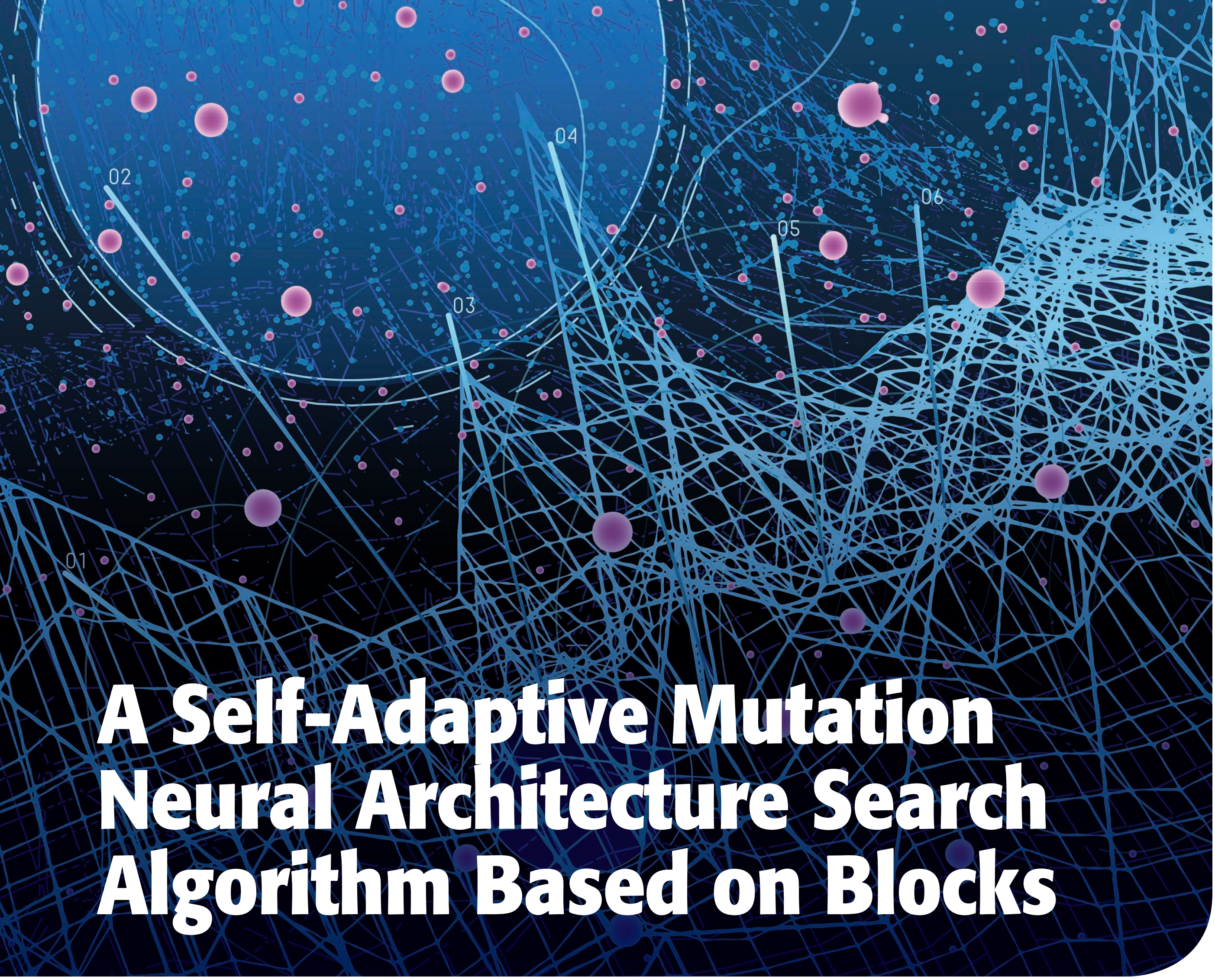

A Self-Adaptive Mutation Neural Architecture Search Algorithm Based on Blocks

©SHUTTERSTOCK.COM/BETTERVECTOR

Yu Xue and Yankang Wang
Nanjing University of Information Science and Technology, CHINA

Jiayu Liang
Tiangong University, CHINA

Adam Slowik
Koszalin University of Technology, POLAND

***Abstract*—**Recently, convolutional neural networks (CNNs) have achieved great success in the field of artificial intelligence, including speech recognition, image recognition, and natural language processing. CNN architecture plays a key role in CNNs' performance. Most previous CNN architectures are hand-crafted, which requires designers to have rich expert domain knowledge. The trial-and-error process consumes a lot of time and computing resources. To solve this problem, researchers proposed the neural architecture search, which searches CNN architecture automatically, to satisfy different requirements. However, the blindness of the search strategy causes a 'loss of experience' in the early stage of the search process, and ultimately affects the results of the later stage. In this paper, we propose a self-adaptive mutation neural architecture search algorithm based on ResNet blocks and DenseNet blocks. The self-adaptive mutation strategy makes the algorithm adaptively adjust the mutation strategies during the evolution process to achieve better exploration. In addition, the whole search process is fully automatic, and users do not need expert knowledge about CNNs architecture design. In this paper, the proposed algorithm is compared with 17 state-of-the-art algorithms, including manually designed CNN and automatic search algorithms on CIFAR10 and CIFAR100. The results indicate that the proposed algorithm outperforms the competitors in terms of classification performance and consumes fewer computing resources.

Digital Object Identifier 10.1109/MCI.2021.3084435
Date of current version: 15 July 2021

Corresponding Author: Yu Xue (e-mail: xueyu@nuist.edu.cn).

1556-603X/21©2021IEEE

I. Introduction

Deep learning has achieved great success in many fields [1], such as speech recognition [2], semantic segmentation [3], [4], image recognition [5], [6], and natural language processing [7]. With the excellent performance in these fields, convolutional neural networks (CNNs) have become one of the most widely used models in deep learning [8]. In general, conventional CNNs consist of convolutional layers, pooling layers, and fully-connected layers. Typical CNNs include AlexNet [9], VGG [10], GoogleNet [11], ResNet [5], and DenseNet [12]. Although these networks have achieved huge accuracy improvements in vision tasks, the design of the CNN architectures is still a difficult task because of the large number of parameters [13]. These models are all hand-crafted and they cannot learn the architecture by themselves, thus, designers need to have much expert knowledge in CNN architecture design [14]. Moreover, the design of CNN architecture is guided by problems; that is, the architectures of CNNs are decided by different data distributions, and a manually designed architecture lacks flexibility. The two main factors that affect the performance of CNNs are architectures and weights. For weight optimization, the gradient descent algorithm has been proved to offer a significant advantage [15]. Unfortunately, for the optimization of CNNs' architectures, there are no explicit functions to directly calculate the optimal architecture [16]. To solve the problems above, Google researchers Zoph *et al.* [17] proposed the concept of the neural architecture search (NAS) for deep neural networks. Following this concept, a large number of researchers have paid attention to this field, and it has become one of the most popular research topics in the automatic machine learning community. The purpose of NAS is to automatically search for parameters such as the optimal architectures of CNNs so that CNNs can outperform those that are hand-crafted. Moreover, the NAS can reduce the high cost of trial-and-error design.

