Automatic Design of Convolutional Neural Network Architectures Under Resource Constraints

Siyi Li, *Graduate Student Member, IEEE*, Yanan Sun, *Member, IEEE*, Gary G. Yen, *Fellow, IEEE*, and Mengjie Zhang, *Fellow, IEEE*

Abstract*— With the rise of various smart electronics and mobile/edge devices, many existing high-accuracy convolutional neural network (CNN) models are difficult to be applied in practice due to the limited resources, such as memory capacity, power consumption, and spectral efficiency. In order to meet these constraints, researchers have carefully designed some lightweight networks. Meanwhile, to reduce the reliance on manual design on expert experience, some researchers also work to improve neural architecture search (NAS) algorithms to automatically design small networks, exploiting the multiobjective approaches that consider both accuracy and other important goals during optimization. However, simply searching for smaller network models is not consistent with the current research belief of "the deeper the better" and may affect the effectiveness of the model and thus waste the limited resources available. In this article, we propose an automatic method for designing CNNs architectures under constraint handling, which can search for optimal network models meeting the preset constraint. Specifically, an adaptive penalty algorithm is used for fitness evaluation, and a selective repair operation is developed for infeasible individuals to search for feasible CNN architectures. As a case study, we set the complexity (the number of parameters) as a resource constraint and perform multiple experiments on CIFAR-10 and CIFAR-100, to demonstrate the effectiveness of the proposed method. In addition, the proposed algorithm is compared with a state-of-the-art algorithm, NSGA-Net, and several manual-designed models. The experimental results show that the proposed algorithm can successfully solve the problem of the uncertain size of the optimal CNN model under the random search strategy, and the automatically designed CNN model can satisfy the predefined resource constraint while achieving better accuracy.

***Index Terms*— Constraint handling, convolutional neural networks (CNNs), evolutionary deep learning, genetic algorithms (GAs), neural architecture search (NAS).**

Manuscript received July 24, 2021; revised October 4, 2021; accepted October 23, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61803277 and in part by the Science and Technology Innovation Talent Project of Sichuan Province under Grant 2021JDRC0001. *(Corresponding author: Yanan Sun.)*

Siyi Li and Yanan Sun are with the College of Computer Science, Sichuan University, Chengdu 610065, China (e-mail: lsy0868@outlook.com; ysun@scu.edu.cn).

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

Mengjie Zhang is with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand (e-mail: mengjie.zhang@ecs.vuw.ac.nz).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2021.3123105.

Digital Object Identifier 10.1109/TNNLS.2021.3123105

I. Introduction

Convolutional neural networks (CNNs) have achieved great success in many challenging real-world application domains, such as natural language processing [1], image recognition [2], and object detection [3]. Besides the notable achievements obtained by CNNs, it can be observed that the size of the CNN model also becomes larger and larger as the model performance gradually improves. There is a common belief that deeper CNN models typically perform better than shallower ones [4], [5]. As a result, an enormous number of researchers have devoted to developing deeper CNNs to achieve exceptional accuracy [6]–[8]. In principle, the superiority of CNNs comes at the cost of higher computational complexity, so they are better suited to be performed on resource-intensive devices. However, in recent years, with the emergence of various types of resource-limit devices, such as mobile phones, unmanned aerial vehicles, and the Internet of Things/smart sensors, the deployment of high-performance CNN models inevitably faces prohibitive resource constraints, such as power consumption and memory capacity. For these devices, the "most critical" concern is on the computing resources that the model will consume rather than the best promising performance the model will achieve. Consequently, many deep CNN models of greater performance cannot be directly applied to these devices due to their high cost. Therefore, there has been a rising interest in building high-performance and low-complexity CNN models that can accommodate various computational resources onboard the devices [9].

There are currently two main ways of designing such kinds of models: manually designed with rich expertise and automatically designed via neural architecture search (NAS) technology. Specifically, researchers have focused on designing and improving "lightweight CNN" models by utilizing less expensive operations, such as depthwise convolution [2] and group convolution [10]. At present, some excellent manually designed network models with both high-performance and lightweight have been proposed. For example, SqueezeNet [9] extensively used the squeeze module to reduce the number of parameters. The squeeze module uses 1×1 convolution to reduce the number of parameters and limits the number of input channels of 3×3 convolution. Obviously, the number of parameters for 1×1 convolution is $1/9$ of those for the 3×3 one, and reducing the number of input channels will significantly reduce the number of the corresponding convolution kernels and further reduce the number of parameters. ShuffleNet [10] successfully achieved excellent accuracy with much fewer parameters by utilizing group convolution. Compared with standard convolution, group convolution divides the input feature map channels into G groups, each group is convolved separately, and the output is finally concatenated. The numbers of input and output feature maps remain unchanged, but the number of convolution kernels becomes $1/G$, and the total amount of parameters reduces down to $1/G$. In 2017, MobileNet [2] was presented for mobile and embedded vision applications by using depthwise separable convolutions. The depthwise separable convolution mainly reduces the number of parameters by convolving the input feature maps with the depthwise convolution. The depthwise convolution can be considered as a special group convolution, where each feature map channel is a group. The number of parameters, in this case, is reduced to $1/N$, given that the number of input feature map channels is N. Then, the depthwise separable convolution became one of the main architecture of lightweight CNNs.

On the other hand, NAS algorithms automatically design high-performance and lightweight CNN models mainly via the idea of multiobjective optimization algorithms [11]. The multiobjective evolutionary algorithms (EAs) are capable of simultaneously optimizing a set of competing objectives, e.g., accuracy and other important objectives [e.g., parameters number and floating-point operations (FLOPs)] that affect the complexity of the model [12], and when they are multiple candidates, the better one is decided based on the more important objective. In this way, the "optimal" solution can be obtained after several comparisons. For example, MONAS [13] and MnasNet [14] used reinforcement learning (RL) methods to search for a Pareto optimal solution that strikes a balance between accuracy and computation cost with the idea of multiobjective optimization. Specifically, they added a model computation resource prediction module to the regular search framework of the recurrent neural network (RNN) controllers and the trainers for evaluating accuracy. During the process, the model resource consumption, such as latency, parameter number, model size, and accuracy, is utilized to jointly constitute the reward to perform two-objective optimization of the controller to finally obtain a model that achieves a balance between accuracy and computation resource. ProxylessNAS [15] and wiNAS [16] incorporated the expected latency of the network into the normal loss function, to help the gradient-based approach handling the latency. Unfortunately, unlike accuracy that can be optimized using the gradient of the loss function, latency is often nondifferentiable. Therefore, the latency prediction function of the network needs to be redesigned as a continuous function, which is then incorporated into the normal loss function to optimize both network accuracy and latency together. Specifically, they added the expected latency obtained from the weighted combination of the latency estimate of each candidate operation and its respective probability of being sampled to the loss function and then optimizes both accuracy and latency simultaneously by the gradient descent approach to achieve multiobjective optimization. NSGA-Net [17] applied the nondominated sorting genetic algorithm (NSGA-II) [18] to optimize both accuracy and complexity. NSGA-Net takes complexity reduction and accuracy improvement as two goals. It continuously searches for a set of CNN architectures that approximate the Pareto front between performance and complexity on an image classification task.

Despite the overwhelming success, these approaches experience limitations in practice. Specifically, the manual-designed approaches propose excellent models with specific network sizes. However, different platforms have different resource constraints. As a result, there are still situations where the model sizes do not satisfy the resource constraint. It is often necessary to pruning or redesigning the network models. However, such operations rely highly on manual expertise, and only experts with extensive experience in both the target application and the CNN domain can roughly design a suitable network architecture that can bring satisfactory performance while meeting resource constraints via an expensive trial-and-error. This has clearly made the manual design more difficult in practice.

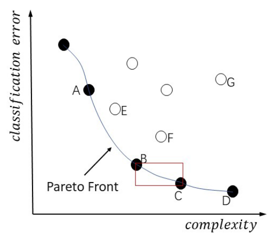

Fig. 1. Tradeoff frontier after the multiobjective NAS optimization search.

Although the NAS-based algorithms for lightweight models can address the expertise requirement caused by the manual-designed approaches, they mostly achieve the goal by formulating the architecture design as a multiobjective optimization problem as mentioned above. Generally, the solution of a multiple-objective optimization problem results in a Pareto front that is composed of a set of well-distributed solutions with good convergence (i.e., the nondominated solutions [19]). After obtaining the Pareto front by the proper multiobjective optimization algorithms [20], the solutions that satisfy the resource constraint from the nondominated solutions are picked up for use. These automated design methods face a similar level of difficulty of not being able to find models that satisfy specific constraints. Because the Pareto front obtained is actually composed of discrete solutions, it is not uncommon that no ideal model can be found in the vicinity of a specific constraint with the chosen multiobjective NAS methods. An illustrative example can be seen from Fig. 1, where the curve denotes a Pareto front obtained by the corresponding multiobjective NAS method, and the horizontal axis denotes complexity, while the vertical axis denotes the classification error. In particular, the Pareto front is composed of five nondominated solutions represented by A, B, C, D, and E. If the resource constraint is set in the red bounding box, only the nearby model B or C can be selected. However, the Pareto optimal architecture C with higher complexity exceeds the resource constraint, which is considered infeasible. Also, architecture B with much lower complexity does not achieve the optimal performance under the resource constraint. This is because the performance of the optimal solution keeps improving as the complexity increases.

