Evolutionary Search for Complete Neural Network Architectures With Partial Weight Sharing

Haoyu Zhang, Yaochu Jin, *Fellow, IEEE*, and Kuangrong Hao, *Member, IEEE*

***Abstract*—Neural architecture search (NAS) provides an automatic solution in designing network architectures. Unfortunately, the direct search for complete task-dependent network architectures is laborious since training and evaluating complete neural architectures over a large search space are computationally prohibitive. Recently, one-shot NAS (OSNAS) has attracted great attention in the NAS community because it significantly speeds up the candidate architecture evaluation procedure through weight sharing. However, the full weight sharing training paradigm in OSNAS may result in strong interference across candidate architectures and mislead the architecture search. To alleviate the problem, we propose a partial weight sharing OSNAS framework that directly evolves complete neural network architectures. In particular, we suggest a novel node representation scheme that randomly activates a subset of nodes of the one-shot model in each generation to reduce the weight coupling in the one-shot model. During the evolutionary search, a tailored crossover operator randomly samples the nodes from two parent individuals or a single parent to construct new candidate architectures, thus effectively constraining the degree of weight sharing. Furthermore, we introduce a new mutation operator that replaces the chosen nodes of the one-shot model with randomly generated nodes to enhance the exploratory capability. Finally, we encode a set of pyramidal convolution operations in the search space, enabling the evolved neural networks to capture different levels of details in the images. The proposed method is examined and compared with 26 state-of-the-art algorithms on ten image classification tasks, including CIFAR series, CINIC10, ImageNet, and MedMNIST series. The experimental results demonstrate that the proposed method can computationally much more efficiently find neural architectures that achieve comparable classification accuracy to the state-of-the-art designs.**

***Index Terms*—Convolutional neural networks (CNNs), evolutionary optimization, image classification, one-shot neural architecture search (OSNAS), pyramidal convolution (PyConv) operations.**

I. Introduction

CONVOLUTIONAL neural networks (CNNs) have shown remarkable success in various computer vision applications during the last decade, such as image classification [1], object detection [2], and semantic segmentation [3]. Generally, the performance of CNNs heavily relies on their architectures and the corresponding weights. Only with an optimal architecture and properly trained weights can a CNN achieve the best performance. Most the existing high-performance CNN models, including GoogleNet [4], ResNet [5], and DenseNet [6], however, are manually designed by human experts. A promising neural network architecture usually contains specific configurations in terms of the number of layers, type of each layer, and the hyperparameters related to the used layers. Hence, designing a successful hand-crafted network architecture depends on extensive expertise in both machine learning and the related application area, which is inconvenient to many practitioners.

Neural architecture search (NAS), automatically constructing network architectures, presents a promising approach to address this tedious procedure by considering the design of CNN architectures for the target task as an optimization problem [7]. In recent years, multiple algorithms have been proposed for the automated network architecture design and achieved promising results, which has attracted much industrial and academic interest [8]–[11], and designing task-specific custom networks for various tasks is one of the most practical and challenging problems in automated machine learning.

Most NAS methods rely on reinforcement learning (RL) or evolutionary computation (EC) to design neural network architectures [7], [12]–[15]. Despite their competitive performance on small datasets, such as CIFAR10 and CIFAR100, these methods are usually computationally intensive, requiring to train a tremendous number of models on the given data in a single optimization run. For example, on the CIFAR10 benchmark dataset, the RL-based NAS method [16] finds the best model by consuming 28 days using 800 graphics processing units (GPUs). The EC-based NAS method [17] consumes seven days using 450 GPUs, which obtains the classification accuracy of 96.88%. In fact, the main computation bottleneck of NAS lies in the required frequent full training of numerous

Manuscript received 25 March 2021; revised 28 August 2021 and 15 November 2021; accepted 3 January 2022. Date of publication 6 January 2022; date of current version 3 October 2022. This work was supported in part by the Fundamental Research Funds for the Central Universities and Graduate Student Innovation Fund of Donghua University under Grant CUSF-DH-D-2021047. The work of Yaochu Jin was supported by the Alexander von Humboldt Professorship for Artificial Intelligence endowed by the German Federal Ministry of Education and Research. *(Corresponding author: Yaochu Jin.)*

Haoyu Zhang and Kuangrong Hao are with the Engineering Research Center of Digitized Textile & Apparel Technology, Ministry of Education, College of Information Science and Technology, Donghua University, Shanghai 201620, China (e-mail: zhy920816@sina.cn; krhao@dhu.edu.cn).

Yaochu Jin is with the Engineering Research Center of Digitized Textile & Apparel Technology, Ministry of Education, College of Information Science and Technology, Donghua University, Shanghai 201620, China, and also with the Faculty of Technology, Bielefeld University, 33619 Bielefeld, Germany (e-mail: yaochu.jin@uni-bielefeld.de).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TEVC.2022.3140855.

Digital Object Identifier 10.1109/TEVC.2022.3140855

models from scratch for evaluations. As a result, it is unpractical to apply the NAS method to large-scale tasks, e.g., the ImageNet dataset. To address the major limitation of NAS, increased attention has been paid to improve the computational efficiency with the help of performance prediction [18], [19], weight sharing [20], [21], and weights generation [12], [22], among others.

NAS methods usually use proxy metrics to replace full training. Adopting a performance predictor to assist NAS methods is one of the effective methods that show the competitive performance. Sun *et al.* [19] proposed an end-to-end random forest-based performance predictor (E2EPP) to accelerate the performance evaluation of neural architectures. Specifically, E2EPP constructs a surrogate that can predict the quality of a candidate architecture, thereby avoiding the training of a mass of models during the search procedure. Another trade-off approach is to generate building blocks on proxy metrics, such as training for fewer iterations [23], starting with a small-scale dataset (e.g., CIFAR10), or learning with fewer blocks. Such kinds of optimized block network structures usually own a promising generalization and can transfer between different datasets or task domains [14], [24], [25]. However, these blocks optimized by proxy metrics may not be optimal on the target task. In addition, it may restrict the diversity of blocks by repeatedly stacking single top-performing blocks to construct a complete network architecture, leading to limited performance. Therefore, it is highly desired to allow an algorithm to design a complete neural architecture with different blocks in different parts of the network for an efficient NAS (ENAS) method.

Apart from assessing architecture candidates under proxy metrics, other methodologies, such as weight sharing also called one-shot NAS (OSNAS), have been adopted to speed up the search process for NAS. OSNAS methods not only obtain state-of-the-art architectures but also greatly reduce the search time [20], [25], [26]. Generally, OSNAS views the search space as a one-shot model (also denoted as the supernet), where all network candidates are different submodels of the one-shot model. Any submodels will directly inherit weights from the one-shot model for evaluation, instead of being trained from scratch. Hence, OSNAS only needs to train the one-shot model for architecture search, which reduces the search time from many GPU-days down to several GPU-hours.

While the OSNAS methods can improve the search efficiency, unfortunately, major limitations remain. First, the weights of the nodes (e.g., the convolutional layers) in the one-shot model are dependant on each other and become strongly coupled during the one-shot model training. As a result, Guo *et al.* [21] observed that some submodels have interference with each other in the one-shot model training process. Benyahia *et al.* [27] found that when training numerous submodels via weight sharing based on a single task, the shared weights of previously trained submodels are overwritten during the training of subsequent submodels. They called this multimodel forgetting. To reduce the degree of weight coupling, Bender *et al.* [28] proposed the "path dropout" strategy, which randomly drops some edges of the one-shot model based on a dropout rate during the one-shot model training. This way, the co-adaptation of the node weights of the one-shot model is weakened. However, according to their experiment, the prediction performance of the one-shot model is highly dependent on the dropout rate, making the one-shot model training more complicated. Second, due to the different levels of maturity in the weights of different nodes of the one-shot model, bias may be introduced during the submodel estimation. To be specific, some nodes in the one-shot model are well trained and others are poorly trained. When the submodels share the weights from the one-shot model for evaluation, their performance may be underestimated. Hence, different submodels sampled from the one-shot model are actually noncomparable and the predicted performance of the submodels may be very different from their true performance. Zhang *et al.* [29] observed that this problem can be alleviated by appropriately controlling the degree of weight sharing. They proposed a group weight sharing strategy that divides the submodels into groups. Then, each group is trained independently and shares the weights within the group only. This can be seen as a partial weight sharing strategy rather than fully weight sharing. Their experiments indicate that appropriately choosing the submodels for weight sharing is the key to achieve a reliable predicted rank. However, a smart grouping requires careful design by the user, especially for a complex search space.

This work aims to overcome the aforementioned limitations by taking advantage of the working mechanisms of evolutionary algorithms. Compared to our previous work [30], the present work proposes a new evolutionary OSNAS (Evo-OSNAS) framework with improved encoding, a more flexible representation of deep neural networks, and dedicated crossover and mutation operators. Moreover, we introduce additional advanced multiscale convolution operations into the search space to improve the performance of the proposed algorithm. The contributions of this work are summarized as follows.

1) A novel node representation scheme is proposed for generating candidate submodels from the one-shot model. Specifically, the proposed neural architecture representation consists of a structure chromosome and switch chromosome. In each generation, we randomly switch on a subset of the nodes using the switch genes in generating candidate submodels and evaluating their performance based on weight sharing. As a result, the scheme not only reduces the memory requirement and computation cost but also weakens the weight coupling in the OSNAS approach by regularizing the one-shot model training.
2) A partial weight sharing method is proposed based on a tailored crossover operator to explore the search space in finding promising neural network architectures. The proposed crossover operator is composed of a node crossover and a switch crossover. An offspring network can be constructed by randomly sampling nodes from two parent networks through the node crossover, or randomly sampling a subset of nodes of a parent network through the switch crossover. This way, the degree of weight sharing is constrained, further reducing the

negative impacts of weight coupling in model training in OSNAS, and therefore, more diverse submodels can be found.