The existing NAS algorithms can be categorized into three types: 1) reinforcement learning based methods [18]; 2) gradient based methods [19]; 3) evolutionary computation based [20] (ENAS) methods. Among these methods, the NAS method based on reinforcement learning requires a large number of computing resources, i.e., thousands of graphics processing units (GPUs) run on a medium-sized dataset such as CIFAR10 for more than 10 days. Zoph *et al.* [17] proposed a NAS method based on reinforcement learning. They used 800 GPUs and spent 28 days completing the entire search process, which is not practical for most interested users. The gradient-based method is more efficient than the reinforcement learning method, and one of the most famous works is proposed by Liu *et al.* [19]. Because of the lack of theoretical support, the architectures searched by them are inadequate and unstable, and the process of constructing a cell consumes significant computing resources and requires much prior knowledge. The ENAS algorithms use the evolutionary computation (EC) technique to design the CNN architecture. In particular, the EC is a population-based technique to obtain the global optimum. Many EC techniques have been proposed, such as genetic algorithm (GA) [21], particle swarm optimization (PSO) [22], and artificial ant colony algorithm [23]. Due to the gradient-free characteristic, the EC technique has been widely used in optimization problems [24] and even in black box problems that do not have mathematical equations explicitly expressed. Therefore, EC techniques have been used to solve NAS problems.

In fact, two decades ago, the EC technique was already used to optimize neural networks and it was called neuroevolution [25]. The goal is to use the EC technique to optimize the architecture and weights of neural networks. Neuroevolution is proposed only for small or medium scale neural networks [16]. With the development of neural networks, the CNNs based on deep neural networks (DNNs) have been proposed and widely used in computer vision tasks. However, the CNNs have a large number of parameters. Therefore, to solve the problem of designing CNNs' architectures, the concept of ENAS was proposed in the evolutionary computation community. According to references, the first ENAS work was completed by Google's Real *et al.* [26] and they proposed a LargeEvo algorithm. The LargeEvo algorithm uses the GA to design CNNs' architectures. However, they only use mutation operators. Experimental results have proven the effectiveness of LargeEvo on CIFAR10 and CIFAR100 [27]. LargeEvo directly evolves basic units in open space, then samples and evaluates each candidate solution, which consumes a lot of computational resources. Since Real *et al.* [26] used layers as the search space, this space contains a relatively simple architecture, LargeEvo initializes the model with the basic layers, and each individual contains only a fully connected layer. Xie *et al.* [28] searched through the same space, and their results indicated that this trivial search space is capable of evolving a competitive architecture. Yet, the large search space brings high computational cost. To overcome this phenomenon, some methods have been proposed to constrain the search space, such as block-based design methods. These methods combine the state-of-the-art architecture in the initial state, so it can greatly reduce the search space and maintain the performance. Many block-based methods have been proposed, such as ResNet blocks [5], DenseNet blocks [12], Inception blocks [11]. Block-based methods are quite promising because of the fewer parameters which greatly speed up the search process.

Some researchers directly used aforementioned blocks to design CNN architectures [29], [30], whereas others have proposed different kinds of blocks. For example, Chen *et al.* [31] proposed eight kinds of blocks, including ResNet block and Inception block, and encoded them into 3-bit strings. After that, they used the Hamming distance to identify blocks that have similar performance. Song *et al.* [32] proposed three types of residual dense blocks, which greatly constrain the search space and reduce computational consumption. After determining the search space, different evolution operators were employed to generate new architectures. Sun *et al.* [14] used polynomial mutation on the encoded information which is represented by real numbers. To make mutations less random, Lorenzo *et al.* [33] proposed a new Gaussian mutation guided by Gaussian regression. Gaussian regression can predict the architectures which have good performance, and can also search and sample in a search space where the fitness value may be better.

The studies above do not effectively record the evolution information, which leads to their inability to guide the entire search process based on past experience. Most of them tend to focus on the improvement of the search space and evaluation methods, but the search strategy also affects an algorithm when finding a competitive architecture. At the same time, the general search strategy loses the 'experience' in the entire search process, resulting in the algorithm being unable to obtain more favorable architectures. This paper focuses on finding a suitable way to guide the whole search process. For this purpose, this paper proposes a self-adaptive mutation based on blocks to design CNN architectures. The self-adaptive mechanism has been widely used in EC, but no one has applied it to the NAS. In addition, the traditional mutation strategy does not focus on retaining the evolution information. To choose an appropriate mutation strategy, the self-adaptive mechanism is added into the evolutionary process. To prevent the phenomenon of population degradation and slow convergence, a semi-complete binary competition selection strategy has also been designed and it can sufficiently prevent the promising individuals in the population from being easily abandoned. Moreover, the ENAS algorithm proposed in this paper is completely automatic, which means that it does not require the user to have rich expert knowledge about CNN architecture design.

The rest of the paper is organized as follows: Section II introduces related work. Section III describes the proposed algorithm in details. Section IV introduces the designs of the experiment and Section V analyzes the experimental results. Finally, Section VI provides our conclusions and directions for future work.

II. Related Work

A. Genetic Algorithm

The GA is an evolutionary algorithm inspired by the natural selection. Because it has gradient-free characteristics, it can be used to optimize nonconvex and nondifferentiable problems [21], [34]. In addition, the GA has the characteristic of being insensitive to local optima, which enables it to find global optima. The GA usually uses a series of biologically inspired operators to solve optimization problems, such as crossover and mutation. Generally speaking, the main steps of GA are as follows: 1) initialize a population; 2) evaluate fitness; 3) execute crossover and mutation operators; 4) evaluate individuals; 5) environmental selection. Among them, steps 3) - 5) are controlled by a loop, and the stop condition is whether the maximum generation is reached.

In this paper, each individual represents the architecture of one CNN, and the fitness is the classification accuracy of each CNN on the validation dataset. The higher the classification accuracy, the higher the fitness.

B. Block-Based Design Method

Because a constraint represents human intervention in the search space, the suitable constraints on the coding space are very important. A method with a large number of constraints can easily obtain a good architecture, but it also will be unable to find a novel architecture. In addition, the size of the search space greatly affects the efficiency of search techniques. With the development of the NAS, more researchers use block-based techniques to define the search space [35], [36], which combines various types of layers as the basic unit. These block-based techniques have good performance and fewer parameters are required to build the architecture.

ResNet and DenseNet are two kinds of DNNs with better performance in hand-designed CNNs. Their success depends on the construction of ResNet blocks and DenseNet blocks, and the block-based method has been proven to have better effectiveness than the design method which is composed of layers [32], [37], [38]. This design method can solve the problem of gradient disappearance [39]. Therefore, the block-based search space can achieve promising prospects. In this paper, ResNet blocks and DenseNet blocks are used as the basic units to determine the search space. For the convenience of representation, ResNet blocks and DenseNet blocks are respectively called RBs and DBs. The units composed of RBs or DBs are called RUs or DUs, respectively. Figure 1 shows the structure of an RU. Figure 2 shows the internal architecture of an RB. In this RB, there are three convolutional layers, which are called conv1, conv2, and conv3. In conv1, 1 × 1 convolution kernels are used to decrease the number of channels of input features and the computing complexity. In conv2, the 3 × 3 filters are used to learn the characteristics of the input. In conv3, the filters with the size of 1 × 1 are used to increase the number of output channels of the previous convolutional layer, so that the number of input and output channels is consistent throughout the entire RB. In addition, a

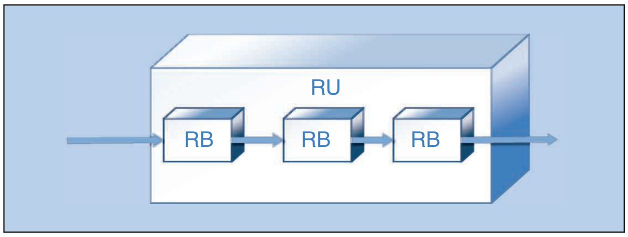

FIGURE 1 An example of the ResNet Unit.

