Efficient Evolutionary Search of Attention Convolutional Networks via Sampled Training and Node Inheritance

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

***Abstract*—The performance of deep neural networks is heavily dependent on its architecture and various neural architecture search strategies have been developed for automated network architecture design. Recently, evolutionary neural architecture search (EvoNAS) has received increasing attention due to the attractive global optimization capability of evolutionary algorithms. However, EvoNAS suffers from extremely high computational costs because a large number of performance evaluations are usually required in evolutionary optimization and training deep neural networks is itself computationally very expensive. To address this issue, this paper proposes a computationally efficient framework for evolutionary search of convolutional networks based on a directed acyclic graph, in which parents are randomly sampled and trained on each mini-batch of training data. In addition, a node inheritance strategy is adopted so that the fitness of all offspring individuals can be evaluated without training them. Finally, we encode a channel attention mechanism in the search space to enhance the feature processing capability of the evolved neural networks. We evaluate the proposed algorithm on the widely used datasets, in comparison with 30 state-of-the-art peer algorithms. Our experimental results show the proposed algorithm is not only computationally much more efficient, but also highly competitive in learning performance.**

***Index Terms*—Evolutionary optimization, neural architecture search, node inheritance, attention mechanism, convolutional neural networks.**

I. Introduction

DEEP convolutional neural networks (CNNs) has achieved remarkable success in solving various tasks such as image classification [1], speech recognition [2], natural language processing [3], among many others. Since the performance of deep neural networks heavily depends on their architectures, increasing research efforts in the deep learning community have been dedicated to the design of novel architectures, such as DenseNet [4], ResNet [5, 6], and VGG [7], among many others. In general, most powerful deep networks were manually designed by human experts who have extensive expertise in both deep learning and the related problem domain. Not until recently has automated neural architecture design, i.e., neural architecture search (NAS), shown great opportunity to allow interested users without adequate domain knowledge to benefit from the success of deep neural networks [8].

深度学习发展迅猛

An NAS task can generally be formulated as a complex optimization problem [9, 10]. In the field of computational intelligence, evolutionary algorithms (EAs) [11] have widely been used to solve various neural network training problems [12], such as weight training [13], architecture design [14], and learning rule adaptation [15]. Most recently, evolutionary neural architecture search employing an EA as the optimizer for NAS received increasing attention [16–19]. Despite their competitive search performance on a variety of optimization tasks [20–23], EAs generally suffer from high computational costs as a class of population-based optimization methods. This is particular true for EvoNAS since EAs typically require a large number of fitness evaluations, and each fitness evaluation in NAS is computationally expensive as it usually involves the training of a deep neural network from scratch on a large amount of data. For example, it takes 27 GPU days for AE-CNN [19] to obtain an optimized CNN architecture on CIFAR10 dataset.

NAS是一个优化问题，但是需要很强的计算能力去做

Therefore, various techniques have been suggested in EvoNAS to reduce the computational costs without seriously degrading the optimization performance. In [24], Bayesian optimization [25] is used to speed up evolutionary optimization, which is called Freeze-thaw Bayesian optimization. The main idea is to build a model to predict the performance based on the training performance in the previous epochs. Unfortunately, this algorithm is based on Markov chain Monte Carlo sampling and also suffers from high computational complexity. Recently, Sun et al. proposed a surrogate-assisted EvoNAS termed E2EPP, which is based on a class of surrogate-assisted evolutionary optimization [18, 26–28] that was meant for data-driven evolutionary optimization of expensive engineering problems. Specifically, E2EPP builds a surrogate that can predict the performance of a candidate CNN, thereby avoiding the training of a large number of neural networks during the EvoNAS. Compared with AE-CNN, a variant of AE-CNN assisted by E2EPP (called AE-CNN+E2EPP) can reduce 214% and 230% GPU days on CIFAR10 and CIFAR100, respectively. However, AE-CNN+E2EPP still requires 17 GPU days on both CIFAT10 and CIFAR100 to achieve its best

降低EvoNAS的计算要求的方法

H. Zhang, Y. Jin and K. 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. Email: zhy920816@sina.cn; krhao@dhu.edu.cn

Y. Jin is with the Department of Computer Science, University of Surrey, Guildford, Surrey GU2 7XH, United Kingdom. He is also with the Engineering Research Center of Digitized Textile & Apparel Technology, Ministry of Education, College of Information Science and Technology, Donghua University, Shanghai 201620, China. Email: yaochu.jin@surrey.ac.uk. (*Corresponding author*)

R. Cheng is with the Shenzhen Key Laboratory of Computational Intelligence, University Key Laboratory of Evolving Intelligent Systems of Guangdong Province, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China. Email: chengr@sustech.edu.cn.

