Particle Swarm Optimization for Compact Neural Architecture Search for Image Classification

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

***Abstract*—Convolutional neural networks (CNNs) are a superb computing paradigm in deep learning, and their architectures are considered to be the key to performance breakthroughs in various tasks. Recently, neural architecture search (NAS) methods have been proposed to automate the process of network architecture design, many of which have discovered novel CNN architectures that are superior to human-designed ones. However, most of the current NAS methods suffer from either prohibitively high computational complexity of resulting architectures which have inadvertently affected the deep model deployment, or limitations which impede the flexibility of architecture design. To address these deficiencies, this work proposes an evolutionary computation (EC) based method for compact and flexible NAS. A valuable search space with parameter-efficient mobile inverted bottleneck convolution blocks as the primitive components is proposed to ensure the initial quality of the compact architectures. In addition, a two-level variable-length particle swarm optimization (PSO) approach is devised to evolve both the micro-architecture and macro-architecture of CNNs. Further, this study proposes an effective scheme by integrating multiple computationally reduced methods to greatly speed up the evaluation process. Experimental results on the CIFAR-10, CIFAR-100 and ImageNet datasets show the superiority of the proposed method against the state-of-the-art algorithms in terms of the classification performance, search cost, and resulting architecture complexity.**

***Index Terms*—Convolutional neural networks, particle swarm optimization, neural architecture search, deep learning.**

I. Introduction

THE overwhelming success of deep convolutional neural networks (CNNs) has aroused extensive interest on deep learning technology all over the world. CNN forms a hierarchical information extraction architecture through the stacking of convolutions and non-linear activations, which is suitable for processing grid-like data such as images and videos [1]. Based on this advantage, more and more researchers from academia and industry have promoted CNNs in various applications, such as object recognition [2], action detection [3], video understanding [4], and autonomous driving [5], just to name a few. However, the design of CNN architectures is a complicated and laborious process, covering countless factors. One needs to determine the network macro-architecture (i.e., number of layers/blocks and the connection ways between the layers/blocks), and the network micro-architecture (i.e., type, number of kernels in each layer/block) of CNNs, which requires considerable engineering insights and domain expertise. Moreover, CNNs are often customized and trained on specific datasets, so relevant experience of deep model design is hard to be shared directly among different applications. Therefore, an efficacious method is eagerly needed to automate the process of CNN architecture design.

Neural architecture search (NAS), one of the most actively researched subfields of automated machine learning (AutoML) [6], has emerged in recent years, which is committed to discovering a novel and well-performing network architecture in a predefined search space. Up to now, many effective NAS algorithms have been proposed, and novel CNN architectures have also been automatically designed [7]. On some vision tasks, especially image classification, NAS has already achieved very high performance that humans cannot surpass [8]–[10] while saving numerous manpower. This in turn brings great convenience to the promotion and application of deep learning technology in more fields.

Despite the great achievements of NAS, its shortcomings have also been exposed in recent developments. One major problem comes from the search space design that many works overuse human ingenuity to predefine the overall backbone of the architecture in order to raise the lower bound of the searched architecture performance. For instance, NASNet [9] manually designed a bi-chain-styled macro-architecture and used a number of repeated cells/blocks to construct CNN architectures. Many state-of-the-art NAS methods [10]–[13] directly adopted the "NASNet search space" [9] to obtain effective architectures. The key architectural parameters such as depth, width, and resolution changes [2] of CNN architectures are often not fully explored. This kind of NAS algorithms is highly inflexible since the resulting architectures are unlikely to be sufficiently diverse and still rely heavily on the human expertise in CNNs.

Regarding the search strategy, early NAS methods [14],

This work was supported in part by the Marsden Fund of New Zealand Government under Contracts VUW1913, VUW1914 and VUW2115, the National Science Challenge – Science for Technological Innovation Challenge (SfTI) under contract 2019-S7-CRS, New Zealand MBIE Data Science SSIF Fund under the contract RTVU1914, New Zealand Ministry of Business, Innovation and Employment under contract C09X1923 (Catalyst: Strategic Fund), New Zealand MBIE Endeavour Research Programme under contract UoCX2104, National Natural Science Foundation of China (NSFC) under Grant 61876169, Zhejiang Lab (NO.2022PG0AB02), and the Science and Technology Innovation Talent Project of Sichuan Province under Grant 2021JDRC0001. This work of Junhao Huang was supported by the China Scholarship Council (CSC)/Victoria University Scholarship.

J. Huang, B. Xue and M. Zhang are with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. (e-mail: junhao.huang@ecs.vuw.ac.nz; bing.xue@ecs.vuw.ac.nz; mengjie.zhang@ecs.vuw.ac.nz).

Y. Sun is with the School of Computer Science, Sichuan University, Chengdu, China (e-mail: ysun@scu.edu.cn, corresponding author).

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

[15] are mostly based on reinforcement learning (RL) [16] to explore well-performing architectures. These works often suffer from expensive expenditure of computing resources, which is mainly due to the intractable optimization of the controller and the slow convergence of architecture search. While gradient-based (GD) NAS methods [11], [17] are much more efficient than RL-based ones, they typically need to build a supernet which involves considerable expertise and costs excessive GPU memory, and design a differentiable algorithm for gradient-based architecture search which might not be reasonable or convincing [7]. Among different search paradigms, evolutionary computation (EC) [18] is a flexible and viable alternative with great potential used in NAS. Basically, EC algorithms serve as a population-based global optimization method which is insensitive to local minima [19]. Therefore, they can handle the architecture design of deep models, which is a non-differentiable and non-convex optimization problem [7]. Specifically, particle swarm optimization (PSO) [20] is a promising EC algorithm in dealing with challenging optimization problems. Compared with genetic algorithms (GAs) which are a widely adopted method in EC-based NAS, PSO is gaining increasing interest with its simple implementation, and fast convergence [21], [22]. Furthermore, in NAS, there are some architectural parameters (e.g., the number of filters in the convolutional layer) which have a wider range of optional values linearly ordered, known as ordinals, in the search space. The smooth momentum-based updating style of PSO makes it possible to intuitively capture the magnitude relationship of this kind of architectural parameters within certain ranges, and effectively search for the appropriate values [19], [23].

Moreover, an NAS algorithm generally involves a large number of performance evaluations on candidate architectures. Each performance evaluation usually requires hundreds of gradient training epochs on the training dataset, which will lead to an extremely high computational cost overall. For example, the Large-scale evolution (LS-Evolution) [8] obtained the final architecture after more than 11 days of training on 250 graphics processing unit cards (GPUs), leading to 2,750 GPU days[1], and NASNet [9] consumed 2,000 GPU days to complete the CNN architecture search under the predefined backbone and the design of transferable cells. The notoriously time-consuming search process will discourage many researchers and institutions without enormous computing resources. Fortunately, various approaches have been proposed to address this issue, such as early stopping policy [24], [25] and low-fidelity proxy [22], [26], but the acceleration effect is not sufficiently satisfying and many still require dozens of GPU-days in architecture search.

From the perspective of the resulting architecture, many previous works focus primarily on improving the performance of the searched network architectures (e.g., classification accuracy) without considering complexity, which will result in the obtained network architectures being usually very complicated and large in size. An overly bloated model will occupy too much storage space and computing resources, inevitably creating obstacles for real-time applications on edge devices with limited computational capability, such as mobile phones and embedded devices. Therefore, in order to promote the application of CNNs in more fields, a compact and computationally efficient CNN design is absolutely needed.

In response to the aforementioned analyses and limitations of NAS, this paper aims to develop a flexible PSO algorithm for efficient and compact[2] CNN architecture design. In particular, a two-level variable-length PSO approach is devised to evolve both the micro-architecture and macro-architecture of CNNs. In addition, multiple computationally reduced methods (i.e., a dynamic early-stopping policy and two low-fidelity proxy factors) are well integrated to greatly speed up the architecture search process. The main contributions of the proposed efficient PSO algorithm for compact NAS (EPCNAS) are listed as follows:

1) From the perspective of architecture search space, the computationally and parametrically efficient mobile inverted bottleneck convolution (MBConv) block [27] is employed to build a search space with arbitrary connection ways. To improve the diversity of the evolved blocks, we remove the element-wise residual connection from input to output of a block, and instead use shortcut connections like in DenseNet [28] to concatenate the feature maps outputted by the previous multiple blocks. Constructing such an informative search space can provide a high-quality initial level and facilitate flexible and compact architecture search.
2) A two-level PSO approach is developed to evolve the entire CNN architecture. In particular, the network architecture is represented by a variable-length particle which is composed of two subparticles of different levels. The micro-level subparticle encodes the configurations of the architecture, including the depth and resolution changes of the network, number of output feature maps of each MBConv block, and the pooling type of each pooling layer, while the macro-level binary subparticle encodes the connection ways of these layers/blocks. In this manner, we do not need any preprocessing or definition of the overall backbone of the target network, which greatly reduces the user's need for prior knowledge of CNN construction.
3) The proposed variable-length and two-level encoding scheme will inevitably result in a massive search space. Therefore, an effective integrated acceleration scheme is proposed by combining an extended dynamic early stopping, downsampling (decrease the resolution of features), and architecture downscaling (reduce the number of filters) in a reasonable way. The three computationally reduced methods not only significantly accelerate the fitness evaluation process, but also effectively reduce the search space thereby speeding up the convergence of the architecture search, leading to orders of magnitude lower search cost on the CIFAR datasets than the original cost.
4) From the experimental results on three commonly used

[1] *the number of GPU days = the number of days to run× the number of employed GPUs*

[2] Regarding the compact CNN architecture design, this paper focuses on the optimization of the number of parameters.

image classification datasets: CIFAR-10 [29], CIFAR-100 [29] and ImageNet [30], as well as the comparison with state-of-the-art peer competitors, we demonstrate that among all the compared methods, the proposed method found the most compact architecture with highly competitive performance under a low computational consumption.

The remainder of this paper is organized as follows. In Section II, background and related works are presented. Section III elaborates on the proposed efficient variable-length PSO in details. Sections IV and V show the experimental design and results, respectively. Finally, the conclusions are drawn in Section VI.

II. Background and Related Work

In this section, one of the most popular compact CNN architectures, i.e., MobileNets, is introduced. Then, the related works on EC-based NAS and compact CNN architecture design are reviewed.