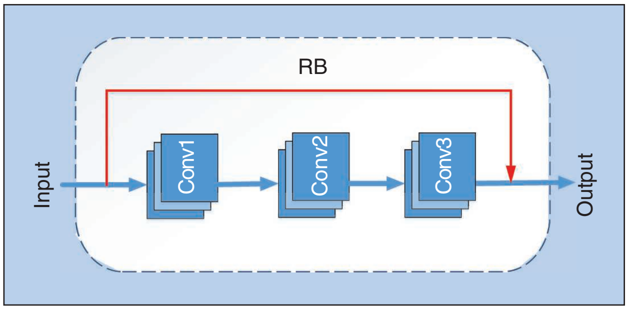

FIGURE 2 An example of the ResNet block with three convolutional layers and one skip connection.

The existing NAS algorithms can be categorized into three types: 1) reinforcement learning based methods; 2) gradient based methods; 3) evolutionary computation based (ENAS) methods.

skip-connection is used to connect the input layer and the output layer, and when the sizes of input and output channels are different, some 1 × 1 filters are added to adjust the size of channels so that the size of the input channels is the same as the size of output channels. Figure 3 shows an example of a DB. As shown in the figure, three convolutional layers are used to form a DB (the number of convolutional layers in each DB can be set by the users). In this DB, the input of each convolutional layer contains the output of each previous layer. In addition, to better control the input and output size of each convolutional layer, a parameter k is used. For example, if the input size of the first layer is a, then the output size of the first layer is $a+k$, the input size of the second layer is $a+k$, and the output size of the second layer is $a+2k$. Through similar operations, some DBs can be constructed. The two hierarchical representation block-based methods above can not only reduce the search space, but also enable deeper convolutional layers to learn features in the shallow layers. This hierarchical representation of input can also improve the classification accuracy and make the search process more flexible [37].

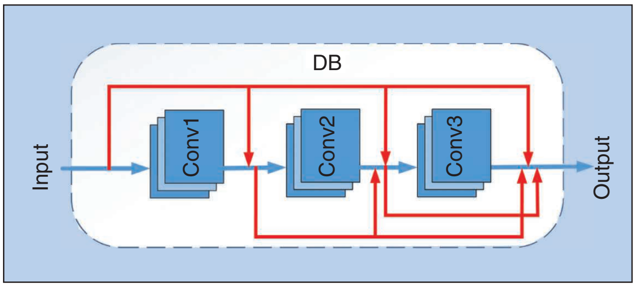

FIGURE 3 An example of the DenseNet block with three convolutional layers.

C. Self-Adaptive Mechanism

In recent years, self-adaptive mechanisms have been increasingly used in the evolutionary computation community. Moreover, many EC algorithms based on self-adaptive mechanisms have been proposed [34]. In the early years of such research, Qin *et al.* [40] introduced the self-adaptive mechanism into the differential evolution (DE) algorithm, combining different types of DE, and proposed a new algorithm called self-adaptive DE. The experimental results showed that self-adaptive DE can obtain higher quality solutions. In recent years, Xue *et al.* [24] applied a self-adaptive PSO to solve large-scale feature selection problems, and experimental results showed its effectiveness in reducing redundant features. In the field of neural networks, adaptive mechanisms, such as adaptive gradient algorithm [41], adadelta [42], and adam [43] have been applied for selecting the learning rate. These algorithms can adjust the learning rate to meet the needs of different learning rates in different training stages. To the best of our knowledge, there is no work that introduces the adaptive mechanism into the CNN architecture design. Therefore, this paper proposes a self-adaptive mutation method to search the CNNs' architectures so that the algorithm can adaptively choose appropriate mutation operations, such as adding, removing, and modifying units.

III. Details of SaMuNet

In this section, the framework of the algorithm is described, and its main components are discussed in details. A self-adaptive mutation network (SaMuNet) is proposed in this paper. The proposed NAS method is mainly used in image classification tasks. For clarity, the frequently used notations and their descriptions are shown in Table I.

A. Framework of SaMuNet

The framework of SaMuNet is shown in Algorithm 1, first, a population with size N is initialized. Then, each individual in the initialized population is evaluated. After that, the architecture of the CNN is evolved by the crossover and self-adaptive mutation operators. A new population will be generated by the environment selection operator. This process continues until the maximum generation number is reached. Finally, the best generated individual is decoded into a CNN with the best architecture. In the following parts, encoding strategy, population initialization, individual evaluation, crossover operators, self-adaptive mutation mechanism, and the proposed environment selection operator are introduced.

TABLE I Notations and Corresponding Descriptions.

NOTATIONS	DESCRIPTIONS	NOTATIONS	DESCRIPTIONS
N	Population size	P	Current population
$nIter$	Maximal iteration	Q	Offspring population
λ	Crossover probability	$nsflag$	Success matrix in a generation
μ	Mutation probability	$nfflag$	Fail matrix in a generation
D_{train}	Train datasets	S	Success matrix in defined generation
D_{valid}	Validation datasets	F	Fail matrix in defined generation
p_1, p_2	Two parent individuals	$COGS$	Candidate offspring generate strategy
q_1, q_2	Two offspring individuals	N_S	Number of strategies

B. Encoding Strategy

Encoding is very important when solving practical problems, and different problems require different coding strategies. For the EC technique, encoding strategies are generally divided into two types: direct encoding and indirect encoding. Direct encoding directly encodes the precise information into each individual. Indirect encoding is different from directly representing the weight or connection of the network, and it usually represents each individual through a new generation rule [16]. This encoding method is widely used in neuroevolution [44]. Because of the increasing number of network layers in CNNs, the number of parameters that need to be optimized has also increased exponentially. The backpropagation algorithm [15] has been proven to be superior in optimizing weights and it is directly applied to the weight optimization of the CNN. Thus, it is not necessary to consider the optimization of the weights of CNN, and only some hyperparameters need to be optimized.

> **In this paper, each individual represents the architecture of one CNN, and the fitness is the classification accuracy of each CNN on the validation dataset. The higher the classification accuracy, the higher the fitness.**

To make operators such as crossover and mutation easier to implement and maintain the flexibility of CNNs' architectures, this paper uses a direct variable-length encoding strategy to represent CNN architecture. This encoding strategy uses basic units RU, DU and PU. For the RU, a random number is used to represent the amount of RBs, and each RB contains at least one 3 × 3 convolution kernel. Each RB also includes a skip connection operation.

For the DU, a random number is used to represent the number of DBs, and each DB also contains at least one 3 × 3 convolution kernel. The difference between DU and RU is that the skip connections in the DB are densely connected (i.e., the input of each layer contains all the previous layer outputs). Meanwhile, each PU includes only one pooling layer of either type: average pooling layer and maximum pooling layer. In summary, RUs, DUs and PUs are composed of RBs, DBs and pooling layers, respectively. For different RBs, the unknown parameters are the number of input and output channels. For different DBs, in addition to the number of input and output channels, the unknown parameters include a growth rate k (to calculate the number of input channels for each layer). To express individual information and make the genetic operator more flexible, this article uses the variable-length encoding strategy [15]. As shown in Figure 4, an individual contains RUs, DUs, and PUs. In the later stage of the evolution process, the block-based method in the figure is used as the basic unit.

C. Initialization of the Population

Population initialization is the first stage in the evolution process, and it involves a random initialization method that obeys a uniform distribution for each individual. As described in Part A, each individual represents a CNN architecture. To find the best individual, GA with self-adaptive mutation is used in the evolution process. The proposed algorithm uses a block-based design method, which is different from traditional NAS methods that use layers as basic units. Inspired by the success of ResNet [5] and DenseNet [12], in this paper, RUs, DUs, and PUs are used as the basic units. RUs are composed of RBs, DUs are composed of DBs, and PUs are composed of pooling layers. The proposed algorithm does not optimize the fully-connected layer. The main reason is that the neurons will cause overfitting due to the characteristics of the fully-connected layers [45]. To alleviate this phenomenon, operations such as dropout [46] need to be used, but this will increase the number of parameters that need to be optimized and increase the number of computing resources.