We find that current approaches are pursuing smaller model complexity to satisfy different resource constraints, but this idea is still experiencing some problems in practice. In order to solve the above problems, in this article, we propose a CNN architecture design method under constraint handling, named CH-CNN, with the aim that CNN architectures can be automatically learned through the NAS method and the chosen CNN architecture can satisfactorily meet the specific constraints, including model complexity. Specifically, the proposed algorithm is developed based on genetic algorithms (GAs), which is a mainstream technique in designing effective

NAS algorithms. Also, two types of constraint handling methods, the adaptive penalty method and adaptive repair method, are developed to help find the feasible CNN models during the search process. In summary, the contributions of the proposed algorithm are shown as follows.

1) We develop two constraint handling methods to limit the complexity of CNNs and help search for optimal CNN models satisfying the constraints. Specifically, we develop the adaptive penalty algorithm and adaptive repair method to help the proposed algorithm search for individual architectures satisfying the constraints quickly from two perspectives: reducing the possibility of infeasible solutions being selected and repairing infeasible solutions to become feasible.
2) We design new building blocks concerning the manual-designed lightweight models to search for promising small CNNs. Currently, many architecture units of NAS are basic components or classical blocks, such as DenseNet blocks and Resnet blocks. The new building blocks designed according to the lightweight CNN models require fewer parameters than the traditional blocks and can effectively reduce the size of the final CNN models. Meanwhile, the corresponding building blocks can be interchanged in the repair method to achieve purposeful parameter adjustment.
3) We present a variable-length encoding strategy to enable automatic search for the suitable depth without manual intervention during the search. The depth for an optimal CNN architecture is mostly unknown in advance. As a result, the specified number may be incorrectly estimated if the encoding strategy is fixed and requires manual intervention during the architecture search. The presented variable-length encoding strategy can improve the diversity of CNN architectures during the search and is conducive to find the CNN models with the best performance during the evolution process.

The rest of this article is organized as follows. The background is introduced in Section II. Section III details the framework and main steps of the proposed algorithm. Sections IV and V provide the experimental design and comparison results, respectively. In Section VI, the conclusion and future works are elaborated.

II. Background

To help readers better understand the proposed algorithm, the background of NAS and the common constraint handling approaches for optimization problems are briefly introduced in this section.

A. Neural Architecture Search

NAS [21] is a methodology that was initially developed to design high-performance deep network architectures without relying too much on the human experience. NAS is often formulated as a challenging optimization problem, such as computationally expensive and bilevel, and then, a proper optimization algorithm is adopted for efficient and effective solving. Based on the adopted optimization algorithms, existing NAS methods can be generally classified into RL-based [22] NAS algorithms [23], gradient-based NAS algorithms [24], and EA-based [25] NAS algorithms [26], [27].

The earliest search method used in NAS is the RL algorithm. NAS-RL [28] proposes to use the RL mechanism to train an RNN controller to automatically generate a neural network without manual design, and the designed network has achieved good results on relevant datasets. However, RL-based algorithms often require a large number of computing resources. For example, the NAS-RL consumed 800 graphic processing units (GPUs) in 28 days to find the promising CNN architecture on the CIFAR10 dataset [29]. It is simply not possible to complete the training of the RNN controller without adequate hardware resources, which is very unfriendly to smaller organizations or businesses.

In recent years, gradient-based algorithms have been proposed by researchers to improve search efficiency. The classical gradient-based algorithm DARTS [24] achieves an efficient search for CNN models in continuous space using gradient descent to find an optimal network architecture and network weights. However, the authors state that they are no assurance for the convergence guarantees for their optimization algorithm, and it is necessary to choose a suitable learning rate to reach an optimal solution in practice. In addition, the gradient-based algorithms require a prior construction of a supernet, which dictates a high level of expertise.

EAs are another major search strategy for NAS, which accounts for the majority of existing NAS algorithms. Due to the promising characteristics of EAs in insensitiveness to the local minimal and no requirement to gradient information, the EA-based NAS algorithms can often find an excellent network model. In fact, EAs have been widely used to search for neural networks since 40 years ago, which can be briefly divided into three different stages. For the first two stages that are termed "evolving neural networks" [30] and "neuroevolution" [31], respectively, EAs were used for optimizing both the architectures and the weights of artificial neural networks (ANNs). The major differences between both are that evolving neural networks concerned small-scale neural networks, while neuroevolution focused on median-scale neural networks. Moreover, the most representative work of neuroevolution is the NeuroEvolution of Augmenting Topologies (NEAT) [32] algorithm and its variants [33]. The third stage mainly refers to exploiting EA for optimizing only the architectures of CNNs, while the weights are still optimized through gradient-based algorithms. The algorithms in this stage are also termed EA-based NAS [34], and the LEIC method [35] proposed by Google was commonly viewed as the seminal work. After that, a large number of EA-based NAS algorithms have been proposed for deep neural networks. For example, in AmoebaNet [36], the evolved model surpasses hand designs in performance for the first time. In addition, EvoCNN [37] designed an efficient variable-length gene encoding strategy and the corresponding genetic operators, which can automatically search for the CNN architectures with potentially optimal depth. In the AE-CNN [38], the efficient ResNet and DenseNet blocks were used for speeding up the architecture design, and it outperforms the state-of-the-art CNNs. After that, the EA-based NAS approaches continue to make breakthroughs, especially for computer vision tasks, and more details can be found from a recent survey paper [39].

In summary, the EA-based NAS tends to save more computational resources than the ones based on the RL and does not require the predesign supernet and can search for the complete CNN architecture through evolution [37] compared to the gradient-based approaches. More importantly, EAs can solve the constrained optimization problems [40]. With this feature of EAs, EA-based NAS can effectively solve the target problems by searching for promising models that satisfy specific constraints.

B. Constrained Optimization Methods

Constrained optimization methods [41] were proposed to solve constrained problems that exist in practical applications. In engineering design, intelligent control, and many other optimization fields, conditional restrictions (constraints) are often encountered, which poses great challenges to find the feasible optimal solution to the problem. This type of optimization problem is called a constrained optimization problem. A constrained optimization problem can be reduced to the problem of deciding a set of decision variables that make the objective function achieve an optimal value while satisfying constraints at the same time. Generally, a constrained optimization problem can be defined in the following equations:

$$\min\ f(x),\quad x \in R^m \tag{1}$$

$$\text{s.t. } g_i(x) < 0,\quad i = 1, 2, \ldots, q \tag{2}$$

$$h_j(x) = 0,\quad j = 1, 2, \ldots, p \tag{3}$$

where $f(x)$ is the objective function, x is the vector of solutions $x = [\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m}]^{\mathrm{T}}$, R is the search space, $g(x)$ and $h(x)$ are the equality function and inequality function, respectively, q is the number of inequality constraints, and p is the number of equality constraints.

EAs, especially GAs, are proven the efficient optimization techniques that can be used with constraint handling technology to handle constrained optimization problems [42]. Constraint handling methods have a significant impact on the performance of constrained optimization algorithms, which can transform a constrained optimization problem into an unconstrained problem to obtain feasible solutions. In this way, the performance of the GAs as the search mechanism can be fully utilized. In order to improve the efficiency of GAs for solving constrained optimization problems, in recent years, researchers have proposed a large number of constrained optimization methods that integrate constraint handling methods into GA frameworks. In the following, we will briefly introduce three popular constraint handling strategies that are the base works of the proposed algorithm.

1) Multiobjective Optimization Technique: In this strategy, the objective and constraints are treated separately, i.e., x is continuously adjusted to simultaneously achieve optimization of $f(x)$ and satisfy $g(x)$ and $h(x)$ [43]. The main idea is to transform the constraints into a new objective function of the optimization problem, which in turn converts the original constrained single-objective optimization problem into an unconstrained two-objective optimization problem [41] to be solved as normal. Specifically, the original single objective function $f(x)$ is kept and the objective $\mathrm{CV}(x)$, i.e., the constraint violation value, is added. For solution x, the constraint violation value can be expressed as

$$\mathrm{CV}(x) = \sum_{j=1}^{p} |h_j(x)| + \sum_{i=1}^{n} \max(0, g_i(x)). \tag{4}$$

Obviously, for solution x, the smaller the $\mathrm{CV}(x)$ value, the better the solution satisfies the overall constraints. All constraint functions can be transformed into the inequality constraints such as (2) or equality constraints such as (3). For equality constraint $h(x)$, it is possible to violate the constraint in two directions, i.e., larger than the constraint value or smaller than the constraint value. In practice, the equality constraint $h(x)$ is often transformed into the inequalty constraint $|h(x)| - \epsilon \leq 0$ for the solving, where ϵ is a tolerance value. For the inequality constraint $g(x)$, there is only one unsatisfied case where the value of the constraint function is greater than zero. When all constraints are met, the constraint violation $\mathrm{CV}(x)$ is zero.

Multiobjective optimization technique compares the objectives, i.e., $f(x)$ and $\mathrm{CV}(x)$, separately, and then select the Pareto optimal solutions. It can solve the constrained optimization problem without additional hyperparameters. However, the comparison process requires more computation. In addition, the greater focus on feasible solutions in the comparison process can make the method experience difficulties in solving constrained optimization problems where the optimal solution lies on the boundary of the feasible region. This is because rejecting some promising infeasible solutions, which may be closer to the global optimal solution during the optimization process, will impair the search capability that eventually affects the ability to find the optimal feasible solutions.

2) Penalty Functions: The penalty function is almost the most common strategy used by the EA community to handle constraints [44]. The idea of this strategy is to transform a constrained optimization problem into an unconstrained one by penalizing a certain value to the objective function $f(x)$ based on the total constraint violation value $\mathrm{CV}(x)$ and penalty factor. Then, the comparison is done according to the new obtained objective function $f'(x)$. The penalty method can effectively retain the infeasible solution that has less constraint violation and significantly better performance than using the feasible solutions alone. Such infeasible solutions are more helpful to search for the global optimal solution. However, the setting of the penalty factor is still the core issue, which will largely affect the subsequent selections.