A. MobileNets

MobileNet [31] is a lightweight CNN designed to deal with the high complexity of traditional deep models. The depthwise separable convolution instead of standard convolution is used to construct the streamlined architecture of MobileNet, so as to greatly reduce the number of parameters and calculations of the network without appreciable performance loss. Depthwise separable convolution is a form of factorized convolutions which divides a single convolution operation into a depthwise convolution and a 1×1 pointwise convolution.

For a standard convolutional layer with m feature maps of size $M \times N$ as input and n feature maps of the same spatial size as output, given a 4-dimensional convolution kernel $\boldsymbol{\mathcal{W}}$ of size $d \times d \times m \times n$ where d denotes the spatial width/height of the kernel, the parameter number of this layer is d^2mn and the computational cost is d^2mnMN. Depthwise separable convolution splits the standard convolutional layer into two layers, where each filter in the depthwise convolution only acts on one input channel, so the number of output channels of this layer is equal to that of input channels m, and the number of parameters and calculations of this layer are d^2m and d^2mMN, respectively. Pointwise convolution is a typical 1×1 convolution, which is responsible for integrating channel information and outputting the feature maps with the target number of channels n. The number of parameters and calculations of this layer are mn and $mnMN$, respectively. Therefore, the overall number of parameters of a depthwise separable convolution is $d^2m + mn$ and that of the computational cost is $d^2mMN + mnMN$. The reduction in parameter size can be calculated as

$$\text{reduction} = \frac{d^2m + mn}{d^2mn} = \frac{1}{n} + \frac{1}{d^2} \tag{1}$$

The chain-styled architecture of MobileNet in part limits the performance of the model. To this end, an improved version MobileNetV2 [27] was then proposed to further decrease the number of operations and improve the feature learning ability of the mobile models. Compared with MobileNet, the main contribution of MobileNetV2 lies in the linear bottleneck and inverted residual block. The inverted residual block, as shown in Fig. 1, is different from ResNet block [32] in shape. The input feature maps are first fed into the 1×1 expansion layer with a factor of 6 to increase the number of channels, and then pass through the depthwise convolutional layer, followed by the 1×1 projection layer to restore the original number of channels. It is worth noting that a linear bottleneck, which is to remove the non-linearity in the projection layer, is introduced to capture the manifold of interest in the low-dimensional space. At the end, the input is added to the output by a residual connection to form the final output feature map.

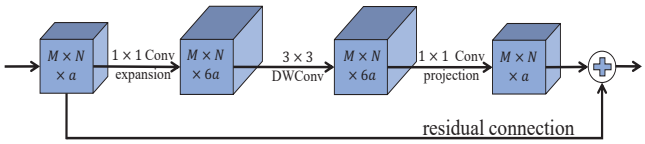


Fig. 1. An example of an inverted residual block in MobileNetV2.

The latest version MobileNetV3 [33] improved this MBConv architecture. An NAS method - MnasNet [34] is adopted to search for a rough MobileNet architecture which is then finetuned to obtain a refined architecture. Meanwhile, the squeeze-and-excitation (SE) block [35] is incorporated to effectively capture the channel relationship. MobileNetV3 significantly improves the model performance and alleviates the computational burden through an efficient new activation function *h-swish*. This work will further improve this efficient MBConv block in an automatic and flexible way to get a more effective and compact CNN architecture.

B. EC-based NAS

Owing to the flexible encoding and strong search capability, EC-based NAS plays a critical role in the field of automated design of CNN architecture [7].

GeNet [36] is an early EC-based NAS method. It divides the searched architecture into multiple stages, each of which has different number of nodes connected with the downsampling (pooling) operation. In each stage, all convolutional layers are with the same predefined settings. A binary string encoding strategy is adopted to represent the inter-node connections. The evolving process follows the typical operations of a GA, namely, initialization, evaluation, selection, crossover and mutation. To further reduce the search space, AmoebaNet [10] proposed to search for the optimal CNN architecture based on the "NASNet search space" [9] which predefines a bi-chain-styled backbone and designs so-called transferable cells. The evolved AmoebaNet-A performs slightly better than the NASNet-A baseline [9]. When being transferred to ImageNet [30], AmoebaNet achieved a new state-of-the-art performance that surpasses those models crafted by hand. However, the computational complexity is still very high, which is 3,150 GPU days. More recently, a PSO-based method was proposed for DenseNet style transferable block evolution [22]. The number of layers and the growth rate of the dense block are encoded in a fixed-length fashion and the evolved block is

This article has been accepted for publication in IEEE Transactions on Evolutionary Computation. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2022.3217290

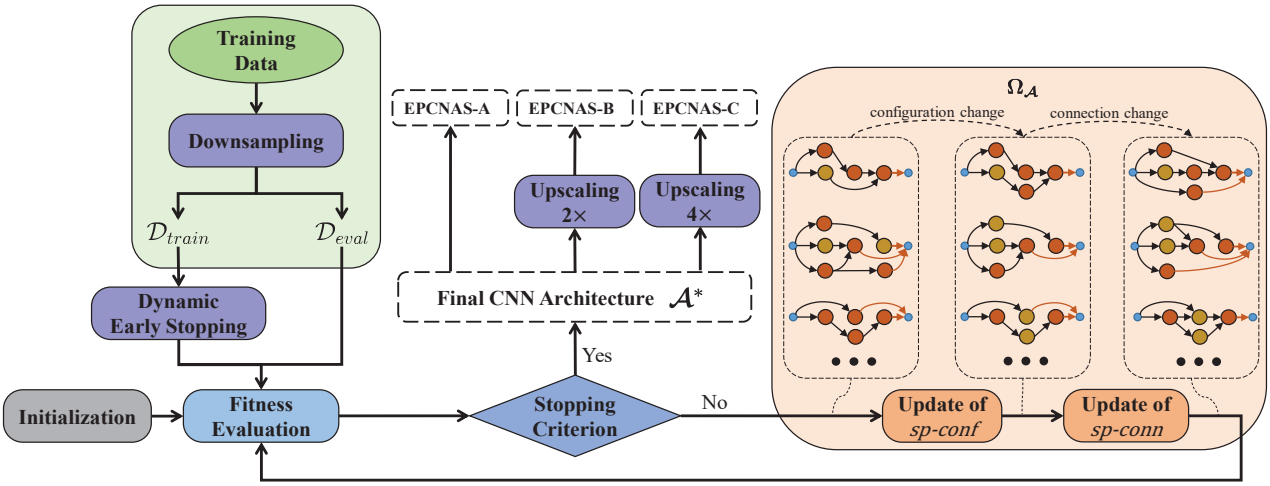


Fig. 2. Framework of the proposed EPCNAS algorithm. A particle in EPCNAS is composed of two subparticles, one subparticle for network configuration (*sp-conf*) and the other subparticle for network connection (*sp-conn*).

stacked repeatedly to construct the final CNN architecture. To reduce the search cost, this method, named EffPNet, used an SVM-based binary classifier as the surrogate model to predict the performance comparison result between the new particle and its local best solution based on the architectural parameters, the losses and accuracies of the early training epochs. Similar to most NAS algorithms, the fixed macro-architecture deprives the resulting architecture of some diversity. These have motivated us to develop a flexible two-level PSO algorithm to evolve CNN architectures with macro-level and micro-level components simultaneously with high efficiency.

C. Automatic Compact CNN Architecture Design

Regarding the compact CNN architecture design, Elsken *et al.* [37] proposed a Lamarckian evolutionary algorithm for multi-objective NAS, named LEMONADE, which conducts the joint optimization of predictive performance and resource constraints. Lu *et al.* [38] proposed NSGA-Net, which formulates the NAS task as a multi-objective problem aiming to minimize two objectives: classification error and the number of floating-point operations (FLOPs). Mnasnet [34] incorporated model latency into the objective function of NAS, and explored a hierarchical search space constructed by MBConv blocks, resulting in very efficient MobileNet-like architectures. [39] used the same MobileNet template-based search space and relaxed the objective function containing complexity constraint to be differentiable to improve search efficiency. Cui *et al.* [40] proposed to search for efficient networks under the constraint of computational complexity less than 300M FLOPs. A search space consists of the MBConv blocks with different expansion ratios and connecting ways is constructed and the network architecture is searched by a bi-level optimization. [41] measured the architecture complexity by the number of blocks in the architecture and the number of nodes per block, and also used a bi-level evolutionary algorithm to solve the compact CNN architecture design. Recently, a GA-based NAS method [42] is proposed to designing CNN architecture under the preset constraint of number of parameters. A modified MBConv block is incorporated into search space to facilitate lightweight CNN design. It is not difficult to see from the above literature that lightweight MBConv blocks are widely used in designing compact architecture, which is the case in this work. However, many other methods explicitly involve strong constraints which might affect the flexibility of the search algorithms. In this work, we focus on the optimization of classification performance and number of parameters. Differently, a search space of more lightweight architectures is constructed, consisting of the modified parameter-efficient MBconv blocks with arbitrary sparse connections and fewer filters. The proposed method can navigate such a search space more flexibly to obtain compact architectures in an implicit fashion without specifically imposing complexity constraints on architecture search.

III. Proposed Algorithm

In this section, the proposed EPCNAS algorithm for compact CNN architecture design is presented in detail. To be specific, we first outline the overall framework of the algorithm in Subsection III-A, and then introduce the search space design and the two-level encoding strategy of EPCNAS in Subsections III-B and III-C, respectively. Last, the main steps of EPCNAS, i.e., fitness evaluation and particle updating, are illustrated in Subsections III-D and III-E, respectively.