Manuscript received February 14, 2020; revised xxxx, 2020.

results since training sufficiently accurate surrogates can still be computationally expensive since it requires to train a large number of deep neural networks.

In addition to EvoNAS, many algorithms use reinforcement learning (RL) to perform NAS [8, 29, 30]. A major limitation of these methods is that RL based methods are computationally also demanding despite their remarkable performance. Recent developed gradient-based (GD) methods, e.g., differentiable architecture search (DARTS) [31], that use the relaxation tricks to make the weighted sum of candidate layers differentiable, and then apply the gradient descent method to train the weights. However, such GD-based NAS methods require excessive GPU memory during the search. As a result, the size of the search space is highly constrained.

介绍了GD和RL两种方法的不足

Apart from automated neural architecture design using EvoNAS, other methodologies such as attention mechanisms [32] have been investigated to improve the performance of deep neural networks such as convolutional neural networks. Inspired by the human visual system, attention mechanisms aim to recognize objects by selecting the key parts of an object instead of the whole object [33, 34]. Consequently, introducing an attention mechanism into CNNs can bring better discriminative feature representation capability [35]. In general, attention mechanism based methods can be divided into two categories. The first category includes methods focusing on channel attention, such as SE-Net [36], ECA-Net [37], while the second category refers to methods based on spatial attention, such as spatial transformer networks [38], and deep recurrent attentive writer (DRAW) neural network [39]. Compared with spatial attention, channel attention can be more easily incorporated into CNNs for them to be trained end-to-end. The principle of the channel attention mechanism is to reconstruct channel-wise features, i.e., assigning a new weight to each channel to make the feature response of the key channel stronger, so that it can learn to selectively emphasize significant informative features and penalize excessive redundant features [36]. However, most previous work merely constructed attention mechanisms by manually stacking multiple attention modules into neural networks. Such naive stacking attention modules may lead to poor performance of the network [35].

自动优化空间注意力机制

To improve the efficiency of EvoNAS, this paper proposes a fast EvoNAS framework based on sampled training of the parent individuals together with a node inheritance strategy, called SI-EvoNAS, which significantly reduces the computational costs. The main contributions of this paper are summarized below:

- A sampling strategy is proposed to train the individuals in the parent population. Specifically, for each mini-batch iteration, a parent individual is randomly chosen and trained on the mini-batch of training data. Since the number of training iterations in an epoch is usually much larger than the population size, each individual in the parent population will be trained on many mini-batches of the training data at each generation, efficiently improving the training performance, while still being able to reduce the computational complexity.

1提出采样策略
2是什么
3为什么 更有效率的评估个体

- A node inheritance strategy is proposed to generate offspring individuals by applying a one-point crossover and an exchange mutation. This way, offspring individuals directly inherit the parameters from their parents and none of them needs to be trained at all for evaluating their fitness value.

节点继承评估个体适应性不需要训练

- A multi-scale channel attention mechanism is incorporated into neural architecture search. As a result, channelwise features can be represented with respect to the spatial information on different scales.

优化通道注意力机制

- A large neural network is generated by repeating the block structures of the neural architecture found by the proposed algorithm on CIFAR10 for three times, which is able to show highly competitive performance compared to state-of-the-art architectures on the ImageNet dataset.

如何构建整个网络

The rest of this paper is organized as follows. Section II introduces the background of this work. Section III describes the encoding of neural network architectures based on directed acyclic graphs (DAGs) and a channel attention mechanism, followed by the details of the proposed algorithm in Section IV. Experimental settings and results are presented in Section V and Section VI, respectively. Finally, Section VII concludes the paper.

II. Related Work

In this section, we briefly review three widely used search methods for NAS, namely evolutionary algorithms, reinforcement learning, and gradient-based methods.