Depending on the setting of the penalty factor, the penalty functions can be classified into static penalties, dynamic penalties, death penalty, and adaptive penalty [42], [43]. The death penalty sets the penalty factor to infinity. When a certain solution violates any constraint, its $f'(x)$ will be penalized by the infinite amount of penalty to make it the worst individual. In static penalties, the penalty factors remain constant during the entire evolutionary process. As for dynamic penalties, the penalty factors are adjusted during the evolutionary process. Also, the penalty function can change over generations. In summary, the main challenge of using these penalty functions is the need to set the penalty factors heuristically. In practice, the penalty factor has a great impact on the final result, and it needs to be updated at different stages to add the appropriate penalty value in the fitness evaluation for better results. In traditional methods, these factors usually need to be set manually and continuously adjusted based on expertise, which defies the concept of automatic design algorithms devoted in the proposed algorithm.

To address this concerning issue, several adaptive penalty functions [45] have been suggested. In the proposed method, we apply the idea of the adaptive penalty method in which the proportion of feasible solutions in the population is used as the penalty factor, and the factor is continuously adjusted with the number of feasible solutions at each generation. As the feasible individuals increase, the penalty for fitness decreases and more attention is paid to the performance, enabling a balance between performance and constraints. It can achieve effective constraint handling without the need to adjust the penalty factor based on expert experience, and the search process under constraint can be completed automatically.

3) Repair Methods: The repair methods can quickly fix an infeasible solution to make it feasible by adjusting x. They are generally implemented through a set of well-designed algorithms to reposition infeasible solutions into feasible regions in an effective manner [46]. In complex constrained optimization problems where the feasible space is often hard to reach, repair methods can help the solution enter the feasible regions more directly and thus obtain the feasible one efficiently. However, the repair method requires specific repair strategies for different situations, and there is no universal repair mechanism presented by existing approaches. Therefore, it is essential to design specific repair methods for different constraints and target problems. At the same time, the performance is strongly influenced by the manual strategy. As expected, the repair method performing well is highly dependent on expert experience. Overall, the design of repair methods is difficult and highly dependent on researchers' knowledge in the field, which is considered unfriendly to the end users.

It is complicated to design the repair methods for each constrained optimization problem individually, and in order to obtain a general approach, the adaptive repair method was designed with the replacement ideas in [47]. The infeasible solution is directly transformed into a feasible solution by replacing some of the parameters in the infeasible solution with some of the coefficients in the feasible solution. The adaptive repair can be well performed according to the predefined replacement principle. Compared with designing different repair strategies for different situations, simple parameters substitution may affect the performance of the repaired solutions, but it can greatly improve the efficiency of constraint handling.

Current NAS-based approaches for lightweight models mainly draw on the idea of multiobjective optimization to balance objectives and constraints. The conversion of the constrained single-objective optimization problem into an unconstrained multiobjective optimization problem is initially proposed to avoid the disadvantage of traditional penalty methods that require setting penalty coefficients. However, it also increases the computational complexity during the comparison. The adaptive penalty method is no longer limited by penalty coefficients, while it can solve the constrained problem efficiently. Therefore, we initially choose the adaptive penalty idea as a constraint handling method in the proposed algorithm to satisfy specific constraint values. The idea of the adaptive penalty algorithm is simple and helps us search for the closest model to the constraint. In addition, the penalty algorithm can retain some infeasible solutions with high accuracy when guiding infeasible solutions into the feasible regions by penalty. This ensures the population diversity and helps to improve the accuracy. At the same time, we develop a repair algorithm to further speed up the search for feasible solutions and help to find excellent feasible solutions in the limited generations. In this way, we can solve the optimization problems efficiently while satisfying the constraints.

III. Proposed Algorithm

In this section, we first present the framework of the proposed CH-CNN algorithm in Section III-A and then detail its main steps in Sections III-B–III-F.

Algorithm 1 Framework of CH-CNN

```
Input: predefined building blocks, the population size $N$;
       the maximal generation number $G$, the target
       dataset.
Output: The discovered best architecture of CNN.
1  $P_0 \leftarrow$ Randomly initialize the population with the
   proposed variable-length encoding strategy;
2  $t \leftarrow 0$;
3  while $t < G$ do
4      Evaluate fitness of $P_t$, and adjust the fitness by the
       adaptive penalty method;
5      if $t < 1/2G$ then
6          Repair beneficial infeasible individuals by the
           adaptive repair method;
7          Train and evaluate the feasible individuals after
           repair;
8      end
9      $O_t \leftarrow$ Choose parent individual by the roulette
       selection operation to generate the offspring with the
       proposed mutation and crossover operations;
10     $P_{t+1} \leftarrow$ Environmental selection from $P_t \cup O_t$;
11 end
12 Return the individual having the best fitness in $P_{t+1}$.
```

A. Algorithm Overview

The framework of the proposed algorithm CH-CNN is given in Algorithm 1. The proposed algorithm is initiated with the predefined CNN building blocks, the preset resource constraint number, the population size, as well as the maximum number of generations, and through a series of evolutionary processes, the best CNN architecture satisfying the preset constraint is finally discovered. Specifically, a population of CNNs is randomly initialized with the predefined population size, and the architectures composed of the building blocks are encoded using the proposed encoding strategy (line 1). Then, the counter of the current generation is initialized to zero (line 2). During the evolution process, the fitness of each individual is evaluated on the given dataset during the fitness evaluation, and the penalty method is applied during the fitness evaluation to penalize infeasible individuals to obtain a penalized fitness (line 4). Then, according to the penalized fitness, the infeasible individuals whose fitness is better than that of the optimal feasible solution are repaired or the optimal infeasible solution is repaired if there is no feasible solution in the population at the early generations (line 6). The repaired feasible individuals are evaluated again. Then, parent individuals are selected based on fitness, and new offspring are generated by genetic operators, including crossover and mutation (line 9). A population of individuals who survives to the next generation is selected from the current population by environmental selection (line 10). Finally, the counter is increased by one and the evolution process is repeated until the counter exceeds the predefined maximum generation number.

B. Population Initialization

In CH-CNN, GAs are utilized as the optimization strategy to continuously search for a better CNN architecture. GAs are optimization methods based on a set of solutions (called populations) and search for the optimal one. Therefore, GAs require an initialization strategy that generates an initial population for the subsequent evolution. A good initial population influences the process of finding the global optimum, such as accelerating the population convergence and improving the accuracy of the final solution.

Algorithm 2 Population Initialization

```
   Input: the population size $N$.
   Output: Initialized population $P_0$.
1  $P_0 \leftarrow \emptyset$
2  while $|P_0| \leq N$ do
3      $L \leftarrow$ Randomly generate an integer from [3,18];
4      $list \leftarrow$ Create a list contains $L$ nodes;
5      $pool_{num} \leftarrow 0$;
6      for $node$ in the $list$ do
7          $p \leftarrow$ Uniformly generate a number from (0,1);
8          if $p<0.5$ then
9              $node.type \leftarrow 1$;
10             $node.info \leftarrow$ Uniformly generate a number from $\{0, 1, 2, 3, 4\}$;
11             $node.F1 \leftarrow$ Randomly generate an integer from the available numbers of feature maps $\{64, 128, 256\}$;
12             $node.F2 \leftarrow$ Randomly generate an integer from the available numbers of feature maps $\{64, 128, 256\}$;
13         else if $pool_{num} < 3$ then
14             $node.type \leftarrow 2$;
15             $q \leftarrow$ Uniformly generate a number from (0,1);
16             if $q<0.5$ then
17                 $node.info \leftarrow max$;
18             else
19                 $node.info \leftarrow mean$;
20             end
21             $pool_{num} = pool_{num} + 1$;
22         end
23     end
24     $P_0 \leftarrow P_0 \cup list$;
25 end
26 return $P_0$
```

Algorithm 2 shows the details of population initialization. Briefly, N individuals are initialized in the same way and then stored in P_0. During the initialization of individuals, the length of individual L is first randomly initialized, which corresponds to the depth of the CNN. The number of convolution architecture blocks is limited to [3, 15], and the number of pooling layers does not exceed three (line 13). Thus, the depth of CNN L is a random integer in [3, 18] (line 3). A linked list containing L nodes is created to encode an individual (line 4), then, the information of each node is configured (lines 6–23), and finally, the encoded linked list is stored into P_0 (line 24). During the configuration of each node, a random number p is generated from (0, 1) (line 7). If $p < 0.5$, the node is marked as a building block by setting the type attribute to 1. Otherwise, if the number of pooling layers is less than 3, the node indicates the pooling layer by setting the type to 2. For the building blocks, the specific type node.info is a randomly generated integer from $\{0, 1, 2, 3, 4\}$, which corresponds to the different architecture blocks. Also, the numbers of the input and the output feature maps are randomly generated from the available numbers of feature maps, say $\{64, 128, 256\}$, based on the settings recommended by the state-of-the-art CNNs [48], [49] and then assigned to the node.F1 and node.F2, respectively (lines 8–12). The pooling type is determined by the probability of flipping a coin. In particular, the pooling type node.info is set to the max pooling when q is below 0.5 and mean pooling otherwise (lines 16–20).

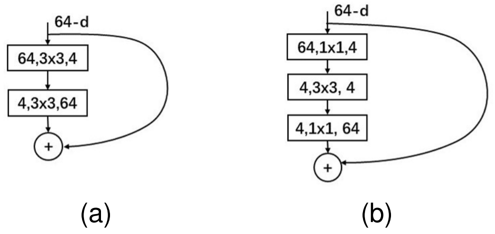

Fig. 2. Schematic of (a) skipV0 block and (b) skipV1 block.