A. Overall Framework

The goal of NAS is to automatically explore the optimal network architecture $\boldsymbol{\mathcal{A}}^*$ in a predefined search space $\boldsymbol{\Omega}_{\mathcal{A}}$, which, after training, can achieve the smallest loss on the evaluation set $\mathcal{D}_{eval}$. This process can be formulated as

$$\begin{cases} \boldsymbol{\mathcal{A}}^* = \arg\min_{\boldsymbol{\mathcal{A}} \in \boldsymbol{\Omega}_{\mathcal{A}}} \mathcal{L}(\boldsymbol{\mathcal{A}}, \boldsymbol{w}^*_{\boldsymbol{\mathcal{A}}}, \mathcal{D}_{eval}) \\ \text{s.t.} \quad \boldsymbol{w}^*_{\boldsymbol{\mathcal{A}}} = \arg\min_{\boldsymbol{w}_{\boldsymbol{\mathcal{A}}}} \mathcal{L}(\boldsymbol{\mathcal{A}}, \boldsymbol{w}_{\boldsymbol{\mathcal{A}}}, \mathcal{D}_{train}) \end{cases} \tag{2}$$

where $\mathcal{L}(\boldsymbol{\mathcal{A}}, \boldsymbol{w}_{\boldsymbol{\mathcal{A}}}, \mathcal{D})$ measures the performance of the network architecture $\boldsymbol{\mathcal{A}}$ with weights $\boldsymbol{w}_{\boldsymbol{\mathcal{A}}}$ on the data $\mathcal{D}$, and $\boldsymbol{w}^*_{\boldsymbol{\mathcal{A}}}$ denotes the weights of the optimal network architecture $\boldsymbol{\mathcal{A}}^*$ that achieves the best performance on the training data $\mathcal{D}_{train}$. In this work, $\boldsymbol{\Omega}_{\mathcal{A}}$ is designed as a search space which

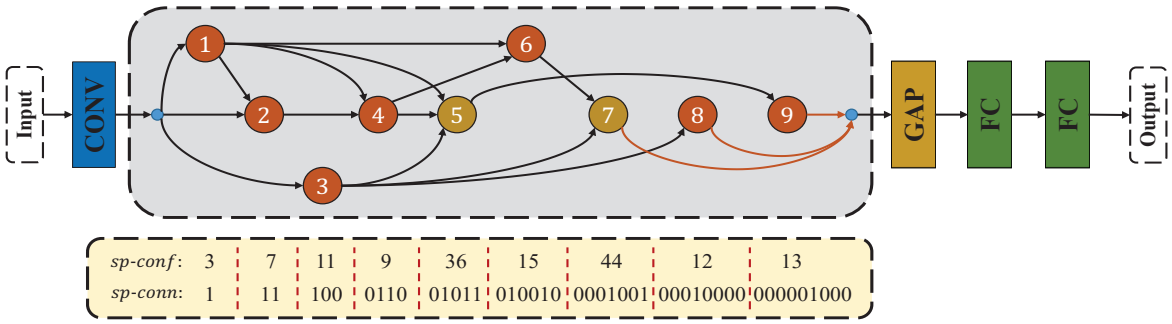


Fig. 3. An example of the CNN architecture evolved by EPCNAS. GAP means global average pooling layer and FC represents fully-connected layer. The gray area is the evolvable part, where each node represents a block or layer. Nodes 5 and 7 denote the pooling layer, while the other nodes are the modified MBConv blocks. The yellow box provides the encoded particle corresponding to this architecture which contains the subparticle *sp-conf* encoded with configuration information (an arbitrary example is given here) and the subparticle *sp-conn* encoded with connection information.

is constructed by lightweight MBConv block (elaborated in Subsection III-B). $\mathcal{A}^*$ is searched by a two-level variable-length PSO at the upper level optimization based on its corresponding weights $\boldsymbol{w}^*_{\mathcal{A}}$ which is learned at the lower level optimization. The overall framework of EPCNAS basically follows the typical EC-based NAS algorithm flow [7], and is outlined in Fig. 2.

The proposed method starts with individuals randomly produced, collectively known as a swarm or population. Due to the extremely large search space, the initial population should have a large diversity to benefit the evolving process. The following evolving process is the core part of the algorithm, which is a loop of fitness evaluation and a two-level PSO update. The particle representing the CNN architecture is composed of two parts: the micro-level and the macro-level. The micro-level subparticle that encodes the configuration of each layer, referred to as *sp-conf*, is updated by a variable-length continuous PSO evolution scheme. The other subparticle that encodes the connections between layers, dubbed *sp-conn*, is evolved by a binary PSO (BPSO) [43] updating approach.

In the proposed method, a reduced training dataset is generated by downsampling the original training set, which is then divided into two parts for architecture search and evaluation. At each iteration in the evolutionary loop, the *sp-confs* in the swarm are updated frist, followed by the update of *sp-conns*, and each of the newly evolved candidate CNN architectures is trained on training part $\mathcal{D}_{train}$ of the reduced training set by the mini-batch stochastic gradient descent (SGD) using a well-designed early stopping policy, and evaluated on the fitness evaluation part $\mathcal{D}_{eval}$ of the reduced training set to obtain its fitness value, e.g., classification accuracy. Finally, once the stopping criterion (say, β iterations of architecture search) is met, the best CNN architecture $\mathcal{A}^*$ (also known as EPCNAS-A) is outputted and fully trained and tested on the training set and the test set, respectively, together with its two upscaled variants EPCNAS-B and EPCNAS-C (by doubling and quadrupling the filter number of convolutional layers), to obtain their final performance in the task.

B. Search Space Design

A typical CNN architecture[3] for classification is composed of three types of layers: convolutional layer, pooling layer and fully-connected layer [44]. Multiple convolutional and pooling layers are stacked to effectively extract features from the input data, while the fully-connected layers are usually placed at the tail of the network to achieve classification. In this work, we will evolve the entire convolution part (i.e. feature extractor network) in CNN for image classification. An example of the evolved CNN architecture is depicted in Fig. 3. The target CNN architecture starts with a standard convolutional layer and ends by a global average pooling (GAP) layer plus two fully-connected layers. The gray box in the middle is an evolvable network with 9 computational nodes, where each node represents a computational block/layer with different configurations, i.e. pooling layers for nodes 5 and 7, and modified MBConv blocks for the others. The MBConv module [27], [33] is a lightweight and high-performance computing paradigm compared to ordinary convolution modules. The connection structure between nodes is a mesh structure with a flexible encoding. Theoretically, such a connection encoding can lead to many popular network architectures, such as ResNet-like [28] and DenseNet-like [28] architectures, and the search space also encompasses more untapped novel architectures. The input of each node can be a concatenation of the output of any previous nodes, and those nodes that have no output connection will be concatenated together as the final output of the evolved feature extractor (which is the input of the classifier as well). For example, in the CNN in Fig. 3, the concatenation of the outputs of nodes 7, 8 and 9 will be the input of the classification part of this network.

The target CNN architectures are expected to be compact, that is, having a small number of parameters. To this end, we construct a valuable search space using the modified parameter-efficient MBConv blocks as building blocks, making compact architecture search easier. The architecture of the modified MBConv block is shown in Fig. 4. Compared with the counterpart in MobileNetV3, the shortcut connection of element-wise feature addition is removed, implying that the modified MBConv block allows the number of input and output feature maps to be inconsistent. Moreover, the SE block

[3]In this subsection we focus on describing the target architecture.

[35] is replaced by the Efficient Channel Attention (ECA) module [45] which captures cross-channel interaction in a more efficient fashion by performing a fast $1D$ convolution with adaptive kernel size. The block takes a input channels that pass through an 1×1 convolutional layer and a $d \times d$ depthwise convolutional layer to obtain kb feature maps which are then fed into an ECA module to learns channel attention. Finally, b feature maps will be outputted through another 1×1 convolutional layer. Generally speaking, such a MBConv block contains several configurations, i.e. number of filters (or output feature maps) b, kernel size of DWConv layer d, and expansion ratio k. Since there are not many typical choices for the latter two (typically 3 and 5 for d, and 3, 4 and 6 for k), and adjusting the number of filters is a more effective way to exploring flexible architecture, the searchable architecture parameter of the modified MBConv block here is b only, i.e., number of feature maps.

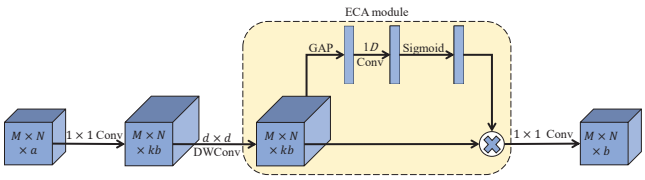


Fig. 4. The modified MBConv block with a input channels and b output channels. DWConv denotes the depthwise convolution operation. Compared with MobileNetV3, the residual connection is removed, and the SE module is replaced by the ECA module.

The search space design has the following characteristics. First of all, the search space allows the proposed algorithm to search for network architectures with varying numbers of nodes that indicate the depth of the final model, and the number of filters of each MBConv block reflecting the network width. Furthermore, the numbers, types, and positions of the pooling layers that determine the spatial resolution change of the architecture are encoded in the search space. Last but not least, the nodes are arranged in order in the target architecture, each of which can be connected to its previous nodes arbitrarily. This provides greater possibilities for flexible and novel architecture design. In a nutshell, the proposed search space allows EPCNAS to simultaneously evolve the depth, width, spatial resolution changes, and connection ways of the CNN architecture.

C. Two-level Encoding Strategy

As discussed earlier, all the blocks or layers are regarded as computational nodes, and the searchable architectural parameters involved in this work include: (1) network depth (determined by the number of evolved nodes), (2) configurations of nodes (number of filters for MBConv block, and pooling type for pooling layer), (3) spatial resolution changes (number and positions of pooling layers), and (4) connections between nodes. Obviously, the intra-node configuration information is not at the same level as the inter-node connection information. In order to search for these architectural parameters effectively, a two-level PSO method is proposed herein for the architecture evolving process. To be specific, a particle representing a complete CNN architecture is composed of two subparticles, i.e. *sp-conf* for network configuration and *sp-conn* for network connection. Moreover, a micro-level updating approach is devised to evolve architectural parameters in (1)-(3) encoded by *sp-conf* while the other macro-level updating approach is responsible for evolving architectural parameter (4) encoded by *sp-conn*. The division of labor and cooperation between the two updating approaches dramatically reduces the encoding complexity without incurring much extra computational overhead. The particle encoding strategies of the two subparticles are described respectively below.