A. NAS based on EAs

EAs have already been used for simultaneous optimization of the topology, weights and hyperparameters of neural networks in late 1990s. The neuroevolution with augmenting topologies (NEAT) algorithm [40] was among the first algorithms that have showed powerful performance. Over the past years, EAs have become gradually popular again in NAS of deep neural network models. For example, Sun et al. [19] use an EA with a variable coding length to automatically evolve the architecture of CNNs. William et al. [41] introduce a DAG-based encoding strategy for evolutionary NAS, which is shown to outperform randomly generated CNN architectures. Real et al. propose Amoebanet [42], which uses an improved tournament selection to evolve a population of networks, and has achieved better results on ImageNet compared with handcrafted models. Wang et al. [43] devise an efficient evolutionary algorithm to optimize generators within GANs framework. This method can effectively improve the performance of generative performance and the training stability of GAN models. Sun et al. [44] propose a variable-length encoding that can represent a different number of building blocks and different numbers of layers in search for optimal deep convolutional neural network. In addition, the weights of the offspring solutions are initialized based on an optimized Gaussian distribution, thereby reducing the amount of required computation resources for fitness evaluations.

In addition, multi-objective NAS methods considering multiple conflicting objectives have been reported. One of the

在20世纪90年代末，EAs已经被用于同时优化神经网络的权重、拓扑和超参数。具有增强拓扑（NEAT）算法[40]是首批表现出强大性能的算法之一。在过去的几年里，EAs在深度神经网络模型的NAS中又逐渐流行起来。例如，Sun等人[19]使用具有可变长度编码长度的EA来自动演化CNN的体系结构。William等人[41]介绍了一种基于DAG的进化NAS编码策略，该策略的性能优于随机生成的CNN架构。Real等人提出了变形虫网[42]，它使用改进的锦标赛选择来发展网络群体，并与手工制作的模型相比，在ImageNet上取得了更好的结果。Wang等人[43]设计了一种有效的进化算法，以在GANs框架内优化发生器。该方法可以有效地提高生成性能和训练稳定性。Sun等人[44]提出了一种可变长度编码，可以表示不同数量的构建块和不同数量的层，以搜索最佳的深度卷积神经网络。此外，基于优化的高斯分布初始化子解的权重，从而减少适应度评估所需的计算资源量。

设计CNN架构的最早进化多目标方法是NEMO【45】，它使用NSGA-II最大化分类性能并最小化网络的推理时间。Elsken等人[46]将NAS描述为一个双目标优化问题，其中两个目标是性能最大化和计算资源最小化。Lu等人[47]提出了NSGANet，它可以自动设计网络，以最大限度地提高模型性能，并最小化同时进行浮点数操作（FLOPs）。[48]中还提出了进化多目标神经结构优化，以降低联邦学习中的通信成本。注意，浅层网络的进化多目标结构优化可以追溯到十年前[49]。

earliest evolutionary multi-objective methods to design CNN architectures is NEMO [45], which uses NSGA-II to maximize the classification performance and minimize the inference time of a network. Elsken et al. [46] formulate the NAS as a bi-objective optimization problem, where the two objectives are maximization of performance and minimization of computational resources. Lu et al. [47] present the NSGA-Net, which automatically design networks to maximize the model performance and minimize the floating point operations (FLOPs) at the same time. Evolutionary multi-objective neural architecture optimization to reduce communication cost in federated learning is also suggested in [48]. Note that evolutionary multi-objective structure optimization of shallow networks can be traced back to a decade ago [49].

One major drawback of EvoNAS is that during the evolutionary optimization, each new candidate neural network needs to be trained on a training dataset and then evaluated on a validation dataset to avoid overfitting. Hence, fitness evaluations in EvoNAS may take hours if the network is large and if the training dataset is huge. Since EAs are a class of population-based search methods, they often require a large number of fitness evaluations, making EvoNAS computationally prohibitive. For instance, on the CIFAR10 and CIFAR100 datasets, CNN-GA [50] consumed 35 GPU days and 40 GPU days, respectively, the genetic CNN method (Genetic CNN) [14] spent 17 GPU days, and the large-scale evolutionary algorithm [16] consumed 2750 GPU days. Therefore, it is essential to accelerate fitness evaluations in EvoNAS when computational resource is limited.