Algorithm 1 **Framework of SaMuNet.**

Input: The population size (N), the number of maximal iteration ($nIter$), the crossover probability (λ), the mutation probability (μ), the current iteration (t).
Output: The best individual with the architecture of the CNN.

```
Randomly initialize population $P_0$;
Evaluate each individual;
while $t < nIter$ do
    $Q_t \leftarrow \{\}$;
    while $|Q_t| < N$ do
        Select two parents $p_1, p_2$ from $P_t$ by binary tournament selection;
        Generate two offspring $q_1, q_2$ from parents $p_1, p_2$ by crossover and mutation;
        $Q_t \leftarrow Q_t \cup q_1 \cup q_2$;
    end while
    Evaluate the individuals in $Q_t$ and update the probabilities of strategies in mutation strategy pool (Adding, Removing, Replacing) by using self-adaptive mechanism;
    Utilize the proposed environment selection strategy to select N individuals from $P_t \cup Q_t$ and generate the new population $P_{t+1}$;
    $t = t + 1$;
end while
Output the best individual with the best architecture of the CNN;
```

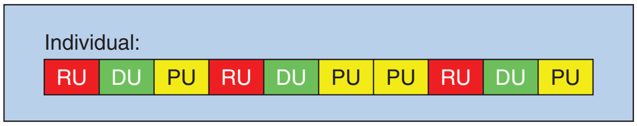

FIGURE 4 The architecture of an individual.

The method of population initialization is presented in Algorithm 2. For each individual, we need to randomly generate a positive integer k to determine the length of the individual (i.e., the number of units that need to be initialized), and then randomly generate a vector with length k. The vector stores each unit that needs to be initialized. Then, the second random number is used to determine whether to generate RUs, DUs, or PUs. Note that, in line 7, the number of PUs is limited. This is because the function of the pooling layer is to halve the size of the input, and this operation is widely used in many state-of-the-art CNNs [5], [10]–[12]. However, if an individual starts with the pooling layer, a lot of information will be lost. In addition, if a 64 × 64 input image passes through seven pooling layers continuously, the output size will be 1 × 1; this operation will lead to logic errors and is thus disallowed.

D. Evaluation of Individuals

The fitness value of each individual indicates the individual's adaptivity to the environment. In the proposed algorithm, the fitness value is the classification accuracy on the image datasets. In EC, individuals with higher fitness values have a higher probability of generating offspring with better performance than themselves. In this paper, to obtain the fitness value, the proposed algorithm decodes each individual into a CNN architecture and trains it on the training set, then tests the trained architecture on the validation set. In Algorithm 3, each individual is decoded into a CNN architecture and its weights are initialized, then the CNN will be trained until the iteration reaches its maximum value. Finally, the trained CNN model searched by the proposed algorithm is put on the validation dataset to obtain the fitness value.

Algorithm 2 **Initialize Population.**

```
Input: The population size ($N$), the maximal PUs $pool_{max}$, $j = 0$, $i = 0$.
Output: The initialized population $P_0$.
 1: $P_0 \leftarrow \{\}$;
 2: while $i < N$ do
 3:     Randomly generate a positive number $k$ to decide the length of individual;
 4:     Initialize an array with the size of $1 \times k$;
 5:     for $j < k$ do
 6:         $U \leftarrow$ Randomly choose one unit from {PU, RU, DU};
 7:         if The last unit of $U$ is not PU and the number of PU is less than $pool_{max}$ then
 8:             $U \leftarrow$ randomly choose one unit from {RU, DU};
 9:         end if
10:         Put the information of $U$ into the $j$th element of $a$;
11:     end for
12:     $P_0 \leftarrow P_0 \cup a$;
13:     $i = i + 1$;
14: end while
15: Output the initialized population $P_0$;
```

E. Crossover Operator

The offspring inherit the genes from the parents and are expected to obtain a higher fitness than the parents. Therefore, individuals with higher fitness values should be selected as parents. In this paper, the binary competition strategy is used to select parents [47]. This strategy first randomly selects two individuals from the population, and then it chooses the fitter of the two as a parent. The second parent is selected in the same way.

After selecting the parents, the crossover operator which has local ability is performed. To make each individual have better flexibility, the fixed-length coding strategy is replaced by the variable-length coding strategy. Because of the different lengths of individuals, the traditional crossover operator is no longer directly applicable. Yet, the crossover operator plays a key role in the search process; a modified single point crossover operator is used based on variable-length individuals. The specific operation is shown in Figure 5.

It should be noted that the number of input channels at the splicing place after the crossover needs to be adjusted, and the

Algorithm 3 **Evaluation of the Individuals.**

```
Input: The population $P_t$ for fitness evaluation, training dataset $D_{train}$, validation dataset $D_{valid}$, $i = 0$.
Output: The fitness of the population.
for each individual in $P_t$ do
    Decode the encoded information to the architecture of CNN;
    Initialize the weights of CNN;
    while $i < epoch$ do
        Train the CNN;
        $i{+}{+}$;
    end while
    Evaluate the accuracy for the CNN on the $D_{valid}$;
end for
Output the fitness values of the individuals in $P_t$;
```

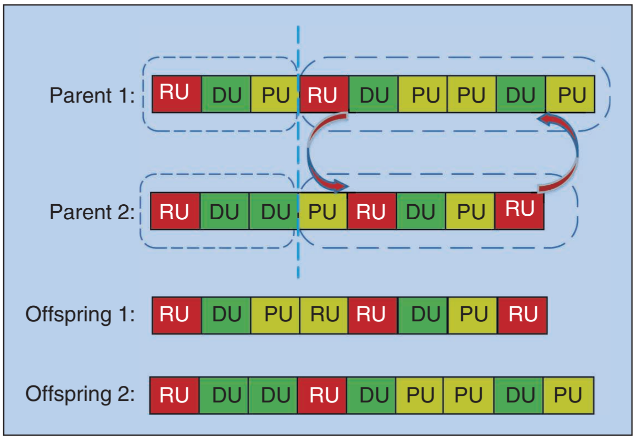

FIGURE 5 The crossover operator based on variable-length encoding strategy.

unit number also needs to be modified. The pseudo-codes of the crossover operator are described in Algorithm 4. A number is randomly generated to determine whether to perform the crossover operator or not. If it is determined to perform the crossover operator, the binary competition strategy is used to select parents firstly, then two crossover points are randomly generated for the parents. Finally, the offspring are generated by the crossover operator.

F. Self-Adaptive Mechanism Strategy Based on Blocks

The mutation operator usually explores the search space to obtain individuals with promising performance. The proposed algorithm uses three types of mutation methods, i.e., adding, removing, and replacing. Adding refers to adding a unit at the selected location of the individual, removing refers to removing the current unit at the designated location of the individual, and replacing refers to randomly replacing a new type of unit with a new one at the selected location of the individual.

The self-adaptive mechanism has been widely used in the evolutionary computation community [24], [34]. In this paper, a self-adaptive mutation strategy pool with three candidate offspring generation strategies (COGSs) is proposed. This mechanism enables the algorithm to adaptively find mutation operators that are more suitable for the current task. Thus it can determine a more suitable generation strategy to match the evolution process at different stages.