1) Encoding of sp-conf: The encoding of *sp-conf* needs to include the depth information of the network and configuration information of each layer in the network, so it should be a variable-length representation and each dimension of the particle encodes the related configurations of the corresponding layer of the network. Here, we adopt a flexible variable-length particle encoding strategy proposed in our previous work FPSO [46] for the encoding of *sp-conf*, which perfectly meets the aforementioned encoding requirements with the same encoding parameters. There are two major differences to be aware of. First, each dimension of the particle in FPSO represents an exact standard network layer, while that of *sp-conf* represents a computational node, either an MBConv block or a pooling layer. The second difference lies in the range of the number of output feature maps of MBConv block set to [1, 16], which is much smaller than that of the convolutional layer in FPSO. This is because the final architecture in this work contains many shortcut connections for feature map concatenations, and a downscaling operation is applied to the intermediate architecture search stage to reduce the computational complexity.

A computational node can be represented by six bits in binary form where the first bit is to distinguish the node type, i.e. 0 for MBConv block and 1 for pooling layer. The second bit is purposefully added to expand the search space to achieve subsequent exploration of different particle lengths, i.e. the depth of the network architecture. We also call this bit the operation bit, because once it becomes 1 after the standard PSO updating process, the dimension will be further adjusted in length through a split operation. Fig. 5(a) illustrates the 6-bit representations of the normal[4] MBConv block and pooling layer in binary form and Fig. 5(b) shows the split operation. For a normal MBConv block, the last four bits of its binary representation carry the information of the output feature map number of the block, while the pooling type of a normal pooling layer which has only two options - maximal and average - can be represented by one bit. Based on this encoding, in decimal form, the encoding ranges of MBConv block and pooling layer are [0, 15] and [32, 47], respectively. Note that the value 15 means 16 output feature maps since the binary string starts from 0. Fig. 3 gives an example of *sp-conf* and Fig. 5(c) shows the corresponding binary form of its two dimensions.

Ignoring the factor of length, the update of *sp-conf* is essentially a standard continuous PSO updating approach.

[4]"normal" means the operation bit of this dimension is 0 and no further split operation is required.

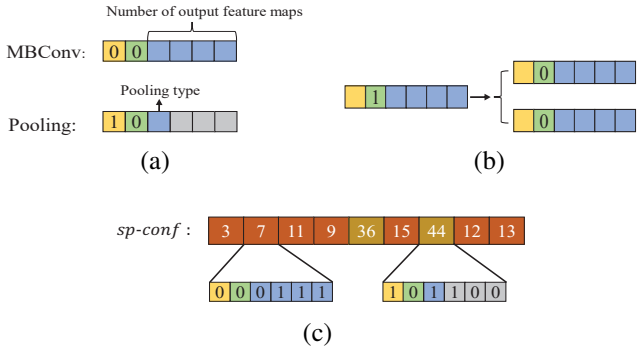


Fig. 5. Encoding strategy of *sp-conf*. (a) Representation of one node in binary form. (b) Split operation performed when the second bit of a dimension value in binary form is 1. (c) An example of an *sp-conf*.

According to the proposed encoding strategy, all dimensions of the evolved *sp-confs* represent normal MBConv blocks or pooling layers. During the standard PSO updating, an updated dimension will be discarded if it is smaller than 0. Otherwise if the operation bit of its binary representation string turns into 1, the dimension will be further divided into two dimensions (as shown in Fig. 5(b)), which is also equivalent to splitting the corresponding node into two nodes representing two normal blocks/layers. Please refer to Table I for the encoding ranges corresponding to different node types and operations, as well as the adjusted parameter(s), given a dimension value s of a standard particle updating. Particularly, if the dimension value falls into the ranges [16, 31] or [48, $+\infty$), the node will be split into two nodes of the same type. Note that the two "Split pooling layer" operations are different in parameter adjustment. The two resulting pooling layers of the former operation have different pooling types, while the latter are the same. The encoding strategy of *sp-conf* provides a reasonable meaning to every dimension value in the entire value space, which greatly facilitates the flexible updating of *sp-conf*.

TABLE I
RANGES AND PARAMETERS OF DIFFERENT NODE TYPES AND OPERATIONS

Range of s	Node type/Operation	Adjusted Parameter(s)
($-\infty$, -1]	Remove the node	-
[0, 15]	MBConv block	s
[16, 31]	Split MBConv block	$\lfloor \frac{s}{2} \rfloor$, $\lceil \frac{s}{2} \rceil$
[32, 47]	Pooling layer	s
[48, 63]	Split pooling layer	39 + $\lceil \frac{s-47}{2} \rceil$, 40 - $\lfloor \frac{s-47}{2} \rfloor$
[64, $+\infty$)	Split pooling layer	$\lfloor \frac{s}{2} \rfloor$, $\lceil \frac{s}{2} \rceil$

2) Encoding of sp-conn: With regard to the encoding of *sp-conn* which carries the connection information between nodes, it is inspired by GeNet [36]. A big improvement is that the encoding flexibility is greatly enhanced. We evolve the entire feature extraction network with pooling layers involved which is more general and flexible, whereas GeNet only searches the convolutional networks within different stages connected with pooling layers. See an example of the schematic of the proposed connection scheme in Fig. 3. The output of the first standard convolutional layer is the input of the evolved architecture, referred to as input node. The computational nodes are arranged in order, and for the i-th node, an i-bit binary string is used to encode its connection relationship with the input node and the previous $i-1$ computational node(s). For example, we use 4 bits to represent the connection between the input node, nodes 1, 2, 3 and node 4. It can be seen from Fig. 3 that node 4 is connected to nodes 1 and 2, and disconnected from the input node and node 3, so the connection code for node 4 is 0110. The final *sp-conn* is obtained by combining the connection codes of all computational nodes in the evolved network. An *sp-conn* is a fairly long binary string whose length depends on the depth of the evolved network architecture. Given that the number of computational nodes of a network architecture is num, the length of the corresponding *sp-conn* is $num(num+1)/2$. Apparently, this is also a variable-length representation, and the proposed BPSO-based approach will be leveraged to effectively handle the updating of the variable-length *sp-conn*.

D. Fitness Evaluation

During the iterative network architecture search phase, each newly updated particle needs to be evaluated to obtain its fitness value. A typical fitness evaluation involves fully training the decoded CNN architecture on the training set $\mathcal{D}_{train}$ (say, e epochs of gradient training), and then examining its performance on the fitness evaluation set $\mathcal{D}_{eval}$. It is not difficult to see that this process occupies the bulk of computational complexity of the NAS algorithm. In order to ease the computational burden of architecture search and evaluation, we propose an effective acceleration scheme by integrating multiple computationally reduced methods, which are early stopping policy, image downsampling and architecture downscaling. The pseudo-code of the fitness evaluation is shown in Algorithm 1.

The standard early stopping policy is to use a small number of epochs instead of e epochs to train the candidate architecture, and directly use its evaluated performance as the fitness value. Nevertheless, the complexities of the searched architectures are diverse. We hope to train more epochs for the potentially good architectures and abandon the bad architectures as early as possible. To this end, an improved early stopping policy is developed by changing the number of epochs in early stopping training to a dynamic number. Given a small number ε ($\varepsilon \ll e$), the learning rate is scheduled to decay within ε epochs during the individual architecture training process by SGD. Small fluctuations in the error curve during training are allowed. As shown in Lines 11 to 17 in Algorithm 1, the architecture training will be terminated if the current epoch is more than λ ($\lambda < \varepsilon$) epochs from the epoch corresponding to the best-so-far evaluated accuracy, or if the evaluated accuracy of the architecture obtained by the current epoch is lower than the best-so-far one by more than a certain level ξ, say, 3%, because the training of the architecture is considered unstable and has low potential in the competition for the best architecture. Otherwise, the architecture will continue to train until the predefined maximal epochs. Under this method, most poor architectures usually stop training after less than ε epochs, while training for better architectures often exceeds ε epochs. As a rough estimate, the proposed dynamic early stopping method makes the computational cost of the proposed algorithm ε/e of the original cost while greatly improves the efficiency of architecture training compared to the standard early stopping method.

The two low-fidelity methods - image downsampling and architecture downscaling - speed up the architecture evaluation process by reducing the amount of calculation in each epoch of training. The original dataset is downsampled to a smaller size in the architecture search process, such as resizing the 32×32 image of CIFAR-10 [29] to a size of 16×16, so that the memory and computation required for training and evaluation of the candidate networks can be greatly decreased on this reduced dataset. Similarly, we downscale the network architecture for the search stage, that is, compress the search range of the number of the MBConv block's filters to 25%[5], which is between [1, 16]. In this way, the searched architecture will be very small in size, which can also significantly improve the efficiency of fitness evaluation. Assuming that the downsampling ratio and the downscaling ratio are γ and η, respectively, then by combining these three computationally efficient settings, the total computational cost of the search process can be reduced to $\gamma^2\eta^2\varepsilon/e$ of the original. Fig. 2 depicts how these three techniques work in the proposed algorithm. Note that in the final training phase, the resulting architecture will be scaled up in two levels by multiplying the width of each layer by two and four, respectively, and a total of three architectures are finally outputted.

Algorithm 1: Fitness Evaluation of EPCNAS

Input: The population P, the training set, the number of training epochs ε, accuracy reduction threshold ξ, the maximal distance λ to the best-so-far epoch.
Output: The population P with fitness.

$\mathcal{D}_{train}$, $\mathcal{D}_{val}$ ← Downsample and split the training set;
$Optimizer$ ← Create a gradient optimizer with ε epochs of cosine annealing learning rate scheduler;
for *each individual* p *in* P **do**
  $epoch \leftarrow 1$;
  $epoch_{best} \leftarrow 1$;
  $acc_{best} \leftarrow 0$;
  $toTerminate \leftarrow False$;
  while *not* $toTerminate$ **do**
    Train p on $\mathcal{D}_{train}$ by $Optimizer$ for one epoch;
    acc ← Evaluate p on $\mathcal{D}_{val}$;
    if $epoch > epoch_{best} + \lambda$ *or* $acc < acc_{best} - \xi$ **then**
      $toTerminate \leftarrow True$;
    end
    if $acc > acc_{best}$ **then**
      $acc_{best} \leftarrow acc$;
      $epoch_{best} \leftarrow epoch$;
    end
    $epoch \leftarrow epoch + 1$;
  end
  Assign acc_{best} as the fitness of p in P;
end
Return P.