The initialization process aims at generating the individuals of the population based on the encoding strategy. In order to generate a promising initial population, a suitable encoding strategy needs to be designed. The performance of a CNN highly relies on its architecture, and thus, new building blocks can form novel CNN architectures. In this way, we design five building blocks, convolution block, skipV0 block, skipV1 block, groupV0 block, and groupV1 block, which are constructed based on the different compositions of the ordinary convolution, the skip connection, the bottleneck architecture, and the group convolution. Specifically, the convolution block consists of two 3×3 convolutional layers on which the skipV0 block and the groupV0 block are adjusted. The skipV0 block adds a skip connection to the convolution block, while the groupV0 block changes the first layer to group convolution. In addition, the skipV1 block and the groupV1 block are the corresponding bottleneck blocks of skipV0 block and groupV0 block by using the bottleneck architecture to reduce the CNN parameters. These building blocks are used in the proposed encoding strategy.

Fig. 2 shows the schematic of the skipV0 block and skipV1 block. The skipV0 block consists of two 3×3 convolutional layers and a skip connection. The skip connections connect from the input of the first convolutional layer to the output of the second convolutional layer. To reduce the number of parameters, the two convolutional layers in the skipV0 block are replaced by a bottleneck architecture consisting of 1×1 and 3×3 convolutions to generate the skipV1 block. The groupV0 block and the groupV1 block generated by adding group convolution to skipV0 and skipV1 are shown in Fig. 3. The groupV0 block changes the 3×3 convolutional layer in the skipV0 block. Similarly, the groupV1 block is improved from the skipV1 block by setting the convolutional layers to group convolutions. In the groupV0 block and the groupV1 block, there is a fixed rule to set the number of groups for group convolutions. When the numbers of input and output channels of feature maps are the same, the group number is equal to the channels number. Otherwise, the number of groups is four. The internal design of the building block imitates the bottleneck layer of ResNet [49]. As a result, the number of channels will become 1/4 of the situation when the bottleneck

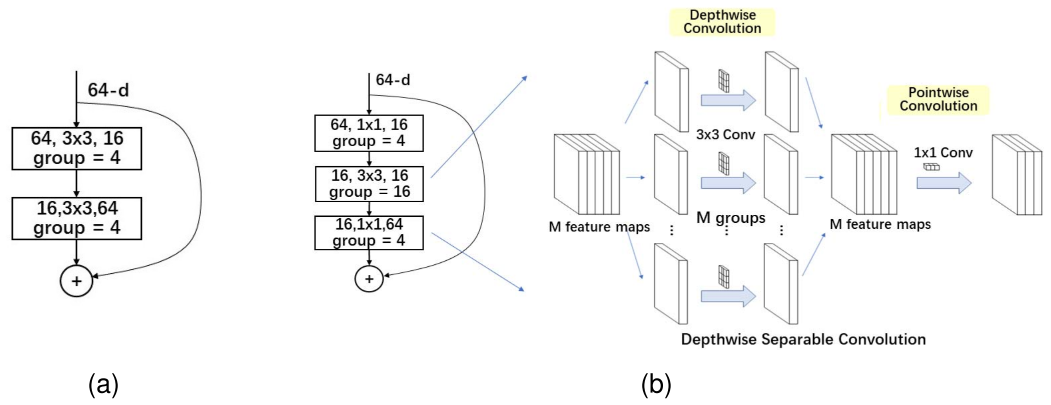

Fig. 3. Schematic of (a) groupV0 block and (b) groupV1 block.

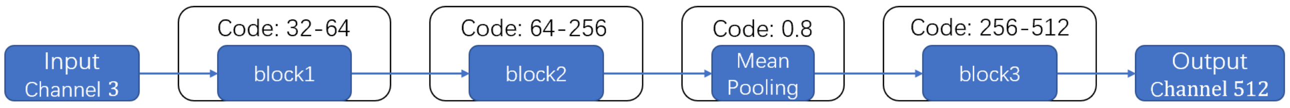

Fig. 4. Example of representing a CNN with the proposed encoding strategy. This CNN consists of three blocks and one pooling layer. The genotype encoding this CNN consists of the codes representing each layer or block. For the block, its codes are the number of input and output feature maps, while the code for each pooling layer is the pooling type. We use a number between (0, 0.5) to represent the max-pooling type and a number between [0.5,1) to represent the mean-pooling type. The numbers above each layer are the code of the corresponding layer. In summary, the code representing this CNN is "(1-32-64)-(2-64-256)-0.8-(3-256-512)."

layer is not used. In Fig. 3(b), group convolution is combined with the bottleneck architecture to form a depthwise separable convolution in the last two layers of groupV1 block, which further reduces the number of parameters needed for the block.

As introduced above, we use the rules of the fixed setting for the filter sizes, stride sizes, and group number of the building blocks in the proposed encoding strategy. Thus, the parameters encoded for the building block are the numbers of the input and output feature maps (denoted as $F1$ and $F2$, respectively). An example of the proposed strategy for encoding a CNN is shown in Fig. 4. The genotype of this CNN architecture consists of the codes representing each pooling layer and each block. For each building block, its codes are the numbers of input and output feature maps, while the code for the pooling layer is the pooling type. We use a number between 0 and 0.5 to represent the max-pooling type and a number in [0.5,1) to represent the mean pooling type. As a result, the genotype representing this CNN is "(1-32-64)-(2-64-256)-0.8-(3-256-512)."

C. Fitness Evaluation and Penalty

Algorithm 3 details the fitness evaluation of each individual in the population. Briefly, for each individual in the population P_t, its fitness is trained and evaluated using the image classification dataset. Immediately after, the fitness of infeasible individuals is adjusted according to the predefined constraint value that is set as the threshold. The penalty method performs constraint handling based on the accuracy and the complexity, which refers to the number of model parameters. Finally, the penalized fitness of individuals is obtained for the subsequent process. Specifically, the corresponding CNN model is first constructed using the information in the individual codes and the number of classifications in the dataset (line 1). After that, the CNN model is trained by the stochastic gradient descent (SGD) algorithm [50] on the training dataset (line 5), and the classification accuracy and complexity are calculated on the fitness evaluation data (line 6). During the process, the number of model parameters of the CNN is defined as the complexity to continuously optimize the size of the CNN model. After the training phase, each individual CNN gets the classification accuracy and complexity on the fitness evaluation data (lines 8 and 9). Finally, the final fitness is obtained after penalizing according to the preset constraint threshold (line 12).

Next, the penalty process for individuals is described in detail. After the evaluation of the population, we compare the number of individual CNN parameters with the preset constraint values C to judge whether the underlying individual CNN is a feasible one. Then, we use an adaptive penalty function to impose a penalty on the fitness of each infeasible individual. The adaptive penalty function is formulated by

$$\text{fitness} = \phi \times \text{accuracy} - (1-\phi) \times \text{CV} \tag{5}$$

where CV is the normalized constraint violation value and ϕ denotes the feasibility ratio of the current population, which are defined as

$$\text{CV} = \frac{|\text{complexity}_i - C|}{|\text{complexity}_{\max} - C|} \tag{6}$$

$$\phi = \frac{N_f}{N} \tag{7}$$

where complexity_i is the current individual complexity, $\text{complexity}_{\max}$ is the maximal complexity value of individuals in a population, C is the resource constraint threshold, and N_f and N represent the number of the feasible individuals and all the individuals in the current population, respectively.

Note that we use ϕ and $1-\phi$ as the coefficients of accuracy and constraint violation, respectively. The symbol ϕ denotes

Algorithm 3 Individual Fitness Evaluation

```
Input: The individual $individual$, the Specified GPU,
       the number of training epochs $epochs$, the
       training dataset $D_{train}$, the evaluation dataset
       $D_{eval}$, and the preset $constraint$.
Output: The individual $individual$ with its $accuracy$,
        $complexity$, and the fitness after penalty
        function $eval\_fitness$.
1  Construct the CNN based on the information encoded in
   the individual and number of classifications in the
   dataset;
2  accuracy $\leftarrow$ 0;
3  complexity $\leftarrow$ 0;
4  for $i \leftarrow 1$ to $epochs$ do
5      Train the CNN on $D_{train}$;
6      $eval\_accuracy, eval\_complexity \leftarrow$ Evaluate the
       accuracy and complexity on $D_{eval}$;
7      if $accuracy < eval\_accuracy$ then
8          $accuracy \leftarrow eval\_accuracy$;
9          $complexity \leftarrow eval\_complexity$;
10     end
11 end
12 $eval\_fitness \leftarrow$ the fitness after penalty function with
   $accuracy$, $complexity$ and $constraint$;
13 return $eval\_fitness$, $accuracy$, $complexity$.
```

the feasibility ratio of the current population. In the earlier stage of the evolution, it is difficult to search for feasible individuals that satisfy the constraints exactly due to the random initialization of the population. As expected, there are a few feasible individuals in the population, and ϕ is close to 0 so that the fitness is mainly affected by the constraint violation CV. Briefly, those infeasible individuals with higher constraint violations are penalized more than those with lower constraint violations and have a smaller fitness value at the early stage. As the number of feasible individuals grows, ϕ continues to increase, the accuracy of the individual begins to mainly affect the fitness. In this way, the feasible solution can be searched at first, and the feasible solution can be continuously optimized during the evolution process.

