLR*, Stockholm, Sweden, Feb. 2018, pp. 1–13.

[21] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proc. 32nd AAAI Conf. Artif. Intell.*, New Orleans, LA, USA, Apr. 2018, pp. 1–8.

[22] Z. Zhao, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, Jun. 2018, pp. 2423–2432.

[23] E. Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 2902–2911.

[24] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proc. Genetic Evol. Comput. Conf.*, Berlin, Germany, Jul. 2017, pp. 497–504.

[25] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, Toulon, France, 2017, pp. 1–16.

[26] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, Toulon, France, Mar. 2017, pp. 1–18.

[27] T. Back, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. England, U.K.: Oxford Univ. Press, 1996.

[28] R. S. Sutton and A. G. Barto, *Reinforcement Learning—An Introduction*, vol. 1. Cambridge, MA, USA: MIT Press, 1998.

[29] A. Krizhevsky and G. Hinton. (2009). *Learning Multiple Layers of Features from Tiny Images*. [Online]. Available: http://www.cs.toronto.edu/kriz/cifar.html

[30] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1975.

[31] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, *Genetic Programming: An Introduction*, vol. 1. San Mateo, CA, USA: Morgan Kaufmann, 1998.

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

[33] 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.

[34] 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. 2018. doi: 10.1109/TEVC.2018.2791283.

[35] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1998.

[36] L. M. Schmitt, "Theory of genetic algorithms," *Theor. Comput. Sci.*, vol. 259, nos. 1–2, pp. 1–61, May 2001.

[37] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Proc. 29th Annu. Conf. Neural Inf. Process. Syst.*, Montreal, QC, Canada, 2015, pp. 2377–2385.

[38] E. Orhan and X. Pitkow, "Skip connections eliminate singularities," in *Proc. ICLR*, Stockholm, Sweden, 2018, pp. 1–22.

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

[40] G. C. Cawley and N. L. Talbot, "On over-fitting in model selection and subsequent selection bias in performance evaluation," *J. Mach. Learn. Res.*, vol. 11, pp. 2079–2107, Jul. 2010.

[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] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Boston, MA, USA, Jun. 2015, pp. 1–9.

[43] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, Mar. 2010, pp. 249–256.

[44] Y. Leung, Y. Gao, and Z.-B. Xu, "Degree of population diversity—A perspective on premature convergence in genetic algorithms and its Markov chain analysis," *IEEE Trans. Neural Netw.*, vol. 8, no. 5, pp. 1165–1176, Sep. 1997.

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

[46] 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. doi: 10.1109/TEVC.2018.2794319.

[47] 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.
[48] L. Davis, *Handbook Of Genetic Algorithms*. New York, NY, USA: Van Nostrand Reinhold, 1991.
[49] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Mach. Learn.*, vol. 3, nos. 2–3, pp. 95–99, 1988.
[50] 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.
[51] G. Zhang, Y. Gu, L. Hu, and W. Jin, "A novel genetic algorithm and its application to digital filter design," in *Proc. IEEE Int. Conf. Intell. Transp. Syst.*, vol. 2, Oct. 2003, pp. 1600–1605.
[52] J. A. Vasconcelos, J. A. Ramirez, R. H. C. Takahashi, and R. R. Saldanha, "Improvements in genetic algorithms," *IEEE Trans. Magn.*, vol. 37, no. 5, pp. 3414–3417, Sep. 2001.
[53] 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.
[54] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 32nd Int. Conf. Mach. Learn. (ICML)*, Lille, France, Apr. 2015, pp. 1–14.
[55] M. Lin, Q. Chen, and S. Yan, "Network in network," in *Proc. Int. Conf. Learn. Represent.*, Banff, Canada, 2014, pp. 1–10.
[56] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," in *Proc. Int. Conf. Learn. Represent. Workshop*, San Diego, CA, USA, 2015, pp. 1–6.
[57] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," in *Proc. Int. Conf. Learn. Represent.*, San Diego, CA, USA, Apr. 2015.
[58] G. Larsson, M. Maire, and G. Shakhnarovich, "Fractalnet: Ultra-deep neural networks without residuals," in *Proc. 5th Int. Conf. Learn. Represent.*, 2016, pp. 1–11. [Online]. Available: https://openreview.net/forum?id=S1VaB4cex
[59] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.
[60] A. Paszke *et al.* (2017). *Automatic Differentiation in Pytorch*. [Online]. Available: https://openreview.net/forum?id=BJJsrmfCZ

Yanan Sun (S'15–M'18) received the Ph.D. degree in engineering from Sichuan University, Chengdu, China, in 2017.

He was a Research Fellow with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. He is currently a Professor (research) with the College of Computer Science, Sichuan University. His current research interests include evolutionary algorithms, deep learning, and evolutionary deep learning.

Dr. Sun is the leading Organizer of the First Workshop on Evolutionary Deep Learning, the leading Organizer of the 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 (M'10) 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 current research interests include 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, IEEE Computational Intelligence Society (CIS), the Vice-Chair of the IEEE CIS Data Mining and Big Data Analytics Technical Committee, and the Vice-Chair of the IEEE CIS Task Force on Transfer Learning and Transfer Optimization.

Mengjie Zhang (M'04–SM'10–F'18) received the B.E. and M.E. degrees from the Artificial Intelligence Research Center, Agricultural University of Hebei, Baoding, 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) of 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.

Dr. Zhang 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 also a Committee Member of the IEEE NZ Central Section. He is the Vice-Chair of the IEEE Computational Intelligence Society (CIS) Task Force on Evolutionary Feature Selection and Construction, the Vice-Chair of the Task Force on Evolutionary Computer Vision and Image Processing, and the Founding Chair of the IEEE Computational Intelligence Chapter, New Zealand.

Gary G. Yen (S'87–M'88–SM'97–F'09) received the Ph.D. degree in electrical and computer engineering from the University of Notre Dame, Notre Dame, IN, USA, in 1992.

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

Dr. Yen received the Andrew P Sage Best Transactions Paper Award from the IEEE Systems, Man and Cybernetics Society in 2011, and the Meritorious Service Award from the IEEE Computational Intelligence Society in 2014. 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. He served as the Vice President for the Technical Activities from 2005 to 2006 and then President from 2010 to 2011 of the IEEE Computational intelligence Society. He was an Associate Editor of the *IEEE Control Systems Magazine*, the IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY, *Automatica*, *Mechantronics*, the IEEE TRANSACTIONS ON SYSTEMS, MAN AND CYBERNETICS, PARTS A AND B, and the IEEE TRANSACTIONS ON NEURAL NETWORKS. He was the Founding Editor-in-Chief of the *IEEE Computational Intelligence Magazine* from 2006 to 2009. He is currently serving as an Associate Editor for the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and the IEEE TRANSACTIONS ON CYBERNETICS.