The self-adaptive mutation operator mechanism is described as follows: First, each COGS has an initial probability $1/N_s$, where N_s represents the number of COGS in the strategy pool. Let P_q represent the probability of the *qth* strategy being selected ($q = 1, 2, \ldots, N_s$). Next, the roulette strategy is used to select a COGS. Through this strategy, a new individual is generated, and it is compared with its parent. The result is recorded in the initial zero matrix $nsflag_{i,s}$ and $nfflag_{i,s}$ ($i = 1, 2, \ldots, N$, $s = 1, 2, \ldots, N_s$) where N represents the number of individuals in the population. After a generation, the sums of each column of $nsflag_{i,s}$ and $nfflag_{i,s}$ are calculated, and they are recorded in two new matrices $S_{k,q}$ and $F_{k,q}$ ($k = 1, \ldots, N_g$, $q = 1, \ldots, N_s$, where N_g represents the updated period). S records the number of successes of the *qth* strategy in the *kth* generation, and F records the number of failures of *qth* strategy in the *kth* generation. After a generation, both $nsflag_{N,Ns}$ and $nsflag_{N,Ns}$ are reset.

After N_g generations, the total number of successes and failures of all COGSs is calculated, and the probability of each COGS is updated as follows:

$$S_q^1 = \sum_{k=1}^{N_g} S_{k,q} \tag{1}$$

$$S_q^2 = \begin{cases} \lambda, & \text{if } S_q^1 = 0 \\ S_q^1, & \text{otherwise} \end{cases} \tag{2}$$

where q represents the strategy in the qth strategy pool, and λ is set to 0.0001 to prevent the probability of some strategies from becoming 0. Next, the generation probability of a new COGS is calculated using the following formulas:

$$S_q^3 = S_q^2 \Bigg/ \left(S_q^2 + \sum_{k=1}^{N_g} F_{k,q} \right) \tag{3}$$

$$P_q = S_q^3 \Bigg/ \sum_{q=1}^{N_q} S_q^3 \tag{4}$$

where S_q^3 represents the new probability of the qth COGS after the update. In addition, this probability needs to be normalized by Equation (4). The above equations are used to update the probability of each COGS according to the performance of the offspring. Obviously, the better the performance of a COGS, the higher the probability that it will be selected during the entire evolutionary process.

In Algorithm 5, the implementation of the self-adaptive mutation strategy is shown in details. First, three operations are added to the strategy pool (line 1). Then, a random number is generated to determine whether to perform mutation operations. If a mutation occurs, a block operation is selected to generate new offspring. The information of generating offspring is used to update the probability of the COGS. Finally, the mutated offspring population is output.

G. Environmental Selection Based on Semi-Complete Binary Competition

In the environmental selection stage, suitable individuals need to be selected from the population to serve as the next generation population. The simplest way to accomplish this is to select the individual with the highest fitness value from the population each time, but this will cause the population to lose diversity and fall into a local optimum [48], [49].

For NAS applications, researchers have proposed many selection strategies. Real *et al.* [50] used aging evolution to

Algorithm 4 Crossover operator of SaMuNet.

```
Input: Two parents $p_1$, $p_2$, the crossover probability $\lambda$.
Output: Two offspring $q_1$, $q_2$.
Randomly generate a number $v$ in range of [0,1];
if $v < \lambda$ then
    Select two parents $p_1$, $p_2$ from $P_t$ by binary tournament selection;
    Randomly generate two integer numbers to separate $p_1$, $p_2$ into two parts, respectively;
    Combine the first part of $p_1$ and the second part of $p_2$;
    Combine the first part of $p_2$ and the second part of $p_1$;
else
    $q_1 \leftarrow p_1$;
    $q_2 \leftarrow p_2$;
end if
Output two offspring $q_1$, $q_2$;
```

kill the oldest individual from the population, thus avoiding the premature convergence problem. Zhu *et al.* [51] combined two methods while removing the oldest and worst individuals. In addition, Xie *et al.* [28] used the roulette strategy to assign probabilities to each individual. Individuals with higher fitness values have a higher probability of survival. Sun *et al.* [14], [29] used a binary competition strategy for environmental selection.

In the traditional binary competition, two individuals are randomly selected; the individual with lower fitness value is eliminated. This means that if two good individuals are selected, the relatively bad one will be killed. This process can have a negative effect on the population. Therefore, this paper proposes an environmental selection operator based on a semi-complete binary competition. The implementation of the method is shown as follows: First, a number *k* is randomly generated, then *k* individuals with the highest fitness values are selected from the parent population and the offspring population. We retain the *k* individuals and use the binary competition strategy to select the needed individuals. In this way, it is ensured that the excellent individuals in the population will not be eliminated. This method, which incorporates the 'elite' [52] mechanism in GAs, can effectively prevent the population from degenerating during evolution.

Algorithm 6 shows the details of the proposed environmental selection based on the semi-complete binary competition. First, randomly generate a number *k* from (0, *N/2*], select *k* best individuals from the parent population and the offspring population, and then use the binary competition strategy to select *N* – *k* from the remaining individuals. Finally, the new population is obtained.

Algorithm 5 **Self-adaptive mutation operator of SaMuNet.**

```
Input: The maximal generation $N$, number of strategies $N_s = 3$, the mutation probability $\mu$, the number of offspring $q$, $i = 0$, current iteration (cIter), the number of iterations (Iter), $flagIter = 0$, $N_g = 5$.
Output: The population of mutated offspring.
Put three operation {Adding, Removing, Replacing} into strategy pool;
while $i < Iter$ do
  for $j < q$ do
    Randomly generate a number $r$ in range from [0,1];
    if $r < \mu$ then
      Choose one strategy with roulette from strategy pool to generate $q_j^{new}$;
      if the new offspring $q_j^{new}$ is better than $q_j$ then
        $nsflag_{i,s} = 1$;
      else
        $nfflag_{i,s} = 1$;
      end if
      $j = j + 1$;
    end if
  end for
  $S_{N_g \times N_s}$ = sum of each column in $nsflag_{i,s}$;
  $F_{N_g \times N_s}$ = sum of each column in $nfflag_{i,s}$;
  Reset matrices $nsflag_{i,s}$ and $nfflag_{i,s}$;
  if $cIter - flagIter == N_g$ then
    $tempS_{N_g \times N_s}$ = sum of each column in $S_{N_g \times N_s}$;
    $tempF_{N_g \times N_s}$ = sum of each column in $F_{N_g \times N_s}$;
    Update the probability: $P_{1 \times N_s} = (tempS_{N_g \times N_s}) / (temp\,S_{N_g \times N_s} + F_{N_g \times N_s})$;
    Reset matrices $S_{N_g \times N_q}$ and $F_{N_g \times N_q}$;
    $flagIter = cIter$;
  end if
  $i = i + 1$;
end while
Output the population of mutated offspring;
```

IV. Experiment Design

This section verifies the effectiveness of the CNN architecture searched by our proposed algorithm on image classification tasks. We first introduce the benchmark datasets (in Part A), then introduce the comparison algorithms (in Part B), and finally describe the parameter setting (in Part C).

A. Benchmark Datasets

CIFAR10 and CIFAR100 are color image datasets that are close to universal objects. For state-of-the-art CNNs, CIFAR10 and CIFAR100 are the most widely used benchmark datasets [27]. Therefore, this paper uses CIFAR10 and CIFAR100 to verify the effectiveness of our proposed algorithm.

CIFAR10 contains a total of 10 classes of RGB color images: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The size of each image is 32 × 32. Each class has 6,000 images. CIFAR10 has a total of 50,000 training images and

Algorithm 6 **Environmental selection based on semi-complete binary competition.**

```
Input: The parent population $P_t$, the offspring population $Q_t$, the population size $N$.
Output: The next population $P_{t+1}$.
$P_{t+1} \leftarrow \{\}, U_t \leftarrow \{\}, q \leftarrow \{\}$;
Randomly generate a number $k$ in range of $(0, N/2]$;
while $j < k$ do
  $p$ ← Select the $k$ individuals with highest values from $P_t \cup U_t$;
  $P_{t+1} \leftarrow P_{t+1} \cup P$;
end while
for $j = k + 1$ to $N$ do
  $P_1, P_2$ ← Randomly selected two individuals from $U_t$;
  $q$ ← Select the one with higher fitness from $U_t$;
  $P_{t+1} \leftarrow P_{t+1} \cup q$;
end for
Output the next population $P_{t+1}$;
```

10,000 test images. We show the examples of CIFAR10 in Figure 6. In contrast, CIFAR100 has 100 classes, each with 500 training images and 100 test images. In the experiment, in order to match the conventions of state-of-the-art CNNs, each side of each image is filled with four zeros, and then the image is randomly cropped to its original size. After that, each side is horizontally flipped randomly, and the images are input finally in the proposed algorithm.