D. Population Repair

According to Section III-C, the repair method can directly adjust infeasible solutions to bring them into feasible regions. This can quickly obtain feasible solutions to help the search for solutions with better performance in the feasible regions. Since the NAS algorithm needs to search for optimal individuals that satisfy the resource constraints within a finite number of generations, we use an adaptive repair approach in the early generations to repair the infeasible individuals whose fitness values are greater than that of the optimal feasible solution to improve the effect of constraint handling. In order to search for lightweight CNN models, we still set the number of model parameters as a constraint. The number of model parameters is the total number of parameters to be trained in the CNN model, determined by all operations in the architecture. Therefore, we reduce the number of model parameters by adjusting the network architecture composition to repair the infeasible individuals.

Algorithm 4 Individual Repair

```
Input: The individual $individual$ with $fitness$ and the
       infeasible type $cv_{type}$
1  , the fitness of the best feasible individual $fitness_{best}$, the
       constraint value $C$, the acceptable feasible range
       $Complexity_{range}$, the number of adjustments $N$.
       Output: The individual after repair.
2  $list_c \leftarrow \emptyset$;
3  $list_i \leftarrow \emptyset$;
4  $indi \leftarrow individual$;
5  foreach $i$ in $N$ do
6      if $fitness > fitness_{best}$ then
7          if $cv_{type} == 0$ then
8              foreach node in individual do
9                  change the type of building block:
10                 skipV0 $\rightarrow$ groupV1 or skipV1;
11                 groupV0 or skipV1 $\rightarrow$ groupV1;
12                 convolution $\rightarrow$ skipV1, groupV0 or
                   groupV1;
13             end
14         else if $cv_{type} == -1$ then
15             foreach node in individual do
16                 change the type of building block according
                   to the correspondence rules;
17             end
18         end
19     end
20     $list_i[i] \leftarrow$ the $individual$;
21     $list_c[i] \leftarrow$ the complexity of the $individual$;
22 end
23 $CV_{best} \leftarrow N + 1$;
24 for $i$ in the $N$ do
25     if $|list_c[i] - C|$
                                  $<$
       $Complexity_{range}$ then
26         $Complexity_{range} \leftarrow |list_c[i] - C|$;
27         $CV_{best} \leftarrow i$;
28     end
29 end
30 if $CV_{best} \neq N + 1$ then
31     return $list_i[CV_{best}]$
32 else
33     return $indi$;
34 end
```

Algorithm 4 shows the individual repair process by using the adaptive repair method. First, the number of parameters of a given individual is compared with the constraint threshold to determine whether the individual is feasible or not. In the proposed algorithm, we search for the model whose parameters keep approaching the constraint value, which is an equality constraint. Because it is difficult to exactly find the model that satisfies the equality constraint during the optimization, we set a feasible range Complexity$_{\text{range}}$ on both sides of the constraint value. There are similarly two infeasible cases: exceeding the maximum value of the feasible domain, $\text{cv}_{\text{type}} = 0$, or less than the minimum value of the feasible range, $\text{cv}_{\text{type}} = -1$. The symbol cv_{type} represented the infeasible type of each individual. Since the repair operation is only targeted for the infeasible individuals, the fitness of the infeasible individual is compared with the threshold, which is usually the best fitness of the feasible ones (line 6). If there is no feasible individual in the population, the threshold is zero. Then, if the fitness of the infeasible individual is better, the individual will be adjusted to meet the complexity constraint (lines 5–22). Because the adjustment of the individual architecture is random, it is difficult to get the optimal repair effect immediately. We try to obtain the best repaired individual whose complexity is the closest to the constraint value by multiple repairs (lines 24–29). Finally, the architecture closest to the constraint value is selected to replace the infeasible individual (lines 30 and 30). If the architecture still does not meet the constraint after the adjustment, the initial infeasible individual will keep unchanged (lines 32–34). During the adjustment, there are two specific cases: the number of parameters is greater than the target constraint value and the number is less than the constraint value. When the number of parameters is greater than the constraint value, the convolution block, the skipV0 block, the skipV1 block, and the groupV0 block are probabilistically adjusted, which is using bottleneck architecture and group convolution to reduce the parameter number (lines 9–12). Otherwise, the blocks are changed to increase the number of parameters with the preset rules (lines 15–17).

Because the repair operation meets the constraint of parameters by changing the architecture, the changed architecture needs to be retrained, which is time-consuming. Considering that there are commonly enough feasible individuals for optimization in the later evolution generations, the repair operation is only used in the earlier stages of the evolution to improve the computation efficiency of the proposed algorithm.

E. Offspring Generation

After evaluating the repaired population, the parents are first selected according to the fitness, and then, they generate the offspring by the genetic operators, which consists of the crossover operation and the mutation operation implementing the local search and global search for the EA.

1) Crossover Operation: During the crossover operation, there will be totally N offspring generated, where N refers to the preset population size. Specifically, two individuals are randomly selected first, and then, one of the two individuals is chosen as the parent based on their fitness values. The selection operation is repeated until N parents are selected. Once all the parents are selected, a random number is generated to determine whether the crossover will be performed or not. If the generated number is less than the predefined crossover probability, each parent individual is randomly split into two parts, and the one-point crossover is realized to create two offspring. Otherwise, the two parents directly enter the offspring population for the next operation.

2) Mutation Operation: The mutation is a random adjustment of the new individuals produced after the crossover operation. Mutation can occur at any position on the individual, and multiple positions on the architecture can mutate at the same time, which can be a source for continuing genetic diversity [51]. During the mutation operation, a random number is generated first, and then, the mutation operation is performed on the current individual if the generated number is smaller than the predefined mutation probability; otherwise, the individual keeps unchanged. In the proposed algorithm, the available mutation operations defined in the mutation list are as follows:

1) adding a building block with random settings;
2) adding a pooling layer with random settings;
3) removing the layer at the selected position;
4) randomly changing the parameter values of the layer at the selected position.

In the proposed algorithm, we expect to improve the performance of some individuals by fine-tuning the network architectures using the mutation operator. The mutation operator can search for the local optimum around the individual by varying the encoded information over a given range. As shown above, four different types of mutation operators are designed in the proposed algorithm. The first three mutation operators are designed to find the suitable depths for the best performance. In particular, the first two operators increase the CNN depth by using the proposed building block or pooling layer, which is based on the inference that a deeper CNN will have the more powerful capability. The fourth mutation operator is designed to adjust the network architecture parameters to explore better performance at the same CNN depth. The mutation operator can change both the type of architecture units and the feature map channels. As for the unit type, the building block type can be adjusted by randomly selecting the corresponding number from $\{0, 1, 2, 3, 4\}$, and the type of pooling layer can be changed by reinitializing a random number from (0, 1). At the same time, the input and output channel number can be adjusted by selecting the number from the preset candidate channel numbers $\{64, 128, 256\}$. During evolution, each individual has a chance to be mutated, which accelerates convergence to the optimal solution while maintaining population diversity.

F. Environmental Selection

The environmental selection helps to select promising individuals for the next generation from the current population that is doubled in size from parents to offspring. In principle, both elitism and diversity should be considered during this process. In order to maintain a population with promising convergence and diversity, the selection operation, specifically roulette wheel selection [52], [53], is used in the environmental selection. The basic idea is that the probability of each individual being selected is proportional to its fitness. The elitism is considered to ensure that the performance becomes better. After roulette wheel selection, the selected individuals are checked whether they contain the best individual before the environmental selection, and then, the selected individual with the worst fitness is replaced by the best individual.

Algorithm 5 shows the details of the environmental selection. First, the sum of the fitness values of all individuals, Sum, is used to build a roulette wheel (line 2). Then, a random number r from (0, 1) is uniformly generated to represent the position of the pointer on the roulette (line 5). Finally, the individual is selected according to the position on the roulette wheel (lines 6–13). Specifically, the value roulette on the wheel is represented by r multiplied by Sum (line 8). The fitness of individuals in the population is summed sequentially to obtain num and the individual whose fitness num greater than roulette is selected for the next generation (lines 9–12).

Algorithm 5 Environmental Selection

```
Input: The current population $P_t \bigcup Q_t$ containing
       individuals with $fitness$, the population size $N$.
Output: The selected population $P_{t+1}$.
1  $P_{t+1} \leftarrow \emptyset$;
2  $Sum \leftarrow$ The sum of fitness of individuals in $P_t \bigcup Q_t$;
3  for $i \leftarrow 1$ to $N$ do
4      $num = 0$;
5      $r \leftarrow$ Uniformly generate a number from (0,1);
6      foreach $individual$ in $P_t \bigcup Q_t$ do
7          $num \leftarrow num + individual.fitness$;
8          $roulette \leftarrow r \times Sum$;
9          if $num > roulette$ then
10             $P_{t+1} \leftarrow P_{t+1} \bigcup individual$;
11             break
12         end
13     end
14 end
15 $P_{best} \leftarrow$ Find the individual with the best fitness from
   $P_t \bigcup Q_t$;
16 $P_{worst} \leftarrow$ Find the individual with the worst fitness from
   $P_{t+1}$;
17 if $P_{best}$ not in $P_{t+1}$ then
18     Replace the $P_{worst}$ in $P_{t+1}$ by $P_{best}$;
19 end
20 return $P_{t+1}$.
```

Finally, in order to ensure the elitism of the population and to enable the new generation to inherit the good attributes of the previous generations, we use an elitism strategy. If the best individual among the parent and offspring is not selected into the next generation after environmental selection, the best individual replaces the individual with the worst fitness (lines 15–19). The environmental selection can ensure the diversity and elitism of the population by using the roulette wheel selection and the elitism strategy.