E. Particle Updating

A complete CNN architecture is jointly represented by a particle including an *sp-conf* and an *sp-conn*. Hence, the evolution of the network architecture involves the joint update of these two subparticles, which is completed by a two-level PSO algorithm. As mentioned earlier, the two-level PSO method includes the update for evolving configurations (i.e., *sp-conf*) and the update for evolving network topology (i.e., *sp-conn*), which are conducted sequentially in one iteration followed by a fitness evaluation. As shown in the framework of Fig. 2, the update of *sp-conf* is performed first at each iteration of the evolutionary process, and based on the evolved configuration information, the connection topology *sp-conn* is then updated. Each dimension of *sp-conf* is a decimal integer, so we can use the standard continuous PSO followed by a clipping (round down) operation to calculate the subparticle's velocity and position. *sp-conn*, unlike *sp-conf*, is a binary string. Consequently, after obtaining the continuous velocity vector, the subparticle *sp-conn* will be further updated by BP-SO (Equations are provided in the Supplementary Material).

An intractable problem here is that we are unable to directly calculate the velocity vectors of both subparticles in standard form due to their variable-length nature. Therefore, we leverage the particle alignment operation in FPSO [46] to adjust the length of the personal best solution and the global best solution before calculating the velocity vector. The particle alignment operation is illustrated in Fig. 6, which mainly consists of two steps: offset selection and particle length adjustment. Setting a different offset for each calculation can increase the randomness of particle updating and the diversity of the subsequent population. Given the current particle x and the reference particle p, an offset is randomly selected from [0, $\mathrm{abs}(\mathrm{len}(x) - \mathrm{len}(p))$] where $\mathrm{abs}(\cdot)$ means taking the absolute value and $\mathrm{len}(\cdot)$ is the length acquisition operation. Then, based on the selected offset, p is extended (if shorter than x) or truncated (if longer than x) to match the length of x.

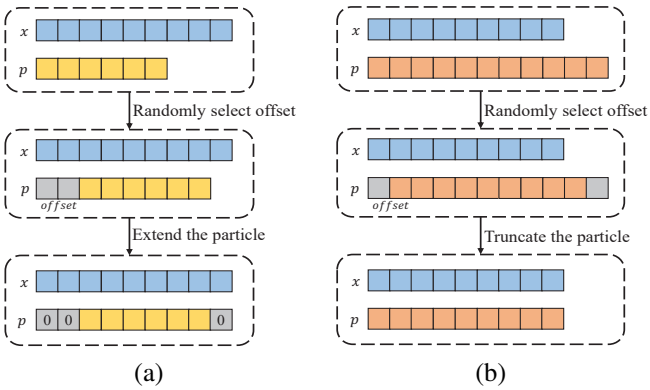


Fig. 6. Two examples of the particle alignment operation. (a) p is shorter than x and the $offset = 2$. (b) p is longer than x and the $offset = 1$.

It is worth noting that the update of *sp-conf* will not only evolve the configuration of the block/layer, but may also bring changes in the length of the network. Posterior to the update of *sp-conf* and prior to the update of *sp-conn*, if there is a reduction or increase in dimension of *sp-conf*, *sp-conn* needs to be adjusted accordingly. See Fig. 7 for details. Fig. 7(a) gives an example of a particle and its corresponding architecture, and Fig. 7(b) shows the particle updating without length change. If a node is removed in the updating process, then the encoding information related to the node needs to be deleted from the corresponding *sp-conn*, as shown in Fig. 7(c). If a node is split into two nodes, then the two nodes will be connected, and the input and output connections of the original node are allocated

[5]According to our designed architecture, it is most appropriate to set the search range of the number of the MBConv block's filters to [1, 64].

to the two new nodes, as shown in Fig. 7(d).

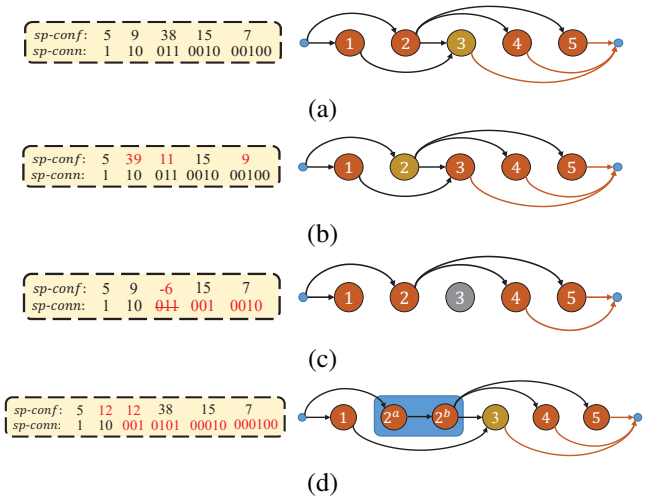


Fig. 7. Examples of update for *sp-conf* and the corresponding changes in connection topology. (a) Particle and the corresponding architecture before *sp-conf* updating. (b) Particle and the corresponding architecture after *sp-conf* updating without changing the length. (c) Particle and the corresponding architecture after *sp-conf* updating where one node is deleted. (d) Particle and the corresponding architecture after *sp-conf* updating where one node is split.

The two updating approaches will jointly and complementarily contribute to the whole architecture search. After the updating process, when decoding and evaluating the new architecture, if a node receives feature maps of different spatial resolutions as input, we will add a standard strided convolution operation to downsample the feature maps with larger resolution to ensure that their sizes for concatenation are identical. This can be caused by both updates, from which it can be seen that the architectural parameter - spatial resolution changes, is actually affected by the two updating processes.

IV. Experiment Design

In this section, we design a series of experiments to demonstrate the efficacy of the proposed EPCNAS algorithm. The three benchmark image classification datasets and the selected peer competitors are discussed in Subsections IV-A and IV-B, respectively. The parameter settings of EPCNAS are detailed in Subsection IV-C.

A. Benchmark Datasets

To quantify the performance of EPCNAS, we adopt three benchmark datasets extensively used in NAS, i.e., the CIFAR-10 [29], CIFAR-100 [29], and ImageNet [30] datasets. These three are also the most used datasets for CNN-based classification tasks in recent years, which in turn have provided many challenging competitors for the performance comparison.

CIFAR-10 consists of 60,000 images including 50,000 for training and 10,000 for testing, each of which is a 32×32 RGB image. These images are evenly divided into 10 different classes, such as airplane, automobile, bird, and cat. The CIFAR-100 dataset is similar to CIFAR-10, except that it has 100 classes, each with 500 training images and 100 test images. More categories and fewer images per category make the classification task on CIFAR-100 more challenging. The above two mentioned datasets are of medium scale. In order to further check the performance of EPCNAS, we will also measure the transferability of the proposed method on ImageNet which is a large-scale dataset with 1,000 classes, and 1.2 million 224×224 color images.

In the experiments, for CIFAR-10 and CIFAR-100, 10% of the training set is extracted as the fitness evaluation set, while the rest is reserved as the training part for architecture search process. When training an architecture, according to the conventions of various peer competitors [11], [28], [32], [47]–[49], the training images of CIFAR-10 and CIFAR-100 will be augmented by zero padding, random cropping, and random horizontal flip.

B. Peer Competitors

In this subsection, a number of state-of-the-art algorithms are included for performance comparison in the experimental part. In particular, we divide the selected peer competitors into three groups according to their degrees of automation in network architecture design. Note that following the convention [47], [50], we do not consider the degree of automation in designing the primitive components (i.e., convolutional operators).

The first group is all comprised of the state-of-the-art compact CNN architectures crafted by hand, including ResNet-110 [32], DenseNet-BC (k = 12) [28], MobileNetV2 [27], ShuffleNetV2 [51] and Ghost-ResNet-56 [52]. Not only are these compact CNN architectures effective in image classification, but also their high-performance modules are widely incorporated into many NAS algorithms.

The second group is comprised of the CNN architectures designed by NAS algorithms, but requires laborious manual assistance in network architecture design, such as artificially predefining macro-architecture or super network. Therefore, this group of algorithms belongs to the "automatic + manual" NAS method. The NAS algorithms in this group are mainly based on RL, GD and EC, some of which focus on compact architecture design, such as LEMONADE [37], NSGA-Net [38], and FPNASNet [40], while other state-of-the-art algorithms are commonly used for comparison, including Block-QNN-S [53], NASNet-A [9], ENAS [13], DARTS [11], NAO [54], SNAS [17], ProxylessNAS [55], GeNet [36], Hierarchical Evolution [56], AmoebaNet-A [10], EffPNet [22], SI-EvoNet-S [49], and PNAS [12].

In contrast, the third group of algorithms as well as the proposed EPCNAS algorithm are fully automatic, which basically do not require rich domain knowledge to assist in architecture search. This group includes NAS V3 [14], MetaQNN [15], LS-Evolution [8], CGP-CNN (ResSet) [57], FPSO [46], AE-CNN [47], and CNN-GA [48]. Note that GD-based NAS algorithms need to manually build a supernet in advance, which involves extensive prior knowledge of CNN, so this group only contains NAS algorithms based on RL and EC.

C. Parameter Settings

The hyperparameters of EPCNAS can be divided into the parameters of the two-level PSO and the hyperparameters of architecture evaluation. The parameter settings of the two-level PSO are listed in Table II, where c_1, c_2 and w are all

specified based on the community conventions [58], [59]. Both the population size α and the number of generations β are set to 20, which is based on previous work [23], [46], [48], and the proposed algorithm is efficient enough to acquire the desired result under these settings. It is worth noticing that to deliberately avoid memory overflow caused by too dense connections, we control the dimension value of *sp-conn* to be less than 64.