3) A set of pyramidal convolution (PyConv) operations is incorporated into NAS. As a result, different types of convolutional kernels can be generated in NAS, enabling the submodels to capture more reliable detailed information on both small and large objects and improve their feature processing capability.

On the basis of the above three main contributions, a computationally efficient and effective Evo-OSNAS algorithm is proposed to search for networks with diverse block structures in the architecture. Evo-OSNAS is able to directly design task-specific complete network architectures without any proxy metrics, making it possible to search for optimal neural architectures on huge datasets such as ImageNet. As a result, optimal networks obtained by Evo-OSNAS can achieve more competitive performance than the state-of-the-art networks found by existing NAS algorithms. The competitive performance of Evo-OSNAS is further confirmed on the MedMNIST series datasets by showing that it can find networks that outperform manually designed models and those designed by existing AutoML methods.

The remainder of this article is organized as follows. Section II introduces the background of this work. Section III describes the architecture search space and PyConv operations, followed by the details of Evo-OSNAS in Section IV. Experimental settings and results are presented in Sections V and VI, respectively. Finally, conclusions are drawn in Section VII.

II. Background

As mentioned before, although the research on NAS methods has made remarkable progress, several grand challenges remain, in particular, the prohibitive computational resources required for performing exploratory NAS. In the following, we provide a brief overview of the recent advances in NAS, focusing on the main ideas of OSNAS methods, which are most relevant to the present work. Finally, the basics of multiscale representation learning will be introduced.

A. Neural Architecture Search

The goal of NAS is to design CNN architectures for a target dataset $D = \{D_{\text{train}}, D_{\text{valid}}, D_{\text{test}}\}$ without the involvement of human experts. NAS can be viewed as a bilevel optimization problem and mathematically formulated as follows:

$$W_A^* = \arg\min_{A \in S} \mathcal{L}_{\text{train}}(N(A, W_A)) \tag{1}$$

$$A^* = \arg\min_{W} \mathcal{L}_{\text{valid}}\left(N\left(A, W_A^*\right)\right). \tag{2}$$

Generally, the search space of NAS S can be represented using a directed acyclic graph (DAG) consisting of an ordered sequence of M nodes, and the architecture of a CNN A can be seen as the subgraph of the DAG. The nodes stand for a layer in CNNs (e.g., a convolutional layer) and the edges stand for the flow of information (e.g., a feature map in CNNs). A CNN can be defined as $N(A, W_A^*)$, where W_A^* defines the weights associated to the architecture A. NAS aims to find the architecture with the best performance on the validation dataset D_{valid} according to (2), and the weights W_A^* can be obtained by training the neural model A on the training dataset D_{train} via minimizing the loss function $\mathcal{L}_{\text{train}}$ according to (1). The testing dataset D_{test} is used for testing the performance of final CNN architecture only.

NAS methods have become more popular recently, mainly because of its success in many different computer vision tasks. One stream of NAS methods relies on RL. RL-based NAS methods adopt a controller to sample a new candidate network to be trained and its performance is used as the reward score. Then, the reward score can be adopted to update the controller for sampling a better candidate network in the next iteration. Although early RL-based NAS methods [16], [31] can yield good performance on the small-scale dataset, these methods demand expensive computational resources when the task dataset is large. In addition, the network architecture designed by RL-based NAS methods cannot perform well when transferred to other tasks with different input data sizes. After that, inspired by Inception and ResNet Series [4], [5], [32]–[34], Zoph *et al.* [24] further proposed a tradeoff method, called NASNet, which designs network architectures blockwise on a small dataset and then transfers the block structure to a big dataset. The block structure is usually repeated several times and stacked sequentially to generate deeper complete network architectures. However, the number of the repeated blocks is a free parameter, which needs to be carefully tuned according to the scale of a target dataset. One never knows the appropriate number of blocks in solving a new task.

EC-based NAS is another class of popular NAS methods. Already decades ago, the evolutionary algorithms have been used for finding not only the topology of neural networks but also the weights and hyperparameters, which is called evolutionary neural networks [35], [36] or neuroevolution [37], [38]. Evolutionary NAS (EvoNAS) is, in principal, similar to evolutionary neural networks or neuroevolution techniques, however, focusing on optimizing the architecture of deep neural networks [17], [39], [40]. The neural architecture will be encoded as a chromosome (also known as an individual) through an associated encoding strategy, which is the first step in accomplishing EvoNAS. Then, EvoNAS randomly generates a population of parent individuals with different architectures, and trains them. When all the individuals are trained for several epochs, the fitness of each individual is achieved by evaluating the network on the validation dataset. After that, an offspring population is generated by applying crossover and mutation operators on the parents. Then, the better offspring individuals become as the parents of the next generation through environmental selection. This evolutionary process repeats for a predefined number of generations until certain conditions are met.

The LargeEvo algorithm [40] is one of the early work of EvoNAS, which adopted the genetic algorithm (GA) to find the good architecture of a CNN. Sun *et al.* [7] proposed a variable-length encoding scheme to find the optimal depth of the CNN architecture using the GA. Moreover, EvoNAS attracted much attention since they can offer a viable solution

for the problem of multiobjective NAS. In fact, as already done in evolutionary optimization of shallow neural networks [41], real-world tasks usually demand to consider other optimization objectives in addition to learning performance, such as floating point operations (FLOPs) [10], [42], the inference time of a network [43], latency [21], [44], network's memory consumption [21], and communication cost [45] in the context of federated NAS [46], to find neural architectures with different complexities for different deployment scenarios. Since a large number of candidate models need to be trained from scratch frequently for evaluation, most of these methods are still computationally intensive.

B. One-Shot Neural Architecture Search

In order to accomplish NAS tasks within a practical period of time, many approaches to reduce the computational complexity of evaluating each candidate architecture have been proposed. OSNAS is one of the early efforts that show promise for reducing computation cost. Central to the idea of OSNAS is the training of a single one-shot model containing all candidate network architectures in the search space, and then all the network architectures sampled from this one-shot model share the weights of their common nodes. Hence, the candidate architecture can inherit trained weights from the one-shot model and be directly evaluated on the validation dataset without any additional training. The efficiency of performance estimation of the candidate architectures is therefore greatly enhanced because it avoids training a large number of candidate models from scratch.

The main difference in different OSNAS methods lies in the different training methods of the one-shot model. Since the search space of neural architectures is discrete and cannot be formulated with a continuous function, designing a neural architecture cannot be solved directly using the GD-based method. To tackle this issue, ENAS [20] uses an RNN controller to sample the network architectures from the search space and trains the one-shot model through approximate gradients. However, since the weights in the one-shot model are strongly coupled, search for the neural architecture may be misled. Much follow-up research on OSNAS focuses on further reduction of the weight coupling effect, such as the random sampling strategy with path dropout [28], uniform sampling strategy [21], and group weight sharing strategy [29].

Another family of OSNAS methods has attempted to transform the discrete search space into a continuous space with a softmax function through relaxation tricks. DARTS [25] transforms the choice of operation into weighting a fixed set of candidate operations. However, DARTS requires heavy GPU memory since all candidate operations have to be explicitly instantiated in the memory [26]. Xie *et al.* [47] proposed a stochastic NAS (SNAS) method, which can use gradient information from generic differentiable loss to improve the efficiency of the architecture search. Chen *et al.* [48] attempted to solve this issue by reducing the size of the search space. ProxylessNAS [49] converts the real-valued network architecture parameters into binary representations via binary gates (0 or 1). Hence, ProxylessNAS forces only a single path to be active during the one-shot model training. This way, the memory consumption of ProxylessNAS will be approximately the same as that of training a single network architecture. Partially connected DARTS (PC-DARTS) [26] proposed a channel sampling strategy that samples a subset of channels in each training step rather than activating all channels to reduce the redundancy of the one-shot model when exploring the search space. Benefiting from the reduced memory cost, ProxylessNAS and PC-DARTS can directly design network architectures on the ImageNet dataset.

Although OSNAS can greatly reduce computation costs, the problem of weight coupling during the training of the one-shot model is a critical issue because it may result in unreliable model rankings. To overcome this, Benyahia *et al.* [27] presented a statistically justified weight plasticity loss function to regularize the training of the shared weights of the one-shot model. Zhang *et al.* [8] suggested a search-based architecture selection loss to alleviate the negative impact of weight coupling.

C. Multiscale Representation Learning

The convolutional kernel is capable of capturing different levels of local image information, depending on the kernel size used. A relatively small kernel size (e.g., 3×3 kernels) can capture detailed features, such as smaller objects and parts of the objects, whereas large kernel sizes (e.g., 5×5 kernels and 7×7 kernels) can cover a large region of the input image or the whole image and capture more reliable global features, such as larger objects and context information. A recent trend in NAS methods is to integrate multiple types of convolution with different kernel sizes into the NAS search space, which has been demonstrated to be able to improve the model performance [10], [49]. Tan and Le [50] used MobileNets [51], [52] based on the ImageNet dataset [53] to study the impact of convolutional kernel sizes on model performance and observed that very large kernel sizes (e.g., a kernel size larger than 9×9) may potentially damage the model performance in terms of both accuracy and efficiency.