IV. Experiment Design

In order to quantify the performance of the proposed CH-CNN algorithm for searching lightweight CNN models while satisfying the specific resource constraint, a series of experiments is conducted under different resource constraints and compared with the state-of-the-art peer competitors on the benchmark dataset. Since there are two main categories of peer competitors, the manual-designed methods, and NAS-based methods for the lightweight models, we have designed two sets of experiments in order to make detailed comparisons separately. In addition, we also conduct experiments under the constraint to analyze the variation in the parameter number and accuracy of feasible individuals searched during the optimization, to demonstrate that the proposed constraint optimization methods can efficiently help search for high-performance feasible individuals as envisioned. In the following, the peer competitors chosen to compare with the proposed algorithm are introduced in Section IV-A. Then, the used benchmark datasets are detailed in Section IV-B. Finally, the parameter settings of the proposed algorithm are shown in Section IV-C.

A. Peer Competitors

In order to show the effectiveness and efficiency of the proposed algorithm, the state-of-the-art algorithms are selected as the peer competitors for comparison. In particular, the peer competitors are selected from two different categories: manual design lightweight models and automated design lightweight models via NAS.

At present, researchers have manually designed a number of well-regarded lightweight CNN models with promising performance, including MobileNet [2], MobileNetV2 [54], ShuffleNet [10], ShuffleNetV2 [55], and SqueezeNet [9]. Specifically, MobileNet is the first small size, less computation, and mobile-friendly CNN model proposed by Google. MobileNetV2 is improved upon MobileNet by adding the skip connections and incorporating an inverted residual block to improve accuracy while maintaining the lightweight model. ShuffleNet is a lightweight network architecture proposed by Face++ Inc. Its main idea is to improve ResNet by using group convolution and channel shuffle that significantly reduces the network size. SqueezeNet develops squeeze modules and compression techniques to design the lightweight model yet with good performance.

As for the peer competitor from the second category, we select the very recently published, state-of-the-art algorithm, NSGA-Net [17], which also applies evolutionary computation as the search strategy. NSGA-Net leverages the ability of classical multiobjective optimization algorithm, NSGA-II [19], to achieve a balance between model accuracy and complexity, continuously selecting the models with better performance and lower complexity during the search process. It achieves excellent accuracy on the CIFAR-10 dataset on par with other state-of-the-art NAS methods but uses orders of magnitude less computational resources.

Similar to the proposed CH-CNN method, NSGA-Net also focuses on the impact of the complexity on CNN models' performance. In the experiment of the NSGA-Net, computational complexity is defined by the number of FLOPs that a network carries out during a forward pass. However, FLOPs is considered indirect indicator of speed [55] and is mostly used as computational complexity measurement to evaluate the latency of CNN models. To meet the needs of particular model size and the requirements of lightweight, we set the number of parameters as the constraint to optimize the models in the CH-CNN. In order to fairly compare with the proposed algorithm, we set the objectives of NSGA-Net as the number of parameters and classification accuracy in the experiments, then run the NSGA-Net algorithm on the chosen benchmark datasets, and finally compare the optimal results for the experiments.

B. Benchmark Datasets

Based on the conventions, the CIFAR10 and CIFAR100 datasets [29] are selected as the benchmark datasets in the experiments. Specifically, the CIFAR10 and CIFAR100 datasets are for the image classification task and are challenging in terms of image size, classification category, noise, and rotation of each image. Both have been widely used to measure the performance of deep learning algorithms and NAS algorithms [48]. Specifically, the CIFAR10 is a dataset to identify ten categories of general objects, such as airplanes, automobiles, and birds, and each category has an equal number of images. It consists of 60 000 RGB images with the size of 32×32. Among them, 50 000 images are serving as the training dataset, and another 10 000 images are reserved for the testing. The CIFAR100 is similar to CIFAR10, except that it has 100 classes from 20 superclasses.

In the experiments, the training dataset is split into two parts. The first accounts for 90% to serve for training the individuals, while the remaining images reserve as the evaluation dataset for evaluating the fitness of individuals. Finally, the effect of the network model is verified using the test dataset, and then, the results are reported for the comparisons.

C. Parameter Settings

The parameter settings include the settings for the peer competitors and those for the proposed algorithm. For a fair comparison, the parameters of the peer competitors are all set based on the respective seminal papers. This is because the settings in the seminal papers are often well-tuned to achieve their respective best performance. As for the parameters of the proposed algorithm, all are set based on the conventions of the communities of evolutionary computation [19] and deep learning [56], which is often regarded as the most common practice of the EA-based NAS community. Specifically, the population size and generation number are both set to 20. The probabilities of crossover and mutation are 0.9 and 0.2, respectively. As for the proposed four mutation operators shown in Section III-E2, we set the probability of adding new building blocks as 0.7, and the probability of three remainings is the same as 0.1. We provide a higher probability for the "adding a building block" mutation, which will give a higher probability of increasing the depth of the CNN. For the other mutation operations, we use equal probabilities. Such a mutation operation design will meet the belief that a deeper CNN will have better performance.

During the population initialization, the maximum numbers of the building blocks and the pooling layers are 15 and 3, respectively. In addition, the available numbers of feature maps for encoding are set to $\{64, 128, 256\}$ based on the settings employed by the state-of-the-art CNNs. In the training process of each individual, the SGD [50] with the learning rate of 0.1 and the momentum of 0.9 is used to train 350 epochs and the learning rate is decayed by a factor of 0.1 at the 1st, 149th, and 249th epochs [48]. During the evolution process, the constraint condition is set to the number of parameters of CNNs. The feasible range is set to 1/10 of the constraint threshold on both sides of the constraint, i.e., when the resource constraint is set to 1.0 M, the feasible range is considered from 0.9 M to 1.1 M.

In order to better compare and demonstrate the search effect of the CH-CNN algorithm in handling specific resource constraints, we set the constraint values according to the range of search results of the peer competitor NSGA-Net. NSGA-Net can find the Pareto optimal set within a certain constraint range and, finally, select the best solution that satisfies a specific constraint. For a fair comparison, we set the constraint values uniformly within the constraint range of the optimal set of NSGA-Net to simulate the resource limitations of the devices. The experimental codes have been implemented on Python 3.7 and PyTorch 1.2.0, which is a popular deep learning library [57], and all the experiments have been performed on three GPU cards with the same models of GeForce RTX 2080 Ti.

TABLE I
EXPERIMENTAL RESULTS WITH THE BEST ACCURACY OF THE PROPOSED CH-CNN ALGORITHM UNDER FIVE CONSTRAINTS OF 0.2, 0.4, 0.6, 0.8, AND 1.0 M ON THE CIFAR10 DATASET

Model	Constraint(M)	Accuracy(%)	Parameters(M)
CH-CNN	0.2	92.3	0.211
	0.4	94.1	0.407
	0.6	93.7	0.587
	0.8	94.4	0.837
	1.0	94.5	1.041
MobileNet [2]	-	91.1	3.217
MobileNetV2 [55]	-	91.5	2.296
ShuffleNet [10]	-	92.2	0.925
ShuffleNetV2 [56]	-	92.3	1.268
SqueezeNet [9]	-	92.6	0.741

V. EXPERIMENTAL RESULTS AND ANALYSIS

In this section, the results of CH-CNN with different parameter constraints are presented, along with the comparison with state-of-the-art methods to show the capability of CH-CNN in searching for high-accuracy models that satisfy the specific constraints. Specifically, we compare two types of methods, the state-of-the-art manual-designed CNN models and the state-of-the-art multiobjective NAS algorithm, i.e., NSGA-Net, this is the main motivation for the proposed algorithm. Then, the experimental results and the search process under a single constraint are specifically analyzed. Due to space limitation, this part of experimental analysis has been placed in the Supplemental Materials attached to this article. To acquire extensive comparisons for the peer competitors and the proposed CH-CNN method, the evaluation indicators are used to analyze the experimental results. We employ two evaluation indicators, including classification accuracy and complexity (number of parameters).

A. Comparisons With State-of-the-Art Manual-Designed CNN Models

The CH-CNN constantly searches for the CNNs with better accuracy in the feasible region of 1/10 on both sides of the constraint value. After the evolution, the proposed CH-CNN algorithm generates many CNN architectures that satisfy the constraint but with different classification performances. We select the CNN with the best classification accuracy performance among them with the similar parameter number as the best result for comparisons against the state-of-the-art lightweight CNN models. Tables I and II show the comparisons on the CIFAR10 and the CIFAR100 benchmark datasets. The experimental results of the proposed CH-CNN algorithm under different constraints on the CIFAR10 are listed in Table I, where each row represents the optimal solution obtained by the chosen peer competitors. For the proposed CH-CNN algorithm, we provide its best results under different constraints, and the second column shows the corresponding constraint value. The experimental results of the state-of-the-art small networks and the proposed CH-CNN algorithm under four constraints on the CIFAR100 are listed in Table II.