B. Comparison Algorithms

To verify the effectiveness of the proposed algorithm, many state-of-the-art algorithms are used for comparison. According to the literature [29], the existing state-of-the-art CNNs are divided into the following three types: The first type is purely hand-designed CNNs, which require a lot of expert knowledge, such as VGG [10], ResNet [5], DenseNet [12], Network in Network [53], Highway Network [54], Maxout [55], and ALL-CNN [56]. The purpose of choosing these hand-designed networks is to prove the superiority of the automatically searched network architecture. The second type of network is designed with a semi-automatic architecture, such as Genetic CNN [28], Hierarchical Evolution [57], and EAS [58]. The third is a fully-automatically designed network architecture, such as Large-scale Evolution [26], NAS [17], MetaQNN [59], AE-CNN [29], Firefly-CNN [60], EPSO-CNN [61], and NSGANet [62] which is based on a multi-objective NAS algorithm [63].

C. Parameter Setting

We compare the existing state-of-the-art algorithms by consulting the results presented in other papers. The reason for this is that the algorithms in other papers generally use the best accuracy, which saves a lot of time. In this paper, because the proposed algorithm is based on GA architecture search [29], the population size is set to 20, the crossover probability is set to 0.9, and the three mutation operation probabilities (adding, removing, and replacing) are set to 0.4, 0.3, and 0.3, respectively. Following the recommendation of the machine learning community, the datasets are randomly divided into one-fifth as validation datasets. Finally, all classification error rates are obtained on the same validation datasets.

In this paper, when training the obtained neural network, Adam is used as the optimizer to learn the weights of the neural network [43]. The number of epochs is set to 250, the initial learning rate is set to 0.1, and the batch size is set to 80, the learning rate is adjusted according to the epoch.

In addition, this paper uses a block-based design method. Each block and its parameters in the block are set as follows: The maximum number of RUs, DUs, and PUs is set to three, and the maximum number of RBs in each RU is set to four. The selectable parameters in the DB are set to 12, 20, and 40. The maximum number of convolutional layers is set to five in the DB. Note that, because the experimental device used in this paper is one GeForce GTX 2080Ti, and the video memory is 11 GB, some of the parameters can be adjusted by the user. Because of the device's video memory limitation, out of memory error may occur if the parameter setting is too large.

V. Experimental Results and Analysis

This paper evaluates the performance of the algorithms using not only the classification accuracy, but also the number of parameters (i.e., model size) and GPU/Days. According to the concept of GPU/Days proposed by some previous researchers, the time consumption in the final search process of the algorithm can be calculated as follows: If n GTX 1080Ti cards are used to run on the CIFAR10 dataset for m days, the computational resource consumed by the algorithm on CIFAR10 is $m * n$ GPU/Days. In addition, the proposed algorithm uses the standard in [29] in which Sun *et al.* divided the current state-of-the-art CNNs into three categories: hand-crafted, semi-automatic, and completely automatic.

A. Classification Performance of the Algorithms on CIFAR10

Table II shows the performance comparison of the proposed algorithm SaMuNet and the comparison algorithms on CIFAR10 and CIFAR100. The second and third columns represent the validation classification error rates of different CNNs on CIFAR10 and CIFAR100, respectively. The fourth column represents the sizes of the final models. The fifth column represents the sizes of GPU/Days consumed by each algorithm, and the sixth column is the constructive method of each model. The '-' in the table indicates that this indicator has not been reported in the relevant reference.

As shown in Table II, compared with the purely hand-crafted model, which includes ResNet and DenseNet, the classification accuracy of SaMuNet on CIFAR10 has achieved the second lowest error rate, which is a little higher than Firefly-CNN. However, the number of parameters from the final model which is found by our algorithm is much lower than Firefly-CNN. The proposed algorithm has an error rate 1.6%

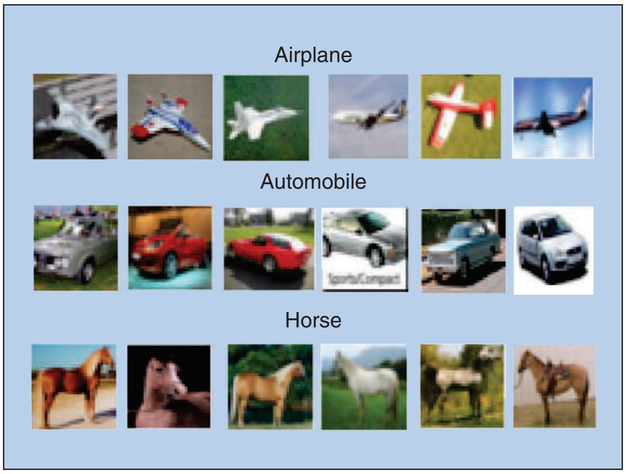

FIGURE 6 Three examples from CIFAR10.

lower than the best model (DenseNet, which is hand-crafted on CIFAR10). Since the hand-crafted model does not have the process of searching the model, its GPU/Days is represented by '-'. Compared with the semi-automatic architecture design methods, our proposed algorithm has the lowest classification error rate on CIFAR10, and consumes less GPU/Days than Hierarchical Evolution. Compared with the completely automatic search algorithm, the proposed algorithm has a classification 1.8% lower than Large-scale Evolution, 2.4% lower than NAS, 3.2% lower than MetaQNN, and 0.7% lower than AE-CNN. Since our proposed algorithm and AE-CNN are both block-based constructive methods and both involve GAs, this also proves the effectiveness of our proposed adaptive mutation strategy.

It can be seen that our algorithm is less time-consuming than most algorithms, except for EPSOCNN. Because EPSOCNN uses a multi-server distributed computing method, it is faster than our algorithm which did not use a weight sharing strategy.

B. The Classification Performance of the Algorithms on CIFAR100

As one of the most widely used datasets for machine learning algorithms, the classification task on CIFAR100 is more difficult than that on CIFAR10. It can be seen from Table II that all models have higher classification errors on CIFAR100. The classification error rate of our proposed algorithm on CIFAR100 is 20.2%.

Compared with the hand-crafted networks, the error rate of our proposed algorithm is 4.2% lower than DenseNet which is the best of the hand-crafted models. Compared to other hand-crafted models, SaMuNet has remarkable advantages; in particular, when compared to Maxout, the error rate of SaMuNet is 18.6% lower. In addition, the table shows that the automatically searched architectures generally have a lower classification error rate than the manually designed networks. In addition, expert knowledge is not required when designing architectures for SaMuNet.

Compared with the semi-automatic and completely automatic architecture design methods, SaMuNet also achieved the lowest error rate. Among them, the classification error rate of our proposed algorithm is 0.6% lower than that of AE-CNN. This also proves the effectiveness of the adaptive mutation strategy proposed in this paper. Compared with the Large-Scale Evolution and NAS algorithms, SaMuNet is dozens of times faster. Because NAS is based on reinforcement learning, it usually consumes more computing resources. In general, SaMuNet has achieved very promising performance in classification accuracy, and the consumption of computing resources is relatively lower than its comparison algorithms.

TABLE II THE PERFORMANCE OF ALGORITHMS ON CIFAR10 AND CIFAR100.