In fact, an image may contain very complex scenes. For example, some objects have large spatial representations, and others have smaller representations. Besides, the same class of objects may be presented in different sizes in the same image. Hence, using a single type of convolutional kernel with a single kernel size is not optimal for such complex images. Multiscale representation learning can process the image or feature maps at different scales to capture different levels of feature information about the content of the scene. There is already work, e.g., feature pyramid network [54], DenseNet [6], and MixNet [50], that combines multiscale feature maps from different layers to improve the model performance for many computer vision applications. Recently, Duta *et al.* [55] proposed PyConv, which mixes up in parallel different sizes of convolutional kernels with varying sizes and depths in a single convolution. To be specific, PyConv contains a double-oriented pyramid of convolutional kernels in which the kernel size increases while the depth of the kernel reduces. Compared with the standard convolution, PyConv

TABLE I
NETWORK OPERATION CODING SPACE

Operation type	Kernel size	Abbreviation	Code
Convolution	3	Conv3	0
Convolution	5	Conv5	1
Convolution	7	Conv7	2
Max pooling	3	MAX	3
Average pooling	3	AVG	4
Squeeze-and-excitation	–	SE	5
PyConv-1	3,5,7	PyConv-1	6
PyConv-2	3,5	PyConv-2	7
PyConv-3	3,7	PyConv-3	8
PyConv-4	5,7	PyConv-4	9

is able to process the input at different convolutional kernel scales without increasing the number of parameters of the model.

III. ARCHITECTURE SEARCH SPACE

In this section, we first present the architecture search space of the proposed algorithm and the corresponding encoding strategy. Then, we introduce the PyConv operations.

A. Convolutional Neural Network Blocks and Encoding Strategy

In our method, a network architecture is composed of several blocks where each block has a different structure. A block is a small convolutional network that implements a feedforward computation process. Hence, the block can be represented by a DAG, where each hidden node stands for a layer in the CNN, and directed edges represent the flow of data information from one layer to another. In our network architecture representation, each hidden node is depicted by a 5-tuple, (o_i, i_1, i_2, m_1, m_2). In the scheme, the first three integer numbers specify the *operation type*, *input index 1*, and *input index 2*, which are called the structure genes. The last two integers are switch genes, indicating whether the edge is activated or not with a binary number (0 and 1). To be specific, for a hidden node a in a block b, $o_i \in O$ stands for an operation applying on input data. O is the operation space. Ten available operations and their corresponding coding values are summarized in Table I. More details about the PyConv operations will be given in Section III-B. Each hidden node has two predecessors. The input indices i_1 and i_2 specify the inputs to node a, which refer to the position of predecessor nodes of node a in block b. The output of two predecessor nodes will be combined by an elementwise addition operation. Next, we will explain the details of the function of the switch genes. As illustrated in Fig. 1, when $m_1 = 1$ and $m_2 = 0$, the output of the first predecessor node i_1 is set to zero, and node a only has the second predecessor node i_2; when $m_1 = 0$ and $m_2 = 1$, the output of the second predecessor node i_2 is set to zero, and node a only owns the first predecessor node i_1; when $m_1 = 0$ and $m_2 = 0$, the output of nodes i_1 and i_2 is summed up via the elementwise addition operation and inputted into node a; finally, when $m_1 = 1$ and $m_2 = 1$, node a is deactivated and the output of nodes i_1 and i_2 is summed up and bypasses node a in a shortcut. In addition, each block only has one input, which is the output of the first convolutional layer or the output of the previous blocks. All nodes without any successor in the block are concatenated along the depth dimension to provide the final block's output.

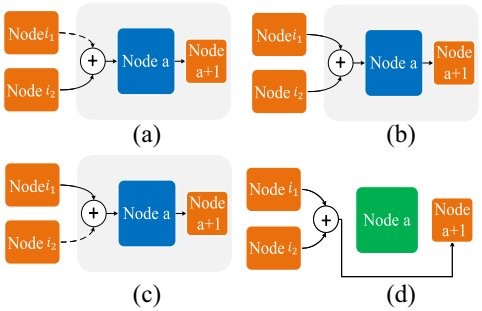

Fig. 1. Hidden node contains two predecessor nodes and two gates. A solid rectangle indicates the input edge is active while a dashed one represents the input edge is inactive. (a) $m_1 = 1$ and $m_2 = 0$. (b) $m_1 = 0$ and $m_2 = 0$. (c) $m_1 = 0$ and $m_2 = 1$. (d) $m_1 = 1$ and $m_2 = 1$.

Based on the above-defined blocks, the simplest way of constructing a complete neural network is to stack these blocks sequentially and generate chain-structured neural networks. Zhong *et al.* [14] observed that stacking blocks may not be optimal for connecting blocks. Hence, we add a bit for each block to represent a shortcut connection between two blocks so that the input flow can go directly to the output and bypass the whole block. We call it *shortcut coding*. In this way, the connection between specific blocks can be automatically optimized by the proposed algorithm. Note that we use the max pooling layer with stride 2 and the convolutional layer with kernel 1 to match the different dimensions for connected blocks.

An example network automatically designed by the proposed algorithm is shown in Fig. 2. To be specific, the overall network architecture starts with an input layer, with five blocks being stacked beyond the input. Each block consists of nine hidden nodes and one source node. The source node is the input of the block, which is the output of the previous blocks or the input of the overall network. The input layer is composed of a convolutional layer with stride 1. We note that the input layer of the network designed based on ImageNet is composed of three convolutional layers with stride 2 to reduce the resolution of the input image from 224×224 to 28×28. For convolutional block 2 and convolutional block 4, we apply all operations with a stride of 2 to perform downsampling for input feature maps. The stride of the other convolutional blocks is set to 1, which returns feature maps with the same spatial resolution. When the size of a feature map is halved, the block's channels will be doubled. Overall, the size of the proposed search space is approximately 5.73×10^{111}.

Several previous NAS algorithms [20], [24], [25], [56] focus on searching local block structures and then stack the same blocks sequentially to construct a complete network architecture. They also restrict the block to own a fixed number of nodes and define the node by a binary tree representation, where the results of two layers (e.g., convolution or pooling) must be combined through an addition or concatenation

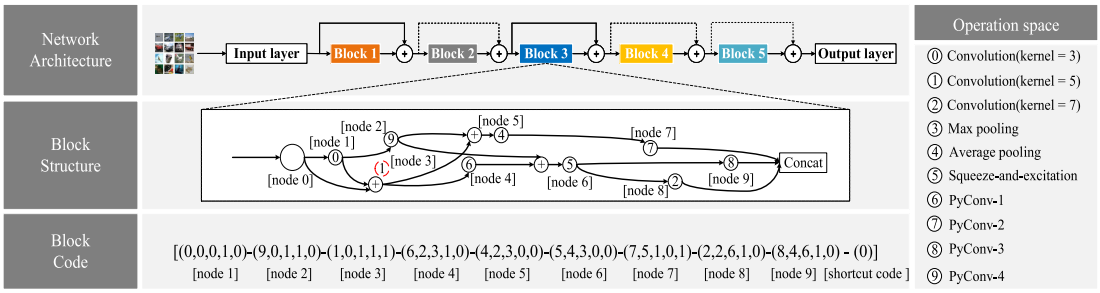

Fig. 2. Top: A designed network architecture composed of five blocks. Middle and bottom: Examples of block 3 with its block code. A block is composed of nine nodes and one source node (node 0). A solid black circle indicates that the node is activated while a dashed red one indicates that the node is deactivated.

operator. This representation proposed in this work differs from the previous ones in the following aspects.

1) The proposed representation is able to discover network architectures composed of blocks with a different structure, thereby providing a larger search space. Additionally, instead of using a fixed number of nodes in a block (all nodes), the block structure can be changed to have different numbers of nodes based on the proposed node representation scheme. Hence, we can construct simple or more complex network structures by controlling the number of active nodes. Inactive nodes can be regarded as temporarily switched off (dropout) and will not participate in training and updating weights during the search. Besides increasing the diversity of block structure and reducing the computational burdens, this strategy reduces the weight coupling in the one-shot model and regularizes the evolutionary search.
2) The proposed representation allows for directly designing a complete network architecture without any proxy metrics, including the optimal block structures and the connection between specific blocks. All generated network architectures can be directly used without any postprocessing or recomposition.

B. Pyramidal Convolution Operations

Apart from the network architecture design using NAS, the operation in the operation space also has a major influence on the performance of a network architecture. Inspired by the promising performance of multiscale representation learning [50], [57], we propose a set of multiscale convolution operations based on pyramidal convolution [55], called PyConv operations. We integrate them into the NAS operation space. Hence, the PyConv operations can adaptively incorporate into neural architectures with the help of the Evo-OSNAS.

As shown in Table II, four types of PyConv operations are designed: 1) PyConv-1; 2) PyConv-2; 3) PyConv-3; and 4) PyConv-4. PyConv-1 has three parallel branches and the rest PyConv operations have two parallel branches. Each branch of the PyConv operation contains different types of convolutional kernels. We also use grouped convolution tricks [58]–[60] to enable each branch of the PyConv operation to have different depths in the convolutional kernels, thus reducing the number of parameters and the computational burden. The number of groups and the number of output channels at each branch are set according to [55].

TABLE II
COMPONENTS OF PYCONV OPERATIONS

PyConv operation	Input channels	Branches	Number of groups (G)	Output channels
PyConv-1	64/128/256	7×7	8	32/64/128
	64/128/256	5×5	4	16/32/64
	64/128/256	3×3	1	16/32/64
PyConv-2	64/128/256	5×5	4	32/64/128
	64/128/256	3×3	1	32/64/128
PyConv-3	64/128/256	7×7	8	32/64/128
	64/128/256	3×3	1	32/64/128
PyConv-4	64/128/256	7×7	8	32/64/128
	64/128/256	5×5	4	32/64/128

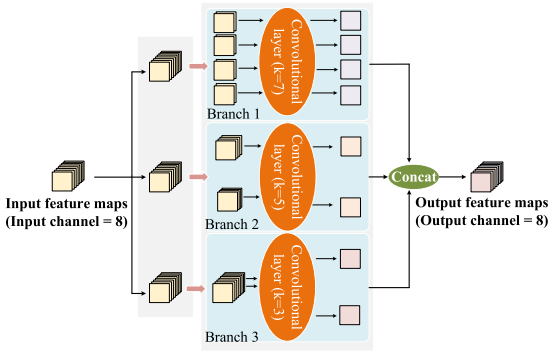

Fig. 3. Example of PyConv-1 including three parallel branches.