The experimental results on the CIFAR10 are listed in Table I. First, it shows that CH-CNN found high-accuracy CNN models satisfying the different constraints. Under the

TABLE II

COMPARISON OF STATE-OF-THE-ART SMALL NETWORKS AND THE PROPOSED CH-CNN ALGORITHM OVER CLASSIFICATION ACCURACY, THE NUMBER OF WEIGHTS ON THE CIFAR100 DATASET

Model	Constraint(M)	Accuracy(%)	Parameters(M)
CH-CNN	0.8	74.1	0.872
	1.2	74.3	1.229
	1.6	75.9	1.636
	2.0	74.9	1.962
MobileNet [2]	-	66.5	3.309
MobileNetV2 [54]	-	67.1	2.412
ShuffleNet [10]	-	71.0	1.012
ShuffleNetV2 [55]	-	69.3	1.360
SqueezeNet [9]	-	70.9	0.786

minimum resource constraint of 0.2 M, CH-CNN finds the model with 92.3% accuracy. Under the 0.4 M constraint, the number of parameters of the optimal result is very close to the constraint value, with a difference of only 0.007 M, while the accuracy reaches 94.1%. Under the 0.6 M constraint, the classification accuracy reaches 93.7%, and the number of parameters differs from the constraint value by 0.013 M. Under the 0.8 and 1.0 M constraints, the accuracy significantly exceeds 94%, reaching 94.4% and 94.5%. Compared with the manual design lightweight CNN models, CH-CNN found good solutions with a significantly smaller number of parameters than those of MobileNet, MobileNetV2, and shuffleNetV2 for each of the five constraints and with better classification accuracy. It is obvious that MobileNet, MobileNetV2, and shuffleNetV2 do not satisfy the preset constraints. SqueezeNet satisfies the constraint of 0.8 M, and the accuracy is 92.6%, which is less than the accuracy of the optimal CNN model searched by CH-CNN, 94.4%. Similarly, ShuffleNet is compared with the optimal model of CH-CNN under the 1.0 M constraint. The accuracy of ShuffleNet is less than that of the CH-CNN optimal model, with a difference of 2.3%.

Table II compares the results of the state-of-the-art lightweight CNN models and the CH-CNN on the CIFAR100 dataset. The CH-CNN has achieved great results under different resource constraints. All the accuracy is over 0.74, and the accuracy of a good CNN model under the 1.6 M constraint reached 0.759. As for the complexity, the errors between the number of parameters of the optimal CNN models and the constraint value are less than 0.1 M, and the minimum error is just 0.029 M under the 1.2 M constraint. The accuracy of the models found by CH-CNN under different constraints is significantly better compared to the manual-designed models. Meanwhile, the numbers of parameters for MobileNet and MobileNetV2 are much larger than the parameters numbers for the models searched by CH-CNN. For ShuffleNet, the optimal individual with the 0.8 M constraint has the closest number of parameters. It achieved a higher classification accuracy than that of the ShuffleNet, with a difference of 3.1%. ShuffleNetV2 is the closest to the 1.2 M constraint, but the difference with the constraint value is significantly larger and the accuracy is merely 69.3%, much lower than the CH-CNN model under the 1.2 M constraint. Finally, SqueezeNet satisfies the preset constraint of 0.8 M, but its accuracy is worse than the model found under the limited constraint, with a difference of 3.2%.

B. Comparisons With State-of-the-Art Multiobjective NAS (NSGA-Net)

The comparison results between CH-CNN and NSGA-Net on the CIFAR10 and CIFAR100 datasets are shown in Table III and Fig. 5(a) and (b), respectively. There are three comparative metrics: accuracy, complexity (number of parameters), and GPU days, the time for the algorithm finding the optimal solution in the table. For the accuracy metric, both the optimal result and the average result of ten runs are reported to demonstrate the performance of the proposed algorithm. In Fig. 5(a) and (b), the vertical axis denotes the classification error rate, while the horizontal axis refers to the parameters number. The blue dots in both figures are the optimal results after 30 generations of NSGA-Net. The red "+" markers are the results of the CH-CNN algorithm under different constraints in terms of the number of the parameters. The well-performing CNN models that satisfy the constraints are all in the black bounding box. Meanwhile, the green lines are used to highlight the difference in accuracy and complexity between the optimal CNN models of the CH-CNN and NSGA-Net. The green vertical dashed line represents the set constraint value, the green horizontal line is used to indicate the difference between the number of parameters of the optimal models and the constraint value, and the green vertical solid line is used to indicate the difference in accuracy between the optimal models. According to the figure, it can be found that the proposed CH-CNN algorithm can find the CNN models that meet the constraints more accurately and achieve greater performance.

As can be seen from Fig. 5(a), the optimal solutions obtained by NSGA-Net have the numbers of parameters in the range of 0–1.0 M on the CIFAR10. For a better comparison, the proposed CH-CNN algorithm experimented with the constraints that the numbers of parameters are specified on 0.2, 0.4, 0.6, and 0.8 M. It is clear that the results of the CH-CNN algorithm under a certain complexity constraint are better in Table III, according to the differences of the best accuracy. Under the 0.2 M constraint, the complexity of the optimal solution obtained by the two algorithms is similar, but the accuracy of CH-CNN is significantly higher (i.e., the error is lower), with a difference close to 3.0%. Under the 0.4 M constraint, the accuracy of individuals searched by CH-CNN is generally higher and has the largest difference of 3.3%. Also, the optimal individuals searched by CH-CNN are closer to the specific constraint of 0.4 M. Under the 0.6 M constraint, NSGA-Net finds individuals that are closer to the specific constraint, but the difference in the number of parameters is very small. More importantly, the individual accuracy of CH-CNN is significantly better, and the difference is 2.1%. Finally, under the 0.8 M constraint, CH-CNN finds the optimal individual with a classification error rate of less than 6.0%, which is nearly 2.0% different from the accuracy of the NSGA-Net optimal individual. Also, the difference with the constraint value is just 0.037 M, nearly 1/2 of the difference of the NSGA-Net optimal individual.

The comparison is also performed on the CIFAR100 dataset, and we also set the constraint values according to the parameter numbers of the NSGA-Net optimal set, and the results are shown in Fig. 5(b) and Table III. Under the 2.0 M constraint, CH-CNN is not as good as NSGA-Net. There is a 1.6% difference in accuracy. However, under the 0.8, 1.2, and 1.6 M constraints, CH-CNN obviously searches for individuals with higher accuracy.

The comparison experiments show that CH-CNN can search for high-performance individuals satisfying the constraint value. The higher accuracy can be achieved by continuously searching for better individuals near the preset constraint value. Compared with the NAS-based method for lightweight

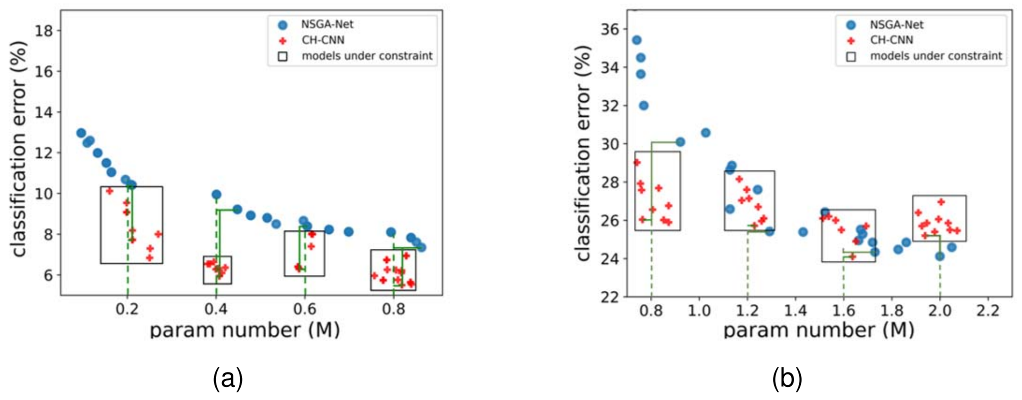

Fig. 5. (a) Results comparison between CH-CNN and NSGA-Net on the CIFAR10. (b) Results of NSGA-Net and CH-CNN on the CIFAR100.

TABLE III
COMPARISON OF NSGA-NET AND THE PROPOSED CH-CNN ALGORITHM ON THE CIFAR10 AND CIFAR100 DATASETS UNDER DIFFERENT CONSTRAINTS

Dataset	Constraint(M)	Method	Best Accuracy (Mean Accuracy)(%)	Parameters(M)	GPU Days
CIFAR10	0.2	CH-CNN	92.3 (89.6)	0.211	11
		NSGA-Net	89.6	0.210	8
	0.4	CH-CNN	94.1 (92.6)	0.407	9
		NSGA-Net	90.8	0.440	8
	0.6	CH-CNN	93.7 (92.6)	0.587	10
		NSGA-Net	91.6	0.605	8
	0.8	CH-CNN	94.4 (92.7)	0.837	10
		NSGA-Net	92.6	0.863	8
CIFAR100	0.8	CH-CNN	74.1 (70.7)	0.872	20
		NSGA-Net	68.0	0.769	12
	1.2	CH-CNN	74.3 (71.5)	1.229	22
		NSGA-Net	72.4	1.242	12
	1.6	CH-CNN	75.9 (73.0)	1.636	19
		NSGA-Net	74.7	1.678	12
	2.0	CH-CNN	74.9 (73.4)	1.962	18
		NSGA-Net	76.5	2.148	12

model algorithms, CH-CNN can be better targeted to meet specific constraint limits while achieving better accuracy.

VI. CONCLUSION

The main objective of this article is to propose an automatic architecture design algorithm based on real-world constraints (in short called CH-CNN), which can design a CNN architecture being capable of meeting a specific number of parameters and, at the same time, achieve a better image classification accuracy. This goal has been successfully achieved by developing a new encoding scheme of feature maps to generate individuals of different lengths with skip connections, group convolutions and bottleneck architectures, an adaptive penalty in the fitness evaluation process, and repair operations for infeasible individuals. The proposed algorithm has been examined under different resource constraints on the CIFAR10 and CIFR100 datasets and compared with a state-of-the-art peer competitor, NSGA-Net, and several manual-designed lightweight CNN models. The CH-CNN algorithm has achieved great performance under different memory capacity constraints. CH-CNN outperforms almost all of the manually designed CNNs while meeting different constraints. Compared with NSGA-Net, in which the restriction on complexity is more stringent, the proposed CH-CNN algorithm can find promising CNN models that satisfy specific constraints with the constraint handling methods. However, only the coarse-grained block-level replacement strategy was used in the repair algorithm, which might not fine-tune the number of model parameters to guarantee the best performance of the proposed algorithm. Moreover, the adaptive repair method requires retraining the architecture, which makes the proposed algorithm time-consuming. In CH-CNN, we use the number of parameters as the CNN complexity to search for a lightweight CNN architecture that satisfies a specific constraint. There are more metrics that need to be limited in real-world applications, such as energy consumption, spectrum efficiency, and latency. In the future, we will place efforts on searching for lightweight CNN models under real-world constraints by more efficient methods.