TABLE II
HYPERPARAMETER SETTINGS FOR PSO

Parameter	Value
population size α	20
number of generations β	20
acceleration constant c_1	1.49618
acceleration constant c_2	1.49618
inertia weight w	0.7298
maximum dimension value of *sp-conn*	63

The hyperparameters involved in the architecture evaluation have two groups, one in the search phase and the other in the final evaluation phase. Note that in the final evaluation phase, the searched architecture will be expanded at two levels, so this work will finally show the performance of three different scales of architecture. Both phases of architecture training use SGD optimizer and a cosine annealing learning rate scheduler with the initial learning rate of 0.08 and the momentum of 0.9. The weight decay rate is 4e-5. Differently, in the search stage, the integrated acceleration scheme is leveraged to shorten the evaluation time, and the hyperparameter settings are shown in Table III. In particular, the input image size is reduced to 16×16, leading to a 50% downsampling ratio. Correspondingly, in order to avoid generating too small feature maps, the number of pooling layers is limited to no more than two. Compared with the output feature map range [1, 64] of the final largest scale architecture, we reduce this in the search stage to 25% of the original, namely [1, 16]. Under these settings, theoretically we compress the architecture search cost to approximately $50\%^2 \times 25\%^2 \times 20/450 \approx 0.07\%$ of the original cost in total. As discussed before, the expansion ratio k and the kernel size d of the MBConv block have too few alternatives to search. Consequently, based on the empirical insights in some related works [33], [34], [55], respectively, we let k and d change from 3 to 4 to 6, and from 3 to 5 as the number of pooling layers before the block increases. The experiments are implemented based on Python 3.8 and PyTorch 1.7.0, and performed on two GPU cards with the same model - Nvidia RTX A5000. The batch size is set to 256 in the search phase and 64 in the final evaluation phase. To alleviate the model overfitting, we also adopt regularization techniques including cutout [60], Dropout [61] and DropPath [62] when training the architectures.

TABLE III
HYPERPARAMETER SETTINGS FOR ARCHITECTURE EVALUATION

Hyperparameter	Value
number of final training epochs e	450
number of early stopping training epochs ε	20
downsampling ratio γ	50%
downscaling ratio η	25%
maximal distance to the best-so-far epoch λ	3
accuracy reduction threshold ξ	0.03

V. RESULTS AND DISCUSSIONS

In this section, we present our experimental results and analyses. We firstly show the performance of EPCNAS on CIFAR-10 and CIFAR-100 compared with the chosen peer competitors. Then, we visualize the evolutionary trajectory of the proposed algorithm and analyze the convergence of the particles. Furthermore, the reliability analysis is provided by investigating the rank-order correlation between the predicted and ground-truth performance. Finally, the network architectures searched on the CIFAR-10 dataset are transferred to the ImageNet dataset to verify the transferability of EPCNAS.

A. Overall Results

Table IV presents the experimental results of EPCNAS and the chosen peer competitors on the CIFAR-10 and CIFAR-100 datasets in terms of architecture complexity, classification performance, and search cost. Particularly, we use the number of parameters (denoted as #Parameters) to indicate the architecture complexity, and GPU days to measure the search cost. The symbol "–" implies the result was not publicly reported by the corresponding algorithm. The three groups of peer competitors are separated by horizontal lines, and we show the best results of the proposed method at the bottom of the table. The three resulting architectures are dubbed EPCNAS-A, EPCNAS-B, and EPCNAS-C according to the scale from small to large. As shown in Table IV, the parameter sizes of EPCNAS-B and EPCNAS-C are significantly smaller than that of almost all comparison methods, not to mention that EPCNAS-A achieves the least numbers of architecture parameters among them. This largely gives credit to the efficient search space construction. From the perspective of the search cost, we only consume slightly more than 1 GPU day in finding the desired architecture which is greatly lower than all the selected "automatic" peer competitors, ranking fourth among 23 NAS algorithms on CIFAR-10, and second among 12 peer competitors on CIFAR-100. In regard to the classification performance of the evolved architectures, it is shown that the resulting architectures, especially EPCNAS-C, also achieve very competitive performance in all comparison methods even with a small parameter scale and search cost. Conspicuously, the high-performing architectures are obtained among 400 searched architectures in only 20 generations which also demonstrates the efficiency of the proposed two-level PSO search algorithm over other GA [10], [47], [49] or PSO [22], [46] based approaches. For example, EffPNet [22] spent 50 generations in a population of size 30 to search for the best architecture; AmoebaNet [10] found the final architecture from 20,000 candidates; neither of them, however, shows better performance than the proposed approach.

The first group of five manually designed CNN architectures with very small numbers of parameters are listed for comparisons. Note that these methods use well-designed modules to improve the compactness of the architecture, such as Dense block for DenseNet and MBConv block for MobileNetV2. However, they are not effective enough in the design of compact architecture. It is shown that both the number of parameters and classification accuracy of these compact architectures are inferior to EPCNAS-B. Besides, EPCNAS-A and

TABLE IV
PERFORMANCE COMPARISON OF EPCNAS WITH PEER COMPETITORS IN TERMS OF THE CLASSIFICATION ACCURACY (%), NUMBER OF PARAMETERS AND THE SEARCH COST (GPU DAYS) ON CIFAR-10 AND CIFAR-100 DATASETS

Method	#Parameters	CIFAR-10	CIFAR-100	Search Cost	
ResNet-110 [32]	1.7M	93.57	74.84	–	manual
DenseNet-BC ($k = 12$) [28]	0.8M	95.49	77.72	–	manual
MobileNetV2 [27]	2.1M	94.56	77.09	–	manual
ShuffleNetV2 [51]	2.47M	94.17	–	–	manual
Ghost-ResNet-56 [52]	0.43M	92.70	–	–	manual
LEMONADE [37]	0.5M	95.43	–	80	automatic + manual
LEMONADE [37]	1.1M	96.31	–	80	automatic + manual
NSGA-Net (macro) [38]	3.3M	96.15	79.26	8	automatic + manual
FPNASNet [40]	5.76M	96.99	–	0.83	automatic + manual
Block-QNN-S [53]	6.1M	96.70	82.95	90	automatic + manual
NASNet-A [9]	3.3M	97.35	–	2000	automatic + manual
ENAS [13]	4.6M	97.11	–	0.45	automatic + manual
DARTS [11]	3.3M	97.24	–	4	automatic + manual
NAO [54]	10.6M	97.52	–	200	automatic + manual
NAO [54]	10.8M	–	84.33	200	automatic + manual
SNAS [17]	2.8M	97.15	–	1.5	automatic + manual
ProxylessNAS [55]	5.7M	97.92	–	1500	automatic + manual
GeNet [36]	–	92.90	70.97	17	automatic + manual
Hierarchical Evolution [56]	15.7M	96.37	–	300	automatic + manual
AmoebaNet-A [10]	3.2M	96.66	81.07	3150	automatic + manual
EffPNet [22]	2.54M	96.51	81.51	<3	automatic + manual
SI-EvoNet-S [49]	1.84M	97.31	–	0.458	automatic + manual
SI-EvoNet-S [49]	3.32M	–	84.30	0.813	automatic + manual
PNAS [12]	3.2M	96.37	80.47	150	automatic + manual
NAS V3 [14]	7.1M	95.53	–	22400	automatic
MetaQNN [15]	11.2M	93.08	72.86	90	automatic
LS-Evolution [8]	5.4M	94.60	–	2750	automatic
LS-Evolution [8]	40.4M	–	77.00	2750	automatic
CGP-CNN (ResSet) [57]	2.01M	94.99	–	30	automatic
FPSO [46]	0.7M	93.72	–	1.65	automatic
AE-CNN [47]	2.0M	95.70	–	27	automatic
AE-CNN [47]	5.4M	–	79.15	36	automatic
CNN-GA [48]	2.9M	96.78	–	35	automatic
CNN-GA [48]	4.1M	–	79.47	40	automatic
EPCNAS-A	0.12M	95.05	–	1.17	automatic
EPCNAS-B	0.31M	96.27	–	1.17	automatic
EPCNAS-C	1.16M	96.93	–	1.10	automatic
EPCNAS-A	0.15M	–	74.63	1.25	automatic
EPCNAS-B	0.35M	–	79.44	1.13	automatic
EPCNAS-C	1.29M	–	81.67	1.10	automatic

EPCNAS-C outperform them in parameter size and classification performance, respectively, while maintaining excellent performance on another metric.

With human ingenuity, we can design exquisite network architectures and paradigms, but the tuning of some detailed hyperparameters is too energy-consuming. As a remedy, NAS can effectively find a better hyperparameter combination thus outperforming those human-designed architectures. Compared with the second group of algorithms, the accuracy of EPCNAS-C on both CIFAR-10 and CIFAR-100 is very competitive, and the parameter sizes of the architectures with three different scales are significantly smaller than those of these methods. Regardless of the remarkable classification performance achieved by these "automatic + manual" NAS algorithms, there are appreciable human effort in model design behind them. On the other hand, the proposed method can obtain good-performing architectures with a very small number of parameters and search costs without much human assistance. For instance, EPCNAS-C achieves an accuracy of only 0.42% lower than NASNet-A on CIFAR-10, which, however, saves nearly three times the parameters and enormous amount of search time. Compared with LEMONADE which also focuses on compact architecture design, EPCNAS achieves the leading classification performance under the same parameter scale.

It is more fair for the proposed method to compare with the third group of algorithms, since they are all automatic NAS methods. As can be seen from the table, our searched architectures show better performance on at least two metrics, and at time even on all three metrics (e.g. EPCNAS-A *vs.* MetaQNN, EPCNAS-B *vs.* AE-CNN, and EPCNAS-C *vs.* CNN-GA, etc.) on both datasets. Notably, FPSO [46] also spent relatively low GPU days to obtain a fairly compact CNN architecture, but the simple chain-styled backbone may limit its classification performance. In summary, comprehensively considering the architecture complexity, classification performance, and search cost, the efficacy of EPCNAS is very encouraging.

To examine the robustness of the proposed method, we collect the corresponding average results (mean value $\pm$ standard deviation) of 10 independent runs on both datasets, as shown in Table V. The accuracy of the three different scale architectures are very competitive on average, even better than those of some algorithms in Table IV which are generally the best results of the corresponding algorithms. Note that the proposed algorithm has a greater accuracy deviation on CIFAR-100 than CIFAR-10 because the task is more difficult.

In regard to the parameter scale, the resulting architectures on both datasets have very similar parameter quantities, and the deviations are very small as well. These statistics strongly indicate that EPCNAS is robust to search for a compact CNN architecture that performs satisfactorily.

TABLE V
STATISTICS OF 10 INDEPENDENT RUNS OF EPCNAS

Dataset	Scale	#Params	Acc.	GPU days
CIFAR-10	A	0.11M ± 0.02M	94.49 ± 0.34	1.17
	B	0.36M ± 0.06M	96.10 ± 0.19	1.17
	C	1.32M ± 0.23M	96.69 ± 0.14	1.17
CIFAR-100	A	0.11M ± 0.02M	72.56 ± 1.29	1.15
	B	0.37M ± 0.06M	77.72 ± 1.11	1.15
	C	1.30M ± 0.21M	80.61 ± 1.03	1.15

B. Evolution Trajectory

In this subsection, we visualize the evolutionary trajectory of the proposed method on CIFAR-10 to observe the behaviour of the network architecture search, and analyze the convergence of the algorithm under the current parameter settings.