Fig. 3 shows the components of PyConv-1, which consists of three branches (in blue). In this example, there are eight input feature maps. In each branch, we use the following steps: we split the input feature maps into different groups and apply the convolutional kernels separately for each group. For example, in branch 1, the eight input feature maps are split into four groups, where the convolutional layers with a kernel size 7 are applied independently in each group. Hence, the depth of the convolutional kernels is reduced to four. The output feature maps in each branch are concatenated to provide the output feature maps of the PyConv operation. Note that the number of input and output feature maps should be equal.

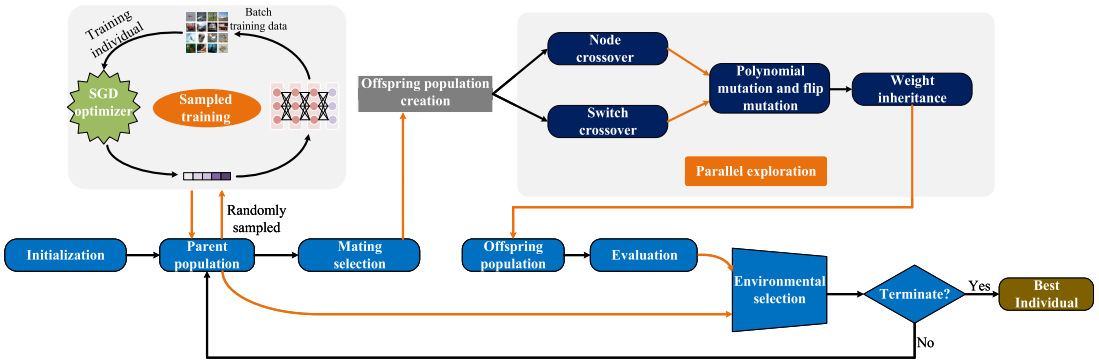

Fig. 4. Overall framework of Evo-OSNAS.

Algorithm 1: Pseudocode of Evo-OSNAS

Input: population size M, the maximal generation T, the crossover probability γ, the mutation probability β, training data D_{train}, validation data D_{valid}
Output: the best neural network architecture
$P_0 \leftarrow$ Initialize the population with the size M via ***the proposed encoding strategy***
$t \leftarrow 0$
while $t < T$ **do**
  Training the individuals in P_t by using the sampled training strategy [30] on D_{train}
  Evaluate the fitness of individuals in P_t on D_{valid}
  $Q_t \leftarrow \varnothing$
  while $|Q_t| < M$ **do**
    $p_1, p_2 \leftarrow$ Select two parent individuals from P_t through binary tournament selection
    $q_1, q_2 \leftarrow$ Generate two offspring with ***the designed crossover operation with the probability of*** γ
    $Q_t \leftarrow Q_t \cup q_1 \cup q_2$
  end
  for *each individual* q_i *in* Q_t **do**
    Perform ***the designed mutation operation with the probability of*** β
  end
  Evaluate the fitness of individuals in Q_t on D_{valid}
  $P_{t+1} \leftarrow$ Select M individuals from $P_t \cup Q_t$ through environmental selection
  $t \leftarrow t + 1$
end
Return the best individual and decode it into the corresponding neural network.

IV. Proposed Algorithm

In this section, we first introduce the overall framework of the proposed algorithm, and then discuss its main components.

A. Overall Framework

Algorithm 1 lists the pseudocode of the main components of Evo-OSNAS, where the main contributions of this work are highlighted by italic and bold. First, a population of size M is randomly initialized through the proposed encoding strategy. Then, the population is evolved until the maximum number of generations T is reached. After the evolutionary phase, the individual that exhibits the best fitness is decoded into the corresponding network architecture. The overall framework of the proposed method is illustrated in Fig. 4.

During the evolutionary phase, all individuals in the population P_t are decoded into the corresponding neural architecture and trained by the sampled training strategy [30]. After training, all individuals are evaluated based on the validation data D_{valid}. In the Evo-OSNAS, the fitness of an individual is the validation classification accuracy. Then, two parent individuals in P_t are selected by tournament selection to generate two new offspring using the proposed crossover and mutation operations. This is repeated until M offspring individuals are generated. Then, the fitness of all offspring individuals in Q_t is evaluated, and the next parent population P_{t+1} is generated via the environmental selection operation proposed in [7]. This is achieved by combining the parent and offspring populations P_t and Q_t and then select M individuals from the combined population using binary tournament selection. Once M individuals are selected for P_{t+1}, check if the best individual of the combined population is not selected, and if not, it is then selected to replace the worst individual in the current P_{t+1}. We repeat the same routine for a predefined number of generations and output the best individual at the end of the algorithm. In the following, the main steps of the proposed algorithm, namely, population initialization, crossover, and mutation, will be elaborated.

B. Population Initialization

The population is randomly initialized based on the proposed encoding strategy, in which each individual represents a candidate neural architecture. The pseudocode for population initialization is given in Algorithm 2. Encoding a hidden node of a block is composed of three steps. We first randomly select one integer o_i from the operation space. Then, two node indexes, i_1 and i_2, are selected from the nodes before the current node. Then, two binary numbers, b_1 and b_2, are randomly generated. Then, a shortcut code is randomly generated. Hence, a block is composed of 45 hidden node genes and 1 shortcut gene. By repeating the above steps, we can generate a chromosome that contains structure genes and

Algorithm 2: Population Initialization

Input: population size M
Output: The initialized population P_0
$P_0 \leftarrow \varnothing$
for $m \leftarrow 1$ *to* M **do**
  $p_m \leftarrow \varnothing$
  for $b \leftarrow 1$ *to* 5 **do**
    $D_b \leftarrow \varnothing$
    for $k \leftarrow 1$ *to* 9 **do**
      $n_k \leftarrow \varnothing$
      $o_i \leftarrow$ Randomly generate an integer from [0, 9]
      $i_1, i_2 \leftarrow$ Randomly generate two integers from [0, k-1]
      $b_1, b_2 \leftarrow$ Randomly generate two integers from [0, 1]
      $n_k \leftarrow o_i \cup i_1 \cup i_2 \cup b_1 \cup b_2$
      $D_b \leftarrow D_b \cup n_k$
    end
    $S \leftarrow$ Randomly generate one integer from [0, 1]
    $p_m \leftarrow p_m \cup D_b \cup S$
  end
  $P_0 \leftarrow P_0 \cup p_m$
end
Return P_0

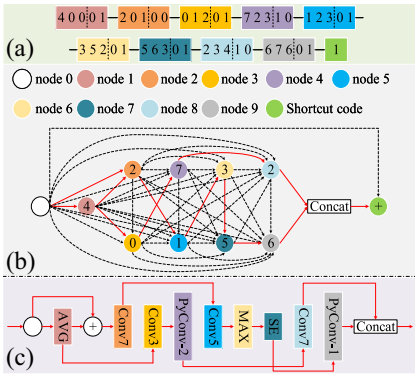

Fig. 5. (a) Exemplar block code of an individual. (b) Representative blockwise search space containing nine hidden nodes. Node 0 is the input to the network, whereas node 8 and node 9 are the output of the network. The digit in each node, such as 2 or 7, is the coding of the ten available operations in the operation space. The red arrows denote the active paths in the search space. Dotted arrows denote the inactivated paths. (c) Block network structure of an individual represented by the block code.

switch genes. The same approach is repeated to generate the initial population.

Then, we will explain the details of decoding a chromosome into a network. First, we represent the proposed search space using a DAG. Fig. 5 exhibits a generic example of the search space, block coding, and corresponding network structure. For convenience, we give only one block as an example that contains nine hidden nodes. We call it a blockwise search space. Specifically, each hidden node can choose one computing operation from the operation space as the local computation module and receive outputs from its previous nodes only. For example, node 3 can receive the output of node 0, node 1, and node 2. Hence, node 3 contains three candidate paths. The blockwise search space contains all candidate paths between all nodes, as shown in Fig. 5(b). The overall search space can be assembled by stacking these blockwise search spaces sequentially. Each individual in the population is a path of the search space as shown in Fig. 5(b). When evaluating the individual's performance, the corresponding nodes and paths can be activated based on the proposed encoding strategy and then decoded into a network as shown in Fig. 5(c). Once the population initialization process is completed, corresponding nodes contained by individuals in the population can be activated. All activated nodes are connected to construct the one-shot model.

C. Offspring Generation

Given a population of neural architectures, the new offspring population is generated by the crossover and mutation operators. Generally, offspring individuals generated by genetic operators represent new candidate solutions for the problem to be solved. In the proposed method, each individual in the offspring population represents a new network architecture. Generally, previous EvoNAS algorithms [7], [10], [13] generate offspring architectures by exchanging the components of two parent architectures by means of crossover. However, since the offspring directly share the trained weights with parents as the initial weights of the corresponding nodes, and can directly evaluate their fitness without any separate training in the OSNAS framework, the traditional crossover may introduce a bias as it underestimates the true performance of offspring individuals.