DATASETS	CIFAR10(%)	CIFAR100(%)	PARAMETERS	GPU/DAYS	NETWORK CONSTRUCTIONS
VGG [10]	6.66	28.05	20.04M	–	Hand-crafted
ResNet [5] (depth=1,202)	7.93	27.82	1.7M	–	Hand-crafted
DenseNet (K=12) [12]	5.24	24.42	1.0M	–	Hand-crafted
Maxout [55]	9.3	38.6	–	–	Hand-crafted
Network in Network [53]	8.81	35.68	–	–	Hand-crafted
Highway Network [54]	7.72	32.39	–	–	Hand-crafted
ALL-CNN [56]	7.25	33.71	–	–	Hand-crafted
Genetic CNN [28]	7.1	29.05	–	17	Semi-automatic search
Hierarchical [57] Evolution	3.63	–	–	300	Semi-automatic search
EAS [58]	4.23	–	23.4M	10	Semi-automatic search
Large-Scale Evolution [26]	5.4	–	5.4M	2,750	Automatic search
Large-Scale Evolution	–	23	40.4M	2,750	Automatic search
NAS [17]	6.01	–	2.5M	22,400	Automatic search
MetaQNN [59]	6.92	27.14	–	100	Automatic search
AE-CNN [29]	4.3	–	2.0M	27	Automatic search
AE-CNN	–	20.85	5.4M	36	Automatic search
NSGANet [62]	4.67	–	0.2M	27	Automatic search
NSGANet	–	25.17	0.2M	27	Automatic search
Firefly-CNN [60]	3.3	22.3	3.21M	–	Automatic search
EPSO-CNN [61]	3.69	–	6.77M	4–	Automatic search
SaMuNet (**ours**)	3.6	–	1.5M	23	Automatic search
SaMuNet (**ours**)	–	20.2	4.6M	25	Automatic search

C. Evolution Process on CIFAR10

To prove the effectiveness of the self-adaptive mutation mechanism and semi-complete binary competition strategy proposed in this paper, the classification accuracy of the final searched individuals is collected, and plot the statistical results. In this paper, the population size is 20, and the total number of generations is 20.

> **The simplest way to choose parents is to select the individual with the highest fitness value from the population each time, but this will cause the population to lose diversity and fall into a local optimum.**

Figure 7 compares the proposed algorithm and GA. To intuitively show how the self-adaptive mutation mechanism guides the evolution process, a tendency line is drawn by connecting the averages over ten points before and after the current point. It is noted that the accuracy of some individuals is 0% in Figure 7. Because the variable-length encoding strategy is used, the length that the offspring individuals generated by the crossover individuals is too long. Due to the memory limitation of our experimental equipment, an out of memory occurs. As shown in Figure 7, for SaMuNet, from the 1st generation to the 7th generation, the average classification accuracy rises faster. From the 12th generation, the average classification accuracy is stable at about 90%, and the gap between individuals narrows, which shows that the algorithm is gradually converging. SaMuNet can achieve about 96% accuracy in the last generation. For the comparison method GA, due to the lack of guidance from the self-adaptive mechanism, the individual accuracy rises slowly. After the 12th generation, the traditional GA gradually converges too, but the final convergence accuracy is only about 94%. The analysis proves that the self-adaptive mechanism has better performance in the later stage, and it can also guide the later evolution.

In addition, to prove the effectiveness of the semi-complete binary competition strategy, this paper makes a comparison between SaMuNet with semi-complete binary competition and SaMuNet with the binary competition algorithm. The conditions remain unchanged from the previous experiment. It can be seen from Figure 8 that, compared with the binary competition, the semi-complete binary competition strategy appears to be more stable throughout the entire evolution process. The competition method tends to retain bad individuals. Especially after the 15th generation, there are some poor individuals with an accuracy of 80%.

The self-adaptive mutation mechanism proposed in this paper can effectively improve the search efficiency of the algorithm, and guide the later evolution through the experience accumulated in the early stage. In addition, the proposed semi-complete binary competition strategy can effectively maintain population stability and prevent population degradation.

VI. Conclusions and Future Work

The goal of this paper is to design a CNN architecture search algorithm by using SaMuNet with an adaptive mutation strategy and a semi-complete binary competition strategy. This goal is successfully achieved on CIFAR10 and CIFAR100. This paper uses a block-based design method to accelerate the entire CNN architecture design process, and uses an adaptive mutation strategy to make the algorithm adaptively select mutation strategies in different evolution stages, so that the algorithm can guide the search process effectively. In addition, a semi-complete binary competition strategy is also designed for environmental selection. This strategy can better retain the elites. It can be seen from the experimental results that our proposed algorithm achieves better results in comparison with different hand-crafted, semi-automatic, and completely automatic structural design methods in terms of accuracy and consumption of computing resources. SaMuNet still consumes a lot of computing resources, thus, in the future, our team will be committed to accelerating the process of evaluating individuals, and reducing consumption of resources.

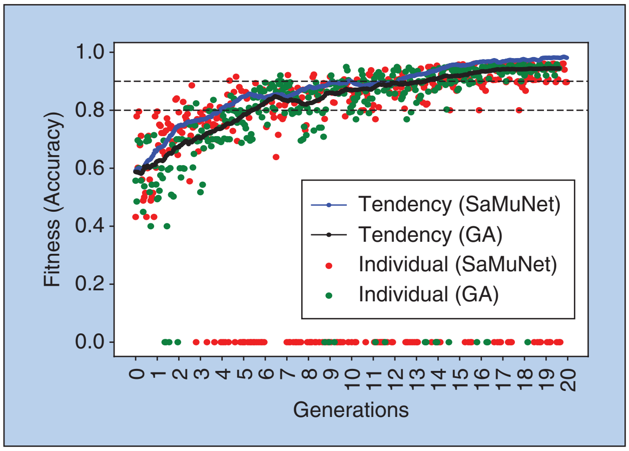

FIGURE 7 Scatter diagram of SaMuNet and GA.

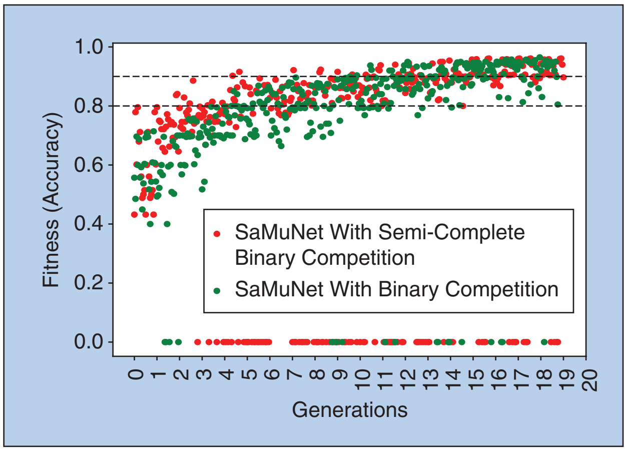

FIGURE 8 Scatter diagram of semi-completely binary competition.

VII. Acknowledgments

This work was partially supported by the National Natural Science Foundation of China (61876089, 61876185, 61902281), the Engineering Research Center of Digital Forensics, Ministry of Education, the Priority Academic Program Development of Jiangsu Higher Education Institutions, and the Postgraduate Research & Practice Innovation Program of Jiangsu Province (KYCX21_1019).

References

[1] I. Al Ridhawi, S. Otoum, M. Aloqaily, and A. Boukerche, "Generalizing AI: Challenges and opportunities for plug and play AI solutions," *IEEE Netw.*, 2020. doi: 10.1109/MNET.011.2000371.

[2] G. Hinton et al., "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, 2012.

[3] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vision Pattern Recogn.*, 2015, pp. 3431–3440.