During the evolutionary process, the movement of each particle in the search space is guided by its personal best solution $pBest$ and the global best solution $gBest$. Therefore, we plot the performance distribution of the personal best solutions in the population at each generation in Fig. 8, as well as the changing curve of the global best solution during the evolving process. In the first generation, the $pBest$ solutions of the particles in the population are themselves, which have a large deviation in the classification performance due to random initialization. Starting from the second generation, it can be seen that $pBest$ performance improves rapidly. In the last few generations, $pBest$ of each particle basically converges to a very similar and quite high level, which strongly proved the effectiveness of the proposed evolutionary algorithm.

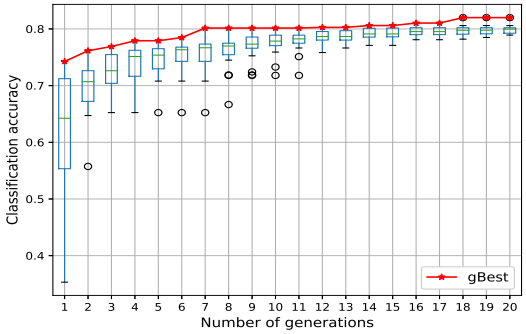


Fig. 8. The evolutionary trajectory of the personal best solutions and the global best solution of EPCNAS on CIFAR-10.

The red solid line represents the change of $gBest$ as the number of generations increases. It is not difficult to see that $gBest$ has a fast improvement in the first few generations, but the rise becomes very slow and small at a later stage, indicating that the population search gradually converges to a relatively stable design within 20 generations. It could be expected that setting a larger population size and the number of generations may allow the algorithm to find a even better $gBest$. Nevertheless, according to the current observation, there may not be a sharp improvement in classification performance, instead additional searches will bring more computational consumption, which is not worthwhile.

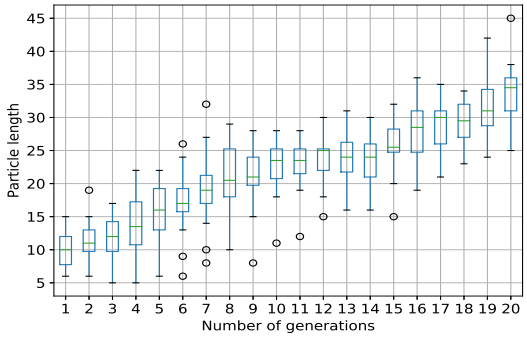


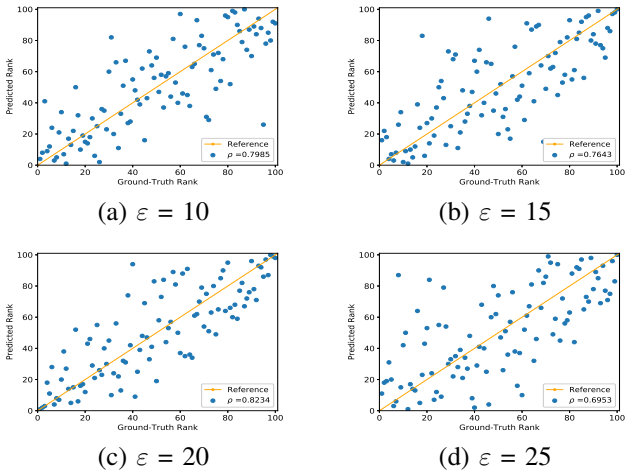
Fig. 9. The evolutionary trajectory of the particle length on CIFAR-10.

The proposed algorithm employs a variable-length encoding strategy, which theoretically can evolve CNN architectures of arbitrary depth. In order to reveal the effectiveness of the proposed algorithm in the search of network depth, the length changes of each particle in the population during the evolutionary process are observed and analyzed. The statistics are shown in Fig. 9. It is shown that during the evolutionary process, the particle length gradually increases, from the initial median value of about 10 to more than 30 in the end. With reference to Fig. 8, it can be concluded that a target network architecture with a number of computational nodes around 30 is sufficiently close to convergence. In addition, it can also be seen from Fig. 9 that the distribution of particle lengths at each generation remains diverse, which ensures the evolution efficiency to some extent.

C. Reliability Analysis

In order to test the reliability of the proposed integrated acceleration scheme in EPCNAS, we measure the Spearman Rank Correlation between the architectures in the reduced settings and ground-truth (by fully training the architectures without downsampling the input). There are three computationally reduced methods adopted in this work, namely early stopping, downsampling and downscaling, among which both the downsampling ratio and downscaling ratio are the lowest values within a reasonable range. Our justifications are 1) if the resolution of the input features is less than 16, the feature information that the architecture can extract is very limited, and the evolution of the pooling layer will also be seriously affected; and 2) by the same token, if the downscaling ratio is less than 25%, the architecture will become too narrow to effectively learn features during architecture search. With these two factors fixed, what we can adjust is the number of training epochs ε for early stopping. Fig. 10 visualizes the rank correlation between the predicted and the ground-truth accuracy of 100 randomly sampled architectures on CIFAR-10 when ε = 10, 15, 20 and 25. Specifically, the Spearman Coefficient ρ is formulated as $\rho = 1 - \frac{6\sum_{i=1}^{K} D_i^2}{K^3 - K}$, where $K = 100$ is the number of architectures, and D_i represents the rank difference of architecture i.

A higher Spearman Coefficient ρ indicates a higher correlation between the predictive performance based on the reduced



(a) ε = 10 (b) ε = 15

(c) ε = 20 (d) ε = 25

Fig. 10. The rank correlation ρ under different number of early stopping training epochs ε. x-axis and y-axis represent the ground-truth rank and the predicted rank, respectively.

settings and the ground-truth performance. As for PSO, since the particle updating is highly dependent on the best solutions, we also need to pay attention to the rank correlation of the top-ranked architectures. As shown in Fig. 10, ρ reaches the highest when ε = 20, and the proposed algorithm is also the most reliable in predicting the performance of the top-ranked architectures, which convincingly demonstrates the reliability of the proposed integrated acceleration scheme in EPCNAS.

D. Large-scale Architecture Transfer

Given the promising results on the CIFAR-10 and CIFAR-100 datasets, we further examine the performance and transferability of the searched architectures on a large-scale dataset, ImageNet [30]. According to the convention [9], [10], [26], [40], we transfer the best architectures searched on the CIFAR-10 to the ImageNet. In particular, we choose two architectures - the best EPCNAS-C architecture, and the best EPCNAS-A architecture but upscale it to the C scale - as the target network architectures, denoted as EPCNAS-C1 and EPCNAS-C2, respectively. In order to quickly reduce the resolution of the ImageNet images to adapt to the architecture used, the stride size of the first standard convolutional layer is set to 2, after which four modified MBConv blocks (two of which have a stride size of 2) are sequentially connected to reduce the image size. Then the entire used architecture is added to form the final network for ImageNet classification. We train the architecture by 300 epochs with an initial learning rate of 0.1 and a batch size of 128. The 0.1 label smoothing and dropout with a probability of 0.3 are also incorporated during training. The performance comparison of EPCNAS with peer competitors is shown in Table VI.

The horizontal lines separate the manual designed models, the state-of-the-art NAS algorithms, as well as the proposed algorithm. In particular, the selected manual designed models are all lightweight architectures which are more suitable for comparison with EPCNAS. Due to the increase in the number of classes, the parameter scale of the classification part increases sharply, so the number of parameters in the transferred architectures on the ImageNet is much larger than

TABLE VI
PERFORMANCE COMPARISON OF EPCNAS WITH PEER COMPETITORS IN TERMS OF THE CLASSIFICATION ACCURACY (%), NUMBER OF PARAMETERS AND SEARCHED GPU DAYS ON IMAGENET DATASET. † THE ARCHITECTURE IS DIRECTLY SEARCHED ON THE IMAGENET.

Method	#Params	Top-1	Top-5	Search Cost
InceptionV1 [63]	6.6M	69.8	89.9	–
ShuffleNetV2 1.5× [51]	3.5M	72.6	–	–
SqueezeNext [64]	3.2M	67.5	88.2	–
GhostNet [52]	5.2M	73.9	91.4	–
MobileNetV1 [31]	4.2M	70.6	89.5	–
MobileNetV2 [27]	3.4M	72.0	91.0	–
MobileNetV3-small [33]	2.9M	67.4	87.7	–
MobileNetV3-large [33]	5.4M	75.2	92.2	–
NASNet-B [9]	5.3M	72.8	91.3	2000
ProxylessNAS† [55]	7.1M	75.1	92.5	8.3
AmoebaNet-B [10]	5.3M	74.0	91.5	3150
DARTS [11]	4.7M	73.3	91.3	4
SNAS [17]	4.3M	72.7	90.8	1.5
FBNet-A† [39]	4.3M	73.0	–	9
NSGANetV1-A1 [26]	3.0M	70.9	90.0	27
FPNASNet [40]	2.95M	72.0	–	0.83
LEMONADE [37]	4 - 6M	71.7	90.4	80
EPCNAS-C1	2.62M	71.4	90.8	1.10
EPCNAS-C2	3.02M	72.9	91.5	1.17

that of the CIFAR datasets. After all, the resulting architectures still have fewer parameters than most peer competitors, while maintaining decent classification performance, especially EPCNAS-C1 has the least number of parameters but reaches an classification accuracy of 71.4%. Furthermore, EPCNAS-C2 owns nearly the same small number of parameters as MobileNetV3-small, NSGANetV1-A1, and FPNASNet, but achieves much better classification performance and is even superior to ShuffleNetV2, NASNet-B, SNAS, and FPNASNet which are much larger in size. Particularly, EPCNAS-C2 also performs better than the compact NAS algorithm, LEMONADE, in terms of parameter scale and classification accuracy. EPCNAS-C2 achieves almost the same classification accuracy as FBNet-A which is conducted directly on the ImageNet dataset, but is more compact in parameter size. This shows that the transferred architecture also has remarkable feature learning ability on the more complex task. Compared with other advanced NAS algorithms, EPCNAS slightly lags behind in classification performance but saves a large proportion of parameters. The architecture derived from CIFAR-10 serves as the main feature extractor of the overall architecture. The above results strongly demonstrate that the architectures searched by EPCNAS from medium-scale datasets have reliable transferability that can be effectively applied to feature learning from large-scale data.