In this work, we aim to reduce some of the negative impacts of weight coupling with the help of the proposed encoding scheme and the corresponding crossover operators. An illustrative example of the proposed crossover is given in Fig. 6. The proposed crossover is composed of node crossover and switch crossover, applied on the structure chromosome and switch chromosome, respectively. During the crossover, the algorithm first needs to choose a crossover operation. Each of the two types of crossover operations is implemented at an equal probability. When crossover for the structure genes is carried out, two offspring individuals are created by the nodes sampled from two parents. Consequently, the offspring will share the corresponding trained weights with the two parents for the fitness evaluations. In contrast, when crossover is applied on the switch genes, one offspring is generated by the nodes sampled from one parent individual. As a result, the offspring can inherit all weights from its parent. Fig. 6(b) illustrates an example of the switch crossover. For example, in offspring q_1, compared with the topology of parent architecture p_1, the edge between the source node and node 4 is switched from inactive to active, and the shortcut connection in this block is changed from active to inactive. Since the component of offspring q_1 is the subset of parent p_1, offspring q_1 and parent p_1 do not have interference with each other when q_1 shares the weights of p_1.

Overall, the proposed crossover operator aims to reduce the degree of weight sharing, since the offspring can only share the weights from two parent individuals or one parent individual

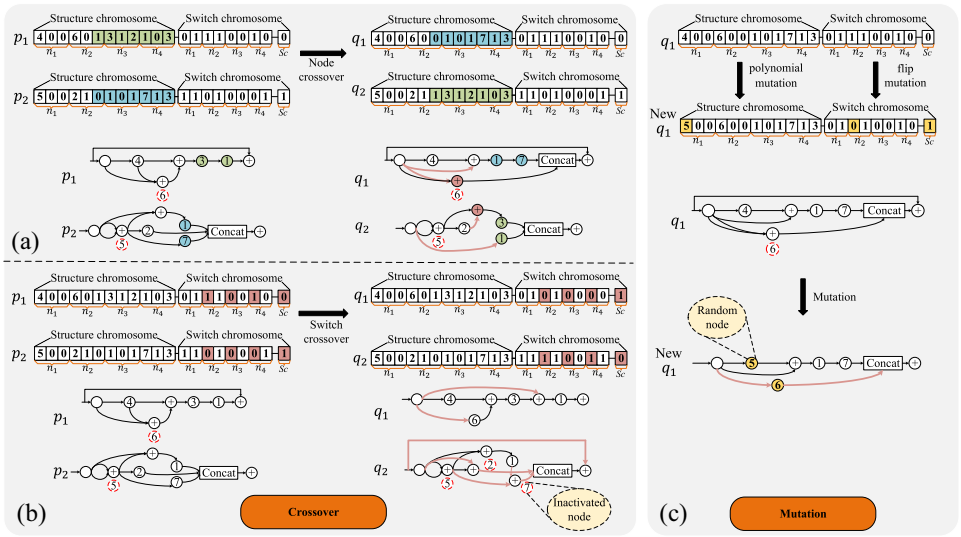

Fig. 6. Illustrations of the proposed crossover and mutation operators. For clarity, we use four nodes as an example instead of all nine hidden nodes in the block structure, including node 1 (n_1), node 2 (n_2), node 3 (n_3), node 4 (n_4), and one shortcut gene (S_c). For each node, the first three structure genes are put into the structure chromosome, and the last two switch genes are put into the switch chromosome. (a) Example of node crossover, where two parent chromosomes, p_1 and p_2, are separated based on the randomly chosen positions. Then, the offspring architecture q_1 is generated by combining the first part of parent p_1 with the second part of p_2. Similarly, the second offspring q_2 consists of the second part of p_1 and the first part of p_2. (b) Example of switch crossover, where multiple points are randomly chosen from parents p_1 and p_2 and then the offspring individuals q_1 and q_2 are generated by exchanging the genes of the corresponding positions. (c) Example of mutation, where offspring individual q_1 has a mutation by mutating the first gene of the structure chromosome, and the third and ninth genes of the switch chromosome. In the above figures, a solid red circle indicates that the node is activated while a red dashed one indicates that the node is deactivated. In addition, the blue and green solid circles represent the exchanged nodes, and the red edges denote the changes of the data flow after the crossover operation. Finally, the yellow solid circles represent the mutated node.

in each generation. As a result, the negative impacts of weight coupling in the one-shot model training can be reduced.

To enhance the algorithm's ability to escape from the local optimums and promote the diversity of the population, we use a hybrid mutation operator following crossover. In evolutionary algorithms, mutation is a local operator that mutates a candidate solution to generate a new solution in its vicinity. In this work, we apply the mutation to every individual generated by the proposed crossover operator. For a given offspring architecture, we apply a discretized version of the polynomial mutation (PM) operator [61] on the structure genes, and flip mutation on the switch genes. The setting of the hyperparameter η_m in the PM follows the recommendation in [44]. The proposed mutation performs the following.

1) Replacement (replacing the selected nodes with the random nodes that do not belong to the one-shot model).
2) Input change (changing the input of the selected node).
3) Switch on (activating disabled edges).
4) Switch off (disabling activated edges).

Fig. 6(c) shows an example of the proposed mutation. For example, the average pooling operation of node 1 in offspring q_1 is mutated to the squeeze-and-excitation operation. The benefit of the proposed mutation operation is to make the one-shot model training less likely to get trapped in local minimums. At the same time, the weight coupling in the one-shot model is also reduced. Note that the weights of a newly generated random node are randomly initialized. The effectiveness of the proposed crossover and mutation will be experimentally demonstrated in Section VI-B.

V. Experimental Settings

In this section, we first introduce the image classification benchmark datasets adopted in this work. Then, we briefly introduce the baselines for comparison and the implementation details of the proposed method.

A. Baselines

To demonstrate the effectiveness and superiority of the proposed algorithm, we consider two different classes of peer competitors for comparison: 1) CNN architectures that are designed by non-OSNAS algorithms (mainly EC or RL) and 2) CNN architectures that are generated through OSNAS algorithms. The first category includes AmoebaNet-A [17], AE-CNN [7], NSGANetV1 [10], BlockQNN-S [14], and MixNet [50]. The second category consists of a set of recent OSNAS methods, many of which achieve state-of-the-art results on various image classification benchmark datasets.

For comparison on classification performance, four widely used benchmark classification tasks are considered, namely, CIFAR10 [62], CIFAR100 [62], CINIC10 [63], and ImageNet [53]. In addition, we also use six medical image analysis datasets that are extracted from MedMNIST series datasets [64] with diverse data scales, including PathMNIST, DermaMNIST, OCTMNIST, OrganMNIST-Axial, OragnMNIST-Coronal, and OrganMNIST-Sagittal. For the convenience of the discussions, we denote these datasets as Path, Derma, OCT, O-A, O-C, and O-S. The details of these datasets and the data augmentation procedure are described in Sections I and II in the supplementary material, respectively.

TABLE III
COMPARISON BETWEEN EVO-OSNAS AND OTHER NAS AND OSNAS METHODS IN TERMS OF THE TEST CLASSIFICATION ACCURACY (ACC), THE NUMBER OF PARAMETERS (P), AND THE SEARCH COST (IN GDS) ON THE CIFAR10, CIFAR100, AND CINIC10 BENCHMARK DATASETS

Algorithm	CIFAR10			CIFAR100			CINIC10			Search method
	ACC(%)	# P(M)	GDs	ACC(%)	# P(M)	GDs	ACC(%)	# P(M)	GDs	
AmoebaNet-A [17]	96.66±0.06	3.1	3150	81.07‡	3.1	3150	–	–	–	EC
CNN-GA [13]	96.78	2.9	35	79.47	4.1	40	–	–	–	EC
AE-CNN [7]	95.30	2.0	27	77.60	5.4	36	–	–	–	EC
AE-CNN+E2EPP [19]	94.70	4.3	7	77.98	20.9‡	10	–	–	–	EC
NSGANetV1-A3 [10]	97.78	2.2	27	82.77	2.2	27	–	–	–	EC
NASNet-A [24]	97.35	3.2	1800	82.19‡	3.2	1800	–	–	–	RL
BlockQNN-S [14]	96.70	6.1	90	82.95	6.1	90	–	–	–	RL
PNAS [15]	96.59±0.09	3.2	225	82.37‡	3.2	225	–	–	–	SMBO
MdeNAS [56]	97.45	3.8	0.16	82.39‡	3.8	0.16	–	–	–	MDL
RENAS [65]	97.12±0.02	3.5	6	–	–	–	–	–	–	RL+EC
DARTS[25]	97.00±0.14	3.3	1	82.46	3.3	1	92.95±0.09*	3.4*	1.4*	OSNAS+GD
SNAS [47]	97.15±0.02	2.8	1.5	79.91‡	2.8	1.5	92.88±0.07*	3.2*	2.2*	OSNAS+GD
PDARTS [48]	97.50	3.4	0.3	84.08	3.6	0.3	–	–	–	OSNAS+GD
PC-DARTS [26]	97.43±0.07	3.6	0.1	82.89‡	3.6	0.1	93.32±0.05*	3.1*	0.22*	OSNAS+GD
GDAS-NSAS [8]	97.25±0.08	3.5	0.4	81.98±0.05	3.5	0.4	–	–	–	OSNAS+GD
Proxyless NAS [49]	**97.92**	5.7	–	–	–	–	–	–	–	OSNAS+GD
ENAS [20]	97.11	4.6	0.5	81.09‡	4.6	0.5	90.11±0.05*	5.5*	1.1*	OSNAS+RL
WPL [27]	96.19	–	–	–	–	–	–	–	–	OSNAS+RL
Random (F=64) [28]	95.60	6.7	–	–	–	–	–	–	–	OSNAS+Random
RandomNAS-NSAS [8]	97.41±0.05	3.1	0.7	82.44±0.05	3.1	0.7	–	–	–	OSNAS+Random
SI-EvoNAS [30]	97.31	1.84	0.458	**84.30**	3.32	0.813	92.99±0.06*	3.5*	1.3*	OSNAS+EC
Evo-OSNet	97.44±0.10	3.3	0.5	84.16±0.04	3.5	0.5	**93.41±0.03**	3.2	0.9	OSNAS+EC

Symbol "‡" indicates that the results were not available in their original papers, and are taken from [8, 10]. Symbol "–" indicates that no results have been reported on the dataset in the original paper. Symbol "*" indicates that the results were obtained by ourselves by running the code released by the authors with the same experimental settings, because the authors of the original paper have not run their algorithm on CINIC10.