[4] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, 2017. doi: 10.1109/TPAMI.2017.2699184.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vision Pattern Recogn.*, 2016, pp. 770–778.

[6] X. Yao and Md M. Islam, "Evolving artificial neural network ensembles," *IEEE Comput. Intelli. Mag.*, vol. 3, no. 1, pp. 31–42, 2008. doi: 10.1109/MCI.2007.913386.

[7] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning. in *Proc. 25th Int. Conf. Machine Learning*, 2008, pp. 160–167.

[8] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Proc. IEEE Conf. Comput. Vision Pattern Recogn.*, 2015*, pp.* 3156–3164.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Information Process. Syst.*, 2012, pp. 1097–1105.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, arXiv:1409.1556.

[11] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vision and Pattern Recogn.*, 2015, pp. 1–9.

[12] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vision and Pattern Recogn.*, 2017, pp. 4700–4708.

[13] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proc. Genetic Evolutionary Comput. Conf.*, 2017, pp. 497–504. doi: 10.1145/3071178.3071229.

[14] Y. Sun, B. Xue, M. Zhang, and G. G Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Trans. Evolutionary Comput.*, vol. 24, no. 2, pp. 394–407, 2019. doi: 10.1109/TEVC.2019.2916183.

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. doi: 10.1038/323533a0.

[16] Y. Liu, Y. Sun, B. Xue, M. Zhang, and G. Yen, "A survey on evolutionary neural architecture search," 2020, arXiv:2008.10937.

[17] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, arXiv:1611.01578.

[18] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, 1996.

[19] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," 2018, arXiv:1806.09055

[20] T. Bäck, D. B. Fogel, and Z. Michalewicz, "Handbook of evolutionary computation," *Release*, vol. 97, no. 1, p. B1, 1997.

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

[22] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. ICNN'95—Int. Conf. Neural Netw.*, vol. 4, pp. 1942–1948, 1995.

[23] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, 2006. doi: 10.1109/MCI.2006.329691.

[24] Y. Xue, B. Xue, and M. Zhang, "Self-adaptive particle swarm optimization for large-scale feature selection in classification," *ACM Trans. Knowl. Discovery Data (TKDD)*, vol. 13, no. 5, pp. 1–27, 2019. doi: 10.1145/3340848.

[25] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: From architectures to learning," *Evolution. Intell.*, vol. 1, no. 1, pp. 47–62, 2008.

[26] E. Real et al., "Large-scale evolution of image classifiers," 2017, arXiv:1703.01041.

[27] A. Krizhevsky et al., "Learning multiple layers of features from tiny images," 2009.

[28] L. Xie and A. Yuille, "Genetic CNN," in *Proc. IEEE Int. Conf. Comput. Vision*, 2017, pp. 1379–1388.

[29] Y. Sun, B. Xue, M. Zhang, and G. G Yen, "Completely automated CNN architecture design based on blocks," *IEEE Trans. Neural Netw. Learning Syst.*, vol. 31, no. 4, pp. 1242–1254, 2019. doi: 10.1109/TNNLS.2019.2919608.

[30] L. Frachon, W. Pang, and G. M Coghill, "Immunecs: Neural committee search by an artificial immune system," 2019, arXiv:1911.07729.

[31] Z. Chen, Y. Zhou, and Z. Huang, "Auto-creation of effective neural network architecture by evolutionary algorithm and ResNet for image classification," in *Proc. IEEE Int. Conf. Systems, Man Cybernetics (SMC)*, 2019, pp. 3895–3900. doi: 10.1109/SMC.2019.8914267.

[32] D. Song, C. Xu, X. Jia, Y. Chen, C. Xu, and Y. Wang, "Efficient residual dense block search for image super-resolution," in *Assoc. the Adv. Artif. Intell.*, 2020, pp. 12,007–12,014. doi: 10.1609/aaai.v34i07.6877.

[33] P. R. Lorenzo and J. Nalepa, "Memetic evolution of deep neural networks," in *Proc. Genetic Evolutionary Comput. Conf.*, 2018, pp. 505–512.

[34] Y. Xue, J. Jiang, B. Zhao, and T. Ma, "A self-adaptive artificial bee colony algorithm based on global best for global optimization," *Soft Comput.*, vol. 22, no. 9, pp. 2935–2952, 2018. doi: 10.1007/s00500-017-2547-1.

[35] M. Suganuma, M. Kobayashi, S. Shirakawa, and T. Nagao, "Evolution of deep convolutional neural networks using Cartesian genetic programming," *Evolut. Comput.*, vol. 28, no. 1, pp. 141–163, 2020. doi: 10.1162/evco_a_00253.

[36] K. Chen and W. Pang, "Immunetnas: An immune-network approach for searching convolutional neural network architectures," 2020, arXiv:2002.12704.

[37] A. Emin Orhan and X. Pitkow, "Skip connections eliminate singularities," 2017, arXiv:1701.09175.

[38] T. Elsken, J.-H. Metzen, and F. Hutter, "Simple and efficient architecture search for convolutional neural networks," 2017, arXiv:1711.04528.

[39] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, 1994. doi: 10.1109/72.279181.

[40] A. Kai Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Trans. Evolut. Comput.*, vol. 13, no. 2, pp. 398–417, 2008. doi: 10.1109/TEVC.2008.927706.

[41] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," 2012, arXiv:1212.5701.

[42] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Machine Learning Res.*, vol. 12, no. 7, pp. 2121–2159, 2011.

[43] D. P. Kingma and J. Ba, "ADAM: A method for stochastic optimization," 2014, arXiv:1412.6980.

[44] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nat. Mach. Intell.*, vol. 1, no. 1, pp. 24–35, 2019. doi: 10.1038/s42256-018-0006-z.

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

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

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

[48] 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, 1997. doi: 10.1109/72.623217.

[49] L. Davis, "Handbook of genetic algorithms," 1991.

[50] E. Real, A. Aggarwal, Y. Huang, and Q. V Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, pp. 4780–4789, 2019. doi: 10.1609/aaai.v33i01.33014780.

[51] H. Zhu, Z. An, C. Yang, K. Xu, E. Zhao, and Y. Xu, "EENA: Efficient evolution of neural architecture," in *Proc. IEEE Int. Conf. Comput. Vision Workshops*, 2019, pp. 1–13.

[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, 2001. doi: 10.1109/20.952626.

[53] M. Lin, Q. Chen, and S. Yan, "Network in network,", 2013, arXiv:1312.4400.

[54] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway networks," 2015, arXiv:1505.00387.

[55] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," in *Proc. Int. Conf. Machine Learning*, 2013, pp. 1319–1327.

[56] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," 2014, arXiv:1412.6806.

[57] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," 2017. arXiv:1711.00436.

[58] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," 2017, arXiv:1707.04873.

[59] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," 2016, arXiv:1611.02167.

[60] A. I. Sharaf and E.-S. F. Radwan, "An automated approach for developing a convolutional neural network using a modified firefly algorithm for image classification," in *Applications of Firefly Algorithm and Its Variants*. Springer-Verlag, 2020, pp. 99–118.

[61] B. Wang, B. Xue, and M. Zhang, "Particle swarm optimisation for evolving deep neural networks for image classification by evolving and stacking transferable blocks," in *Proc. IEEE Congr. Evolut. Comput. (CEC)*, 2020, pp. 1–8. doi: 10.1109/CEC48606.2020.9185541.

[62] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "NSGA-NET: Neural architecture search using multi-objective genetic algorithm," in *Proc. Genetic Evolut. Comput. Conf.*, 2019, pp. 419–427.

[63] Y. Xue, Y. Tang, X. Xu, J. Liang, and F. Neri, "Multi-objective feature selection with missing data in classification," *IEEE Trans. Emerg. Topics Comput. Intell.* doi: 10.1109/TETCI.2021.3074147.