VI. CONCLUSIONS AND FUTURE WORK

This paper proposed EPCNAS, an efficient PSO algorithm for evolving compact CNN architectures for image classification tasks. In order to effectively discover compact CNN architectures, a search space was constructed using the modified parameter-efficient MBConv blocks as building blocks. Furthermore, a flexible two-level PSO algorithm was developed to evolve both the configurations of CNN and connections between nodes in a variable-length manner. The computational complexity is significantly reduced by a well-designed integrated acceleration scheme, which effectively combines early stopping policy, downsampling, and architecture downscaling.

The proposed algorithm is capable of simultaneously and automatically searching the depth, width, spatial resolution changes, and connection ways of the network architecture. The proposed algorithm is examined on three challenging benchmark datasets and compared with various state-of-the-art peer competitors. The experimental results demonstrate that with a very small parameter size and low search cost, EPCNAS is considered superior to almost all the handcrafted architectures and automatic NAS algorithms, and achieves a very competitive performance among the "automatic + manual" peer competitors. Regarding potential limitations, one is that this method cannot precisely control the parameter scale of the generated architecture, which may cause it to fail to deploy on platforms with specified hardware requirements.

With respect to the future work, there are three aspects worthy of further research and improvement. First of all, the proposed algorithm is expected to be applied in designing more different compact neural networks, such as RNNs, attention-based networks, etc. Moreover, the compact NAS task can be formulated as a multi-objective optimization problem, thus better balancing the objectives. Last but not least, the CNN architecture trained on ImageNet is searched through CIFAR-10, which can not be guaranteed to be an optimal architecture on the target task. In the future, how to efficiently perform architecture search on large-scale datasets like ImageNet could be a very meaningful research.

REFERENCES

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

[2] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning (ICML)*, 2019, pp. 6105–6114.

[3] J. Wu, Z. Kuang, L. Wang, W. Zhang, and G. Wu, "Context-aware rcnn: A baseline for action detection in videos," in *European Conference on Computer Vision (ECCV)*, 2020, pp. 440–456.

[4] C. Feichtenhofer, H. Fan, J. Malik, and K. He, "Slowfast networks for video recognition," in *IEEE International Conference on Computer Vision (ICCV)*, 2019, pp. 6202–6211.

[5] Y. Chen, F. Liu, and K. Pei, "Cross-modal matching cnn for autonomous driving sensor data monitoring," in *IEEE International Conference on Computer Vision (ICCV)*, October 2021, pp. 3110–3119.

[6] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, pp. 1–27, 2021.

[7] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE Transactions on Neural Networks and Learning Systems (Early Access)*, pp. 1–21, 2021.

[8] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *International Conference on Machine Learning (ICML)*, 2017, pp. 2902–2911.

[9] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 8697–8710.

[10] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *AAAI Conference on Artificial Intelligence*, 2019, pp. 4780–4789.

[11] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *International Conference on Learning Representations (ICLR)*, 2019.

[12] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *European Conference on Computer Vision (ECCV)*, 2018.

[13] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International Conference on Machine Learning (ICML)*, 2018, pp. 4095–4104.

[14] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2017.

[15] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2017.

[16] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[17] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: stochastic neural architecture search," in *International Conference on Learning Representations (ICLR)*, 2019.

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

[19] A. Darwish, A. E. Hassanien, and S. Das, "A survey of swarm and evolutionary computing approaches for deep learning," *Artificial Intelligence Review*, vol. 53, no. 3, pp. 1767–1812, 2020.

[20] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[21] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "A particle swarm optimization-based flexible convolutional autoencoder for image classification," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 8, pp. 2295–2309, 2019.

[22] B. Wang, B. Xue, and M. Zhang, "Surrogate-assisted particle swarm optimization for evolving variable-length transferable blocks for image classification," *IEEE Transactions on Neural Networks and Learning Systems (Early Access)*, pp. 1–14, 2021.

[23] B. Wang, Y. Sun, B. Xue, and M. Zhang, "Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification," in *IEEE Congress on Evolutionary Computation (CEC)*, 2018, pp. 1–8.

[24] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 394–407, 2020.

[25] M. Suganuma, M. Kobayashi, S. Shirakawa, and T. Nagao, "Evolution of deep convolutional neural networks using cartesian genetic programming," *Evolutionary Computation*, vol. 28, no. 1, pp. 141–163, 2020.

[26] Z. Lu, I. Whalen, Y. Dhebar, K. Deb, E. D. Goodman, W. Banzhaf, and V. N. Boddeti, "Multiobjective evolutionary design of deep convolutional neural networks for image classification," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 2, pp. 277–291, 2020.

[27] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4510–4520.

[28] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261–2269.

[29] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," *University of Toronto*, 2009.

[30] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "ImageNet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[31] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[33] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for MobileNetV3," in *IEEE International Conference on Computer Vision (ICCV)*, 2019, pp. 1314–1324.

[34] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 2820–2828.

[35] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 7132–7141.

[36] L. Xie and A. Yuille, "Genetic CNN," in *IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 1379–1388.

[37] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," in *International Conference on Learning Representations (ICLR)*, 2019.

[38] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "NSGA-Net: neural architecture search using multi-objective genetic algorithm," in *Genetic and Evolutionary Computation Conference (GECCO)*, 2019, pp. 419–427.

[39] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 10 734–10 742.

[40] J. Cui, P. Chen, R. Li, S. Liu, X. Shen, and J. Jia, "Fast and practical neural architecture search," in *IEEE International Conference on Computer Vision (CVPR)*, 2019, pp. 6509–6518.

[41] H. Louati, S. Bechikh, A. Louati, C.-C. Hung, and L. B. Said, "Deep convolutional neural network architecture design as a bi-level optimization problem," *Neurocomputing*, vol. 439, pp. 44–62, 2021.

[42] S. Li, Y. Sun, G. G. Yen, and M. Zhang, "Automatic design of convolutional neural network architectures under resource constraints," *IEEE Transactions on Neural Networks and Learning Systems (Early Access)*, pp. 1–15, 2021.

[43] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, vol. 5, 1997, pp. 4104–4108.

[44] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NeuIPS)*, 2012, pp. 1097–1105.

[45] Q. Wang, B. Wu, P. Zhu, P. Li, W. Zuo, and Q. Hu, "ECA-Net: Efficient channel attention for deep convolutional neural networks." in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 534–11 542.

[46] J. Huang, B. Xue, Y. Sun, and M. Zhang, "A flexible variable-length particle swarm optimization approach to convolutional neural network architecture design," in *IEEE Congress on Evolutionary Computation (CEC)*, 2021, pp. 934–941.

[47] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely automated CNN architecture design based on blocks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 4, pp. 1242–1254, 2019.

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

[49] H. Zhang, Y. Jin, R. Cheng, and K. Hao, "Efficient evolutionary search of attention convolutional networks via sampled training and node inheritance," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 2, pp. 371–385, 2021.

[50] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019.

[51] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in *European Conference on Computer Vision (ECCV)*, 2018, pp. 116–131.

[52] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, "GhostNet: More features from cheap operations," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 1580–1589.

[53] Z. Zhong, Z. Yang, B. Deng, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Blockqnn: Efficient block-wise neural network architecture generation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 7, pp. 2314–2328, 2021.

[54] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," in *Advances in Neural Information Processing Systems (NeuIPS)*, 2018, pp. 7827–7838.

[55] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *International Conference on Learning Representations (ICLR)*, 2019.

[56] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *International Conference on Learning Representations (ICLR)*, 2018.

[57] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Genetic and Evolutionary Computation Conference (GECCO)*, 2017, pp. 497–504.

[58] I. C. Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection," *Information Processing Letters*, vol. 85, no. 6, pp. 317–325, 2003.

[59] D. Bratton and J. Kennedy, "Defining a standard for particle swarm optimization," in *IEEE Swarm Intelligence Symposium*, 2007, pp. 120–127.

[60] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *arXiv preprint arXiv:1708.04552*, 2017.

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

[62] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *European Conference on Computer Vision (ECCV)*, 2016, pp. 646–661.

[63] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[64] A. Gholami, K. Kwon, B. Wu, Z. Tai, X. Yue, P. Jin, S. Zhao, and K. Keutzer, "SqueezeNext: Hardware-aware neural network design," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, 2018.

Junhao Huang (Graduate Student Member, IEEE) received the B.E. and M.E. degrees from Shenzhen University, Shenzhen, China, in 2017 and 2020, respectively. He is currently pursuing the Ph.D. degree in computer science at Victoria University of Wellington, Wellington, New Zealand.

His current research interests include deep learning, evolutionary computation, and neural architecture search.

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

She is currently a Professor of Artificial Intelligence. She has published over 300 research papers. She is currently the Chair of IEEE CIS Evolutionary Computation Technical Committee, and Editor of IEEE CIS Newsletter.

Yanan Sun (Member, IEEE) received the Ph.D. degree from the College of Computer Science at Sichuan University in China in 2017. He is currently a Professor at the College of Computer Science at Sichuan University, China. Before that, he was a postdoctoral Research Fellow at Victoria University of Wellington, New Zealand, in 2017-2019. He is the founding chair of the IEEE CIS Task Force on Evolutionary Deep Learning and Applications. He was selected as "World's Top 2% Scientists" by Sandford University in both 2021 and 2022.

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

He is currently a Professor of Computer Science. He has published over 800 research papers. He is a Fellow of the Royal Society and a Fellow of Engineering New Zealand, a Fellow of IEEE, and an IEEE Distinguished Lecturer.

Gary G. Yen (Fellow, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Notre Dame in 1992. Currently he is a Regents Professor in the School of Electrical and Computer Engineering, Oklahoma State University, USA.

He served as Vice President for the Technical Activities in 2005-2006 and President in 2010-2011 of the IEEE Computation Intelligence Society and is the founding editor-in-chief of the IEEE Computational Intelligence Magazine 2006-2009. He is a fellow of IEEE, IET and IAPR.