B. Implementation Details

For Evo-OSNAS, the parameter settings are based on the conventions of the deep learning and EC communities. The details of the parameter settings are provided in Section III in the supplementary material. We employ the standard SGD algorithm to perform sampled training strategy [30] for all individuals in the parent population. When the predefined maximum number of the generations is reached, the optimized neural architecture will be fully retrained on the original training dataset with the same SGD parameter settings. All trained architectures will be evaluated on the test dataset. To ensure the statistical reliability of the experimental results of the Evo-OSNAS, the evolutionary search is repeated for five times on the above-discussed classification datasets with different initial random seeds. The average classification accuracy on the test dataset is reported for comparison. Note that the original test dataset should never be involved in the architecture search. All experiments are performed on 8 Nvidia RTX8000 GPU cards.

VI. EXPERIMENTAL RESULTS

In this section, we first present the experimental results of the proposed algorithm on the four benchmark classification datasets in terms of not only the classification accuracy but also the number of parameters, and the consumed "GPU days (GDs)" are used to search the corresponding neural network. As suggested in [7], [13], and [19], we use the "GDs," the computation time spent on GPUs, to measure the search cost of different NAS algorithms. The number of GDs is calculated by multiplying the number of used GPU cards by the execution time in days. Moreover, to help the readers to understand whether the Evo-OSNAS converges with the used parameter settings, we also observe the evolutionary trajectories of the Evo-OSNAS in discovering the best networks on each dataset, and the results and discussions are described in Section IV in the supplementary material. Then, we analyze the effectiveness of the proposed crossover operation, mutation hyperparameters, and PyConv operations. Finally, we apply the proposed method to the medical image analysis tasks, for further validating Evo-OSNAS' capability of automatically designing task-dependent neural architectures.

A. Results and Discussion

The experimental results comparing Evo-OSNAS with the peer competitors on CIFAR10, CIFAR100, CINIC10, and ImageNet classification datasets are shown in Tables III and IV, respectively. The best classification results are highlighted in bold. The model designed by the proposed algorithm is called Evo-OSNet. Note that the most experimental results of the peer competitors provided in Tables III and IV are extracted from their original papers.

1) Results on CIFAR10, CIFAR100, and CINIC10: As shown in Table III, Evo-OSNAS performs better than the non-OSNAS peer competitors [7], [13]–[15], [17], [19], [24], [65] with similar parameters, but the search costs only 0.5 GDs, on both CIFAR10 and CIFAR100. Although NSGANetV1-A3 [10] shows better classification accuracy with fewer parameters than Evo-OSNet, the search costs of Evo-OSNet is much lower (only 0.5 GDs, which is much less than 27 GDs). MdeNAS shows a bit better classification accuracy on CIFAR10 but slightly worse accuracy on CIFAR100 compared with Evo-OSNet. Compared with the OSNAS peer competitors, Evo-OSNet achieves competitive results with

TABLE IV
COMPARISON BETWEEN EVO-OSNAS AND OTHER NAS AND OSNAS METHODS IN TERMS OF CLASSIFICATION ACCURACY (%), THE NUMBER OF PARAMETERS, AND THE CONSUMED SEARCH COST ON THE IMAGENET BENCHMARK DATASET

Algorithm	ImageNet test accuracy (%)		# Params(M)	Search Cost (GPU-days)	Search dataset	Search method
	Top-1	Top-5				
NSGANetv1-A3 [10]	76.2	93.0	5.0	27	CIFAR100	EC
AmoebaNet [17]	75.7	92.4	6.4	3150	CIFAR10	EC
PNAS [15]	74.2	91.9	5.1	225	CIFAR10	SMBO
MdeNAS [56]	73.2	–	6.1	0.16	CIFAR10	MDL
MnasNet-A3 [57]	76.7†	93.3†	5.2	–	ImageNet	RL
MixNet-M [50]	77.0†	93.3†	5.0	–	ImageNet	RL
NASNet-A [24]	74.0	91.6	5.3	1800	CIFAR10	RL
RENAS [65]	75.7	92.6	5.36	1.5	CIFAR10	RL+EA
DARTS [25]	73.3	91.3	4.7	4.0	CIFAR10	OSNAS+GD
SNAS(mild) [47]	72.7	90.8	4.3	1.5	CIFAR10	OSNAS+GD
Proxyless NAS(GPU) [49]	75.1†	92.5†	7.1	8.3	ImageNet	OSNAS+GD
PDARTS [48]	75.6	92.6	4.9	0.3	CIFAR10	OSNAS+GD
PDARTS [48]	75.3	92.5	5.1	0.3	CIFAR100	OSNAS+GD
PC-DARTS [26]	74.9	92.2	5.3	0.1	CIFAR10	OSNAS+GD
PC-DARTS [26]	75.8†	92.7†	5.3	3.8	ImageNet	OSNAS+GD
BayesNAS [66]	73.5	91.1	3.9	0.2	CIFAR10	OSNAS+GD
SETN [67]	74.3	92.0	5.4	1.8	CIFAR10	OSNAS+GD
GDAS-NSAS-C [8]	74.1	–	5.2	0.4	CIFAR10	OSNAS+GD
RandomNAS-NSAS-C [8]	74.5	–	5.4	0.7	CIFAR10	OSNAS+Random
FairNAS [68]	75.3†	–	4.6	12	ImageNet	OSNAS+EC
SI-EvoNAS [30]	75.8	92.59	4.7	0.458	CIFAR10	OSNAS+EC
Evo-OSNet	**77.48±0.04**†	**93.53±0.05**†	5.0	8.6	ImageNet	OSNAS+EC

Symbol "†" in Table IV implies this network was searched on ImageNet directly, otherwise, it was searched on CIFAR10/CIFAR100. The "search dataset" means the search and evaluation datasets used for the neural architecture search process.

similar or less search costs on both CIFAR10 and CIFAR100. Although Proxyless NAS [49] shows a better classification accuracy than Evo-OSNet on CIFAR10, the Evo-OSNet has fewer parameters (3.3M < 5.7M). On CIFAR100, Evo-OSNet achieves an average classification accuracy of 84.16%, which is only slightly lower than our previous work (0.14%) [30]. Finally, Evo-OSNet achieves an average classification accuracy of 93.41% on CINIC10 datasets and outperforms all peer competitors considered in the experiments.

2) Results on ImageNet: It is important to note that most NAS and OSNAS methods are computationally too expensive to be directly applied to the large-scale dataset. For example, Xu *et al.* [26] observed that DARTS may suffer from severe degradation and cannot be applied to large-scale tasks when the number of search epochs becomes bigger or when there is an obvious gap in the network depth between the search and evaluation spaces. As a compromise, most NAS and OSNAS methods propose to search the optimal network on a small dataset (e.g., CIFAR10 or CIFAR100) and transfer the found optimal network to a large-scale task. However, the network optimized on the small dataset might be suboptimal on the target task.

In this work, by taking advantage of the low GPU memory requirement and high search efficiency of Evo-OSNAS, we aim to directly search a network architecture on the ImageNet dataset to further demonstrate its effectiveness. The experimental results on ImageNet are provided in Table IV. Evo-OSNet takes 8.6 GDs to find a superior network with similar or fewer parameters compared to those found by peer competitors. FairNAS has found a model with a smaller number of parameters, whereas Evo-OSNet achieves a model with better performance and consumes fewer GDs (8.6 GDs < 12 GDs). Although PC-DARTS consumes only 3.8 GDs, it runs based on a subset of ImageNet that randomly extracts 10% and 2.5% images, respectively, from the entire training dataset for search and evaluation. Evo-OSNet achieves gains of 1.68% and 0.83% in top-1 and top-5 classification accuracy rates with a smaller number of parameters (5.0M < 5.3M) in comparison to PC-DARTS. Hence, Evo-OSNAS is still efficient in this sense. Moreover, because the peer competitors [8], [10], [15], [17], [24], [25], [30], [47], [48], [56], [66], [67] in Table IV did not directly design the network on ImageNet, a direct comparison in terms of the number of parameters and the consumed GDs is unfair for Evo-OSNet.

3) Discussion: Most existing OSNAS methods propose to search for top-performing blocks on the target dataset and then stack these blocks sequentially to construct the complete network. Hence, domain expertise is still required because the number of repetitions of optimized blocks is considered as free parameters that the users need to specify according to the scale of an image classification task. With the proposed encoding strategy, Evo-OSNAS does not bear such limitations and therefore, it can directly search the complete network on the dataset without introducing the proxy of repeating optimized blocks. In addition, since only a single individual is trained in each step of an epoch during the evolutionary search, the GPU memory requirement of Evo-OSNAS is the same as training a single network. In summary, the experimental results demonstrate that Evo-OSNAS can efficiently design task-specific neural networks without relying on human expertise on the image classification tasks ranging from small-scale dataset to large-scale ones such as ImageNet, and achieve satisfactory performance. To further validate the effectiveness of Evo-OSNAS, we conduct statistical tests and stability analysis

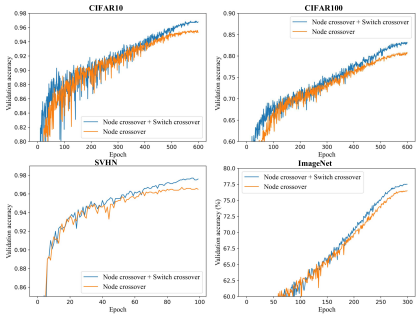

Fig. 7. Comparing the effectiveness of the proposed node operator + switch crossover and node crossover on four benchmark datasets.