REFERENCES

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[2] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.

[4] Y. Tao, R. Ma, M.-L. Shyu, and S.-C. Chen, "Challenges in energy-efficient deep neural network training with FPGA," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2020, pp. 400–401.

[5] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

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

[7] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-ResNet and the impact of residual connections on learning," 2016, *arXiv:1602.07261*.

[8] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," 2015, *arXiv:1512.00567*.

[9] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size," 2016, *arXiv:1602.07360*.

[10] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," 2017, *arXiv:1707.01083*.

[11] N. Gunantara, "A review of multi-objective optimization: Methods and its applications," *Cogent Eng.*, vol. 5, no. 1, Jan. 2018, Art. no. 1502242.

[12] E. Liberis, L. Dudziak, and N. D. Lane, "μNAS: Constrained neural architecture search for microcontrollers," 2020, *arXiv:2010.14246*.

[13] C.-H. Hsu *et al.*, "MONAS: Multi-objective neural architecture search using reinforcement learning," 2018, *arXiv:1806.10332*.

[14] M. Tan *et al.*, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2820–2828.

[15] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," 2018, *arXiv:1812.00332*.

[16] J. Fernandez-Marques, P. N. Whatmough, A. Mundy, and M. Mattina, "Searching for winograd-aware quantized networks," 2020, *arXiv:2002.10711*.

[17] Z. Lu *et al.*, "NSGA-Net: Neural architecture search using multi-objective genetic algorithm," 2018, *arXiv:1810.03522*.

[18] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," in *Parallel Problem Solving From Nature PPSN VI*, M. Schoenauer *et al.* Eds. Berlin, Germany: Springer, 2000, pp. 849–858.

[19] 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.

[20] K. Deb, "Multi-objective optimization," in *Search Methodologies*. Boston, MA, USA: Springer, 2014, pp. 403–449.

[21] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, 2019.

[22] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," 1996, *arXiv:AI/9605103*.

[23] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *IEEE Trans. Neural Netw.*, vol. 9, no. 5, p. 1054, Sep. 1998.

[24] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," 2018, *arXiv:1806.09055*.

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

[26] D. Ashlock, *Evolutionary Computation for Modeling and Optimization*. New York, NY, USA: Springer, 2006.

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

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

[29] A. Krizhevsky, "Learning multiple layers of features from tiny images," M.S. thesis, Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, 2009.

[30] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 694–713, May 1997.

[31] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Mach. Intell.*, vol. 1, pp. 24–35, Jan. 2019.

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

[33] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci, "A hypercube-based encoding for evolving large-scale neural networks," *Artif. Life*, vol. 15, no. 2, pp. 185–212, Apr. 2009. [Online]. Available: https://doi.org/10.1162/artl.2009.15.2.15202

[34] X. Zhou, A. K. Qin, Y. Sun, and K. C. Tan, "A survey of advances in evolutionary neural architecture search," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2021, pp. 950–957.

[35] E. Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. 34th Int. Conf. Mach. Learn.*, D. Precup and Y. W. Teh, Eds., vol. 70, Aug. 2017, pp. 2902–2911. [Online]. Available: https://proceedings.mlr.press/v70/real17a.html

[36] 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, 2019, pp. 4780–4789.

[37] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 394–407, Apr. 2020.

[38] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely automated CNN architecture design based on blocks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 4, pp. 1242–1254, Apr. 2020.

[39] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Aug. 6, 2021, doi: 10.1109/TNNLS.2021.3100554.

[40] L. I. Zhi-Yong, H. Tao, C. Shao-Miao, and L. I. Ren-Fa, "Overview of constrained optimization evolutionary algorithms," *J. Softw.*, vol. 28, no. 6, pp. 1529–1546, 2017.

[41] C. A. C. Coello, "Constraint-handling techniques used with evolutionary algorithms," in *Proc. Genetic Evol. Comput. Conf. Companion*, Jul. 2021, pp. 675–701.

[42] Y. G. Woldesenbet, G. G. Yen, and B. G. Tessema, "Constraint handling in multiobjective evolutionary optimization," *IEEE Trans. Evol. Comput.*, vol. 13, no. 3, pp. 514–525, Jun. 2009.

[43] C. A. C. Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art," *Comput. Methods Appl. Mech. Eng.*, vol. 191, pp. 1245–1287, Jan. 2002.

[44] S. Venkatraman and G. G. Yen, "A generic framework for constrained optimization using genetic algorithms," *IEEE Trans. Evol. Comput.*, vol. 9, no. 4, pp. 424–435, Aug. 2005.

[45] B. Tessema and G. G. Yen, "An adaptive penalty formulation for constrained evolutionary optimization," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 39, no. 3, pp. 565–578, May 2009.

[46] S. Salcedo-Sanz, "A survey of repair methods used as constraint handling techniques in evolutionary algorithms," *Comput. Sci. Rev.*, vol. 3, no. 3, pp. 175–192, Aug. 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013709000379

[47] F. Samanipour and J. Jelovica, "Adaptive repair method for constraint handling in multi-objective genetic algorithm based on relationship between constraints and variables," *Appl. Soft Comput.*, vol. 90, May 2020, Art. no. 106143. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494620300831

[48] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing CNN architectures using the genetic algorithm for image classification," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3840–3854, Sep. 2020.

[49] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015, *arXiv:1512.03385*.

[50] J. M. Cherry *et al.*, "SGD: Saccharomyces genome database," *Nucleic Acids Res.*, vol. 26, no. 1, pp. 73–79, 1998.

[51] D. Whitley, "A genetic algorithm tutorial," *Statist. Comput.*, vol. 4, no. 2, pp. 65–85, Jun. 1994.

[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] D. Goldberg, G. D. Edward, D. Goldberg, and V. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning* (Artificial Intelligence). Reading, MA, USA: Addison-Wesley, 1989. [Online]. Available: https://books.google.com/books?id=2IIJAAAACAAJ

[54] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

[55] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 116–131.

[56] G. E. Hinton, "A practical guide to training restricted Boltzmann machines," in *Neural Networks: Tricks of the Trade*. Berlin, Germany: Springer, 2012, pp. 599–619.

[57] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 8026–8037.

Siyi Li (Graduate Student Member, IEEE) received the B.S. degree from the School of Computing, Sichuan University, Chengdu, China, in 2020, where she is currently pursuing the M.S. degree with the School of Computing.

Her main research interests include neural architecture search, evolutionary algorithms, and evolutionary deep learning.

Yanan Sun (Member, IEEE) is currently a Research Professor with Sichuan University, Chengdu, China. Prior to that, he was a Post-Doctoral Research Fellow with the Victoria University of Wellington, Wellington, New Zealand, from July 2015 to March 2019. He has published 36 peer-reviewed articles, including 15 articles in IEEE journals. He was the Thought Leader of Evolutionary Deep Learning from one of the six research focuses established in the Victoria University of Wellington. As the Principal Investigator, he has received two research grants from the Science and Technology Department of Sichuan Province and one from the National Natural Science Foundation of China.

Prof. Sun server as a reviewer for over 30 avenues. He is invited to be the organizing committee chair, the program committee chair, the special session chair, and the tutorial chair of nine international conferences. In 2016, he received the Best Student Paper Award of the IEEE CIS Chengdu Chapter, the National Scholarship of China, and the IEEE Student Travel Grant. He is the leading Organizer of one workshop and one special session in the topic of Evolutionary Deep Learning, and the Founding Chair of the IEEE CIS Task Force on Evolutionary Deep Learning and Applications. He is also a Guest Editor of the Special Issue on Evolutionary Computer Vision, Image Processing and Pattern Recognition in *Applied Soft Computing*.

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

In 1997, he was with the Structure Control Division, Air Force Research Laboratory, Albuquerque, NM, USA. He is currently a Regents Professor with the School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK, USA. His research interests include intelligent control, computational intelligence, conditional health monitoring, and 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 Vice President for the Technical Activities and then as the President for the IEEE Computational intelligence Society from 2005 to 2006 and from 2010 to 2011, respectively. 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 was the Founding Editor-in-Chief of the *IEEE Computational Intelligence Magazine* from 2006 to 2009. He was an Associate Editor of *IEEE Control Systems Magazine*.

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

He is currently a Professor of computer science, the Head of the Evolutionary Computation Research Group, and the Associate Dean of the Faculty of Engineering, Victoria University of Wellington, Wellington, New Zealand. 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. He is also a member of the IEEE CIS Award Committee and a Committee Member of the IEEE NZ Central Section. He is chairing the IEEE CIS Intelligent Systems and Applications Technical Committee and the Evolutionary Computation Technical Committee. He is also the Vice Chair of the IEEE CIS Task Force on Evolutionary Feature Selection and Construction and the Task Force on Evolutionary Computer Vision and Image Processing and the Founding Chair of the IEEE Computational Intelligence Chapter in New Zealand.