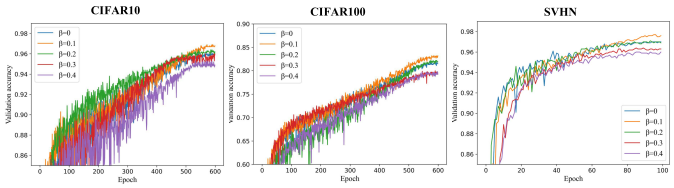

Fig. 8. Analysis of mutation probability β on the CIFAR10, CIFAR100, and SVHN. We select mutation probability β to be 0, 0.1, 0.2, 0.3, and 0.4.

on the given classification datasets, and the results are provided in Sections V-A and V-B in the supplementary material, respectively.

To further validate the generalization and transferability capability of Evo-OSNAS, we also search the network on CIFAR10. Then, the network is transferred to CIFAR10.1, ImageNet V2, ImageNet, and medical image analysis tasks. From these results, we can see that the Evo-OSNAS achieves comparable accuracies with lower or similar search costs compared to the peer competitors' methods [20], [25], [47]. More details are provided in Sections V-C and V-D in the supplementary material.

B. Ablation Studies

1) Analysis of Crossover: The crossover and the mutation operations play significant roles in evolutionary algorithms. As previously introduced in Section IV-C, the proposed crossover operator differs from most existing EC-based NAS methods in the exploration of the search space. We run the proposed method with and without the switch crossover on four datasets: 1) CIFAR10; 2) CIFAR100; 3) SVHN [69]; and 4) ImageNet for comparison. Fig. 7 presents the classification results of the four datasets. As we can see from Fig. 7, on all four datasets, compared with the single node crossover, the proposed crossover operator is able to find neural network architectures with better performance.

2) Analysis of Mutation: The mutation operator employed in the proposed method is controlled by the mutation probability β. The higher the mutation probability, the more random nodes may be incorporated into the one-shot model. To identify the optimal mutation probability rate, we perform the parameter sweep experiments on CIFAR10, CIFAR100, and SVHN datasets. We sweep the value of β from 0 to 0.4. Fig. 8 shows the effect of mutation probability β. Specifically, the proposed method with the mutation probability of 0.1 can achieve the best performing architectures on CIFAR10, CIFAR100, and SVHN, while a mutation probability 0, 0.2, 0.3, and 0.4 harms performance. This observation indicates that properly incorporating random nodes that are not in the one-shot model can help the algorithm find individuals with better performance. Based on this observation, we set the mutation probability β to 0.1.

3) Analysis of PyConv Operations: In the deep learning community, the convolutional kernel size is one of the most significant factors that can strongly impact the performance of the model for visual recognition. In fact, large convolutional kernels can cover a large region of the image through the cost of high model complexity. In order to take advantage of large kernel sizes without increasing the number of parameters, one of the conventional practices is to incorporate dilated convolution operations [70] with different kernel sizes into the search space [10], [25], [48]. In this work, the PyConv operations are used to extend the receptive field of the convolutional kernel without additional parameters. Meanwhile, PyConv operations provide the ability for parsing the input with multiple scales to capture more detailed information. To check the effectiveness of the PyConv operations, we compare the model generated by Evo-OSNAS with another network. The network is designed by Evo-OSNAS with dilated convolution operations, including 3×3 dilated convolution, 5×5 dilated convolution, 7×7 dilated convolution, and 9×9 dilated convolution, which is denoted by Evo-OSNAS + dilated conv. At the same time, the PyConv operations are switched off. For a fair comparison, all other parameter settings are kept the same as described in Section V-B. All experiments are performed five times with five different random seeds to get the statistical results.

TABLE V
COMPARISON OF EVO-OSNAS+PYCONV AND EVO-OSNAS+DILATED CONV IN TERMS OF THE CLASSIFICATION ACCURACY (%) ON THE CIFAR10, CIFAR100, AND CINIC10 BENCHMARK DATASETS

	CIFAR10	CIFAR100	CINIC10
Evo-OSNAS + PyConv	97.44±0.10	84.16±0.04	93.41±0.03
Evo-OSNAS + dilated conv	95.33±0.10	77.77±0.08	91.06±0.10

Table V presents the classification results of the two models under comparison on CIFAR10, CIFAR100, and CINIC10. As we can see from Table V that Evo-OSNAS + PyConv obtains the classification accuracies of 97.44%, 84.16%, and 93.41% on CIFAR10, CIFAR100, and CINIC10, respectively, while Evo-OSNAS + dilated conv obtains 95.33%, 77.77%, and 91.06%, respectively. These results imply that the PyConv operations used in the Evo-OSNAS are more powerful than the dilated convolution operations.

C. Application to Medical Image Analysis

In this section, the effectiveness of the Evo-OSNAS will be demonstrated on medical image analysis tasks. We choose six multiclass classification datasets in MedMNIST [64], which are provided in Section I in the supplementary material. We first execute Evo-OSNAS on these datasets, respectively, to find task-dependent models. Then, we train these models

TABLE VI
COMPARISON BETWEEN EVO-OSNET AND THE MODELS DESIGNED BY OTHER BASELINE METHODS IN TERMS OF CLASSIFICATION ACCURACY (%) ON THE MEDMNIST SERIES DATASETS

Network	Path	Derma	OCT	O-A	O-C	O-S
ResNet18 [5]	86.0	75.0	75.8	92.1	88.9	76.2
ResNet50 [5]	84.6	72.7	74.5	91.6	89.3	74.6
Auto-sklearn [71]	18.6	73.4	59.5	56.3	67.6	60.1
AutoKeras [72]	86.4	75.6	73.6	92.9	91.5	80.3
Google AutoML Vision [73]	81.2	76.1	73.2	81.6	86.2	70.7
SI-EvoNAS [30]	90.58±0.65	76.66±0.39	78.14±0.23	92.98±0.31	91.80±0.43	80.14±0.27
Evo-OSNet	**91.88 ± 0.44**	**78.56 ± 0.33**	**80.20 ± 0.43**	**94.24 ± 0.19**	**92.58 ± 0.25**	**81.02 ± 0.16**

from scratch until convergence and test them on the corresponding test datasets. The associate parameters are described in Section III in the supplementary material. The comparative results on the MedMNIST series datasets are presented in Table VI. In comparison, the results of the peer competitors are extracted from [64] that originally introduced the MedMNIST series datasets, including manually designed state-of-the-art model, i.e., ResNet, and commercial AutoML systems, e.g., Auto-sklearn [71], AutoKeras [72], and Google AutoML Vision [73]. Overall, the architectures generated by Evo-OSNAS can achieve better performance. For example, the performance enhancement is larger than 5% compared to AutoKeras on PathMINIST, and the performance enhancement is larger than 4% compared to ResNet-18 on OCTMNIST. Moreover, we also run SI-EvoNAS [30] with five different random seeds on these datasets to get the statistical results. Evo-OSNAS also surpasses SI-EvoNAS on all medical image analysis tasks. More details of the statistical results of Evo-OSNAS on these classification datasets are provided in Section V-A in the supplementary material. These experimental results further demonstrate the ability of the proposed method to design high-quality task-dependent models.

VII. CONCLUSION

This article proposed an Evo-OSNAS framework that reduces, to a certain degree, the negative impact of weight coupling in the one-shot model and directly generates neural architectures on target classification tasks without any proxy metrics. This objective has been achieved by proposing a new node representation scheme for neural network architectures, together with corresponding crossover and mutation operators. To further improve the accuracy and efficiency in evolving large network architectures, we encoded the PyConv operations into search space, which mixes multiple convolutional kernels in a single operation to utilize different kernel sizes. The experiments on four benchmark classification tasks, including CIFAR10, CIFAR100, CINIC10, and ImageNet, have demonstrated the effectiveness of Evo-OSNAS. The proposed algorithm is also validated on real-world medical image analysis classification tasks. Finally, we analyzed the generalization and transferability performance of the designed architectures. Our experimental results demonstrate that the proposed Evo-OSNAS can obtain promising classification performance compared with state-of-the-art NAS and OSNAS algorithms.

Although Evo-OSNAS can efficiently find a complete and high-performance neural network architecture, the weakness it has is, similar to most existing centralized NAS approaches, that it may no longer be applicable to the federated learning environment because data may be non-identically distributed (non-IID) and the raw data cannot be accessed by the central server. In addition, as deep network architectures contain millions of weights, the offline NAS method requires a large number of communication budgets. Hence, developing a lightweighted encryption method for masking the gradient information is of great importance to prevent privacy leakage and reduce communication costs.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.
[2] R. Girshick, "Fast R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 1440–1448.
[3] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 3431–3440.
[4] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.
[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[6] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 4700–4708.
[7] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Completely automated CNN architecture design based on blocks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 4, pp. 1242–1254, Apr. 2020.
[8] M. Zhang *et al.*, "One-shot neural architecture search: Maximising diversity to overcome catastrophic forgetting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 9, pp. 2921–2935, Sep. 2021.
[9] F. E. Fernandes and G. G. Yen, "Automatic searching and pruning of deep neural networks for medical imaging diagnostic," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 12, pp. 5664–5674, Dec. 2021.
[10] Z. Lu *et al.*, "Multiobjective evolutionary design of deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, vol. 25, no. 2, pp. 277–291, Apr. 2021.
[11] Z. Yu, J. Wan, Y. Qin, X. Li, S. Z. Li, and G. Zhao, "NAS-FAS: Static-dynamic central difference network search for face anti-spoofing," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 9, pp. 3005–3023, Sep. 2021.
[12] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 394–407, Apr. 2020.
[13] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing CNN architectures using the genetic algorithm for image classification," *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3840–3854, Sep. 2020.

[14] Z. Zhong *et al.*, "BlockQNN: Efficient block-wise neural network architecture generation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 7, pp. 2314–2328, Jul. 2021.

[15] C. Liu *et al.*, "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 19–35.

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

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

[18] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," 2017, *arXiv:1705.10823*.

[19] Y. Sun, H. Wang, B. Xue, Y. Jin, G. G. Yen, and M. Zhang, "Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 350–364, Apr. 2020.

[20] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," 2018, *arXiv:1802.03268*.

[21] Z. Guo *et al.*, "Single path one-shot neural architecture search with uniform sampling," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 544–560.

[22] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "SMASH: One-shot model architecture search through HyperNetworks," 2017, *arXiv:1708.05344*.

[23] A. Zela, A. Klein, S. Falkner, and F. Hutter, "Towards automated deep learning: Efficient joint neural architecture and hyperparameter search," 2018, *arXiv:1807.06906*.

[24] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8697–8710.

[25] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–13.

[26] Y. Xu *et al.*, "Partially-connected neural architecture search for reduced computational redundancy," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 9, pp. 2953–2970, Sep. 2021.

[27] Y. Benyahia *et al.*, "Overcoming multi-model forgetting," 2019, *arXiv:1902.08232*.

[28] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. V. Le, "Understanding and simplifying one-shot architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 550–559.

[29] Y. Zhang *et al.*, "Deeper insights into weight sharing in neural architecture search," 2020, *arXiv:2001.01431*.

[30] H. Zhang, Y. Jin, R. Cheng, and K. Hao, "Efficient evolutionary search of attention convolutional networks via sampled training and node inheritance," *IEEE Trans. Evol. Comput.*, vol. 25, no. 2, pp. 371–385, Apr. 2021.

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

[32] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015, *arXiv:1502.03167*.

[33] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, 2016, pp. 2818–2826.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 630–645.

[35] J. D. Schaffer, D. Whitley, and L. J. Eshelman, "Combinations of genetic algorithms and neural networks: A survey of the state of the art," in *Proc. Int. Workshop Comb. Genet. Algorithms Neural Netw. (COGANN)*, Baltimore, MD, USA, 1992, pp. 1–37.

[36] X. Yao, "Evolving artificial neural networks," *Proc. IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.

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

[38] B. Inden, Y. Jin, R. Haschke, and H. Ritter, "Evolving neural fields for problems with large input and output spaces," *Neural Netw.*, vol. 28, pp. 24–39, Apr. 2012.

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

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

[41] Y. Jin and B. Sendhoff, "Pareto-based multiobjective machine learning: An overview and case studies," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 38, no. 3, pp. 397–415, May 2008.

[42] Z. Lu *et al.*, "NSGA-Net: Neural architecture search using multi-objective genetic algorithm," in *Proc. Genet. Evol. Comput. Conf.*, 2019, pp. 419–427.

[43] Y.-H. Kim, B. Reddy, S. Yun, and C. Seo, "Nemo: Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy," in *Proc. ICML AutoML Workshop*, 2017.

[44] Z. Lu, G. Sreekumar, E. Goodman, W. Banzhaf, K. Deb, and V. N. Boddeti, "Neural architecture transfer," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 9, pp. 2971–2989, Sep. 2021.

[45] H. Zhu and Y. Jin, "Multi-objective evolutionary federated learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 4, pp. 1310–1322, Apr. 2020.

[46] H. Zhu, H. Zhang, and Y. Jin, "Multi-objective evolutionary federated learning," *Complex Intell. Syst.*, to be published.

[47] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: Stochastic neural architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–17.

[48] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in *Proc. IEEE Int. Conf. Comput. Vis.*, Seoul, South Korea, 2019, pp. 1294–1303.

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

[50] M. Tan and Q. V. Le, "MixConv: Mixed depthwise convolutional kernels," 2019, *arXiv:1907.09595*.

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

[52] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Salt Lake City, UT, USA, 2018, pp. 4510–4520.

[53] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.

[54] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Honolulu, HI, USA, 2017, pp. 2117–2125.

[55] I. C. Duta, L. Liu, F. Zhu, and L. Shao, "Pyramidal convolution: Rethinking convolutional neural networks for visual recognition," 2020, *arXiv:2006.11538*.

[56] X. Zheng, R. Ji, L. Tang, B. Zhang, J. Liu, and Q. Tian, "Multinomial distribution learning for effective neural architecture search," in *Proc. IEEE Int. Conf. Comput. Vis.*, Seoul, South Korea, 2019, pp. 1304–1313.

[57] M. Tan *et al.*, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Long Beach, CA, USA, 2019, pp. 2820–2828.

[58] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Honolulu, HI, USA, 2017, pp. 1492–1500.

[59] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Salt Lake City, UT, USA, 2018, pp. 6848–6856.

[60] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 122–138.

[61] K. Deb and R. B. Agrawal "Simulated binary crossover for continuous search space," *Complex Syst.*, vol. 9, no. 2, pp. 115–148, 1995.

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

[63] L. N. Darlow, E. J. Crowley, A. Antoniou, and A. J. Storkey, "CINIC-10 is not ImageNet or CIFAR-10," 2018, *arXiv:1810.03505*.

[64] J. Yang, R. Shi, and B. Ni, "MedMNIST classification decathlon: A lightweight AutoML benchmark for medical image analysis," 2020, *arXiv:2010.14925*.

[65] Y. Chen *et al.*, "RENAS: Reinforced evolutionary neural architecture search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Long Beach, CA, USA, 2019, pp. 4787–4796.

[66] H. Zhou, M. Yang, J. Wang, and W. Pan, "BayesNAS: A Bayesian approach for neural architecture search," 2019, *arXiv:1905.04919*.

[67] X. Dong and Y. Yang, "One-shot neural architecture search via self-evaluated template network," in *Proc. IEEE Int. Conf. Comput. Vis.*, Seoul, South Korea, 2019, pp. 3681–3690.

[68] X. Chu, B. Zhang, and R. Xu, "FairNAS: Rethinking evaluation fairness of weight sharing neural architecture search," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 12239–12248.
[69] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2011, p. 5.
[70] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," 2015, *arXiv:1511.07122*.
[71] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran, 2015, pp. 2962–2970.
[72] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, 2019, pp. 1946–1956.
[73] B. Zoph, V. Vasudevan, J. Shlens, and Q. Le, *AutoML for Large Scale Image Classification and Object Detection*, vol. 2, Google AI Blog, Mountain View, CA, USA, 2017, p. 2017.

Haoyu Zhang received the M.Sc. degree from the Inner Mongolia University of Technology, Hohhot, China, in 2018. He is currently pursuing the Ph.D. degree, with a focus on deep learning, object detection, and evolutionary neural architecture search, with the College of Information Science and Technology, Donghua University, Shanghai, China.

Yaochu Jin (Fellow, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in automatic control from Zhejiang University, Hangzhou, China, in 1988, 1991, and 1996, respectively, and the Dr.-Ing. degree from Ruhr-University Bochum, Bochum, Germany, in 2001.

He is currently an Alexander von Humboldt Professor for Artificial Intelligence, the Chair of Nature Inspired Computing and Engineering, Faculty of Technology, Bielefeld University, Bielefeld, Germany, and a Distinguished Chair Professor of Computational Intelligence with the Department of Computer Science, University of Surrey, Guildford, U.K. He was a Finland Distinguished Professor funded by the Finnish Funding Agency for Innovation (Tekes) and a Changjiang Distinguished Visiting Professor appointed by the Ministry of Education, China. His main research interests include data-driven surrogate-assisted evolutionary optimization, secure machine learning, deep learning, swarm robotics, and evolutionary developmental systems.

Dr. Jin is the recipient of the 2018 and 2021 IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION Outstanding Paper Award, and the 2015, 2017, and 2020 *IEEE Computational Intelligence Magazine* Outstanding Paper Award. He is named as a Highly Cited Researcher for 2019, 2020, and 2021 by the Web of Science Group. He is currently the Editor-in-Chief of *Complex & Intelligent Systems*, and was the Editor-in-Chief of the IEEE TRANSACTIONS ON COGNITIVE AND DEVELOPMENTAL SYSTEMS. He is a member of Academia Europaea.

Kuangrong Hao (Member, IEEE) received the B.S. and M.S. degrees in mechanical engineering from the Hebei University of Technology, Tianjin, China, in 1984 and 1989, respectively, the M.S. degree in applied mathematics and computer sciences from the École Normale Supérieure de Cachan, Cachan, France, in 1991, and the Ph.D. degree in applied mathematics and computer sciences from the École Nationale des Ponts et Chaussées, Paris, France, in 1995.

She is currently a Professor with the College of Information Sciences and Technology, Donghua University, Shanghai, China. She has published more than 200 technical papers and five research monographs. Her scientific interests include intelligent perception, intelligent systems and network intelligence, robot control, and intelligent optimization of textile industrial process.