EvoNAS的一个主要缺点是，在进化优化过程中，每个新的候选神经网络都需要在训练数据集上进行训练，然后在验证数据集上进行评估，以避免过度拟合。因此，如果网络庞大，训练数据集庞大，EvoNAS中的体能评估可能需要数小时。由于EAs是一类基于群体的搜索方法，它们通常需要大量的适应度评估，这使得EvoNAS在计算上难以实现。例如，在CIFAR10和CIFAR100数据集上，CNN-GA[50]分别消耗35 GPU天和40 GPU天，遗传CNN方法（遗传CNN）[14]消耗17 GPU天，大规模进化算法[16]消耗2750 GPU天。因此，在计算资源有限的情况下，加速EvoNAS中的适应度评估至关重要。

B. RL based NAS

A large body of research on NAS adopts RL for the search of a good network architecture [30, 51]. In the above work, Q-learning is used as a value iteration method and a recurrent neural network (RNN) as a controller to sample new networks from the search space, then utilizes the network performance as the reward. And then the reward is used to update the RNN controller so that better candidate networks can be sampled in the next iteration. For example, BlockQNN [30] uses a Q-Learning paradigm with epsilon-greedy exploration strategy to build high-performance architecture. Zoph et al. [8] employs a policy gradient method in NAS to train a RNN controller that constructs networks. Although the RL-based methods can yield good models, they are also computationally very intensive. To reduce computational costs, various approaches have been suggested to improve the search efficiency of the NAS [52, 53]. Pham et al. [52] propose a parameter sharing method that views all network candidates (sub-models) as different subgraphs of a large-graph (also denoted as the one-shot model). Hence, all sub-models can share a common set of weights. As a result, it takes less than 16 hours for the parameter sharing method to search an optimal architecture on the CIFAR10 dataset.

C. GD based NAS

There is a growing interest in using GD-based methods for NAS. Liu et al. [31] implement this idea in the DARTS algorithm. Subsequent approaches include differentiable architecture sampler (GDAS) [54] and stochastic NAS (SNAS) [55]. GDAS develops a learnable differentiable sampler, which samples individual architecture in a differentiable way to accelerate the architecture search procedure. SNAS optimizes a probability distribution of the connections between different candidate operations. ProxylessNAS [56] binarizes the model parameters to cope with the linear increase in usage GPU memory.

There are also other NAS methods that do not fully fall in the above categories. Cai et al. [57] reuse the parameters of the current network with the Net2Net method [58]. Zheng et al. [59] consider the architecture search space as a joint multinomial distribution such that the operation between two nodes can be sampled from this distribution. And then the designed model is obtained by connecting the operations with the most likely probability in the distribution. However, the network architectures generated by these algorithms are still dataset-specific or do not consider applications under resource constraints.

III. Architecture Search Space

The search space has a major influence on the quality of designed architecture and the search efficiency. In this section, we describe the basic designs and properties of the proposed search space.

Deep CNN architectures, e.g., Inception [60, 61] and ResNet and its variances, are often designed by stacking several blocks to construct a neural network. The network structure design includes the determination of the depth (the number of layers), the width (the number of channels), and the spatial resolution change (the number of pooling layer), while the block structure design specifies the layer-wise connections and local computations. By means of such block-wise design methods, the generated model can not only achieve high performance, but also be able to generalize to different datasets and tasks. Therefore, we follow the same block-level design methodology as in [8, 31, 59], which will be elaborated in the following subsection.

A. Block structure

As illustrated in Figs.1 (a) and (b), a block is a small convolutional network, which can be represented by a DAG. Specifically, a node represents the basic components of a block, and an edge between two nodes denotes the information flow, e.g., a feature map in CNNs. To process different intermediate information more efficiently in the forward propagation, three types of convolutional blocks are designed according to different grid sizes of the feature maps: convolutional block 1, convolutional block 2, convolutional block 3. The convolutional blocks output information with a grid size that is the same as that of the input. Meanwhile, the reduction block is designed to increase the receptive field of the deeper layers and halve the grid size of the feature maps by applying all operations with a stride of 2. Following the conventions from the modern CNN architectures [5, 60, 61], we double the number of channels (filters) of the block when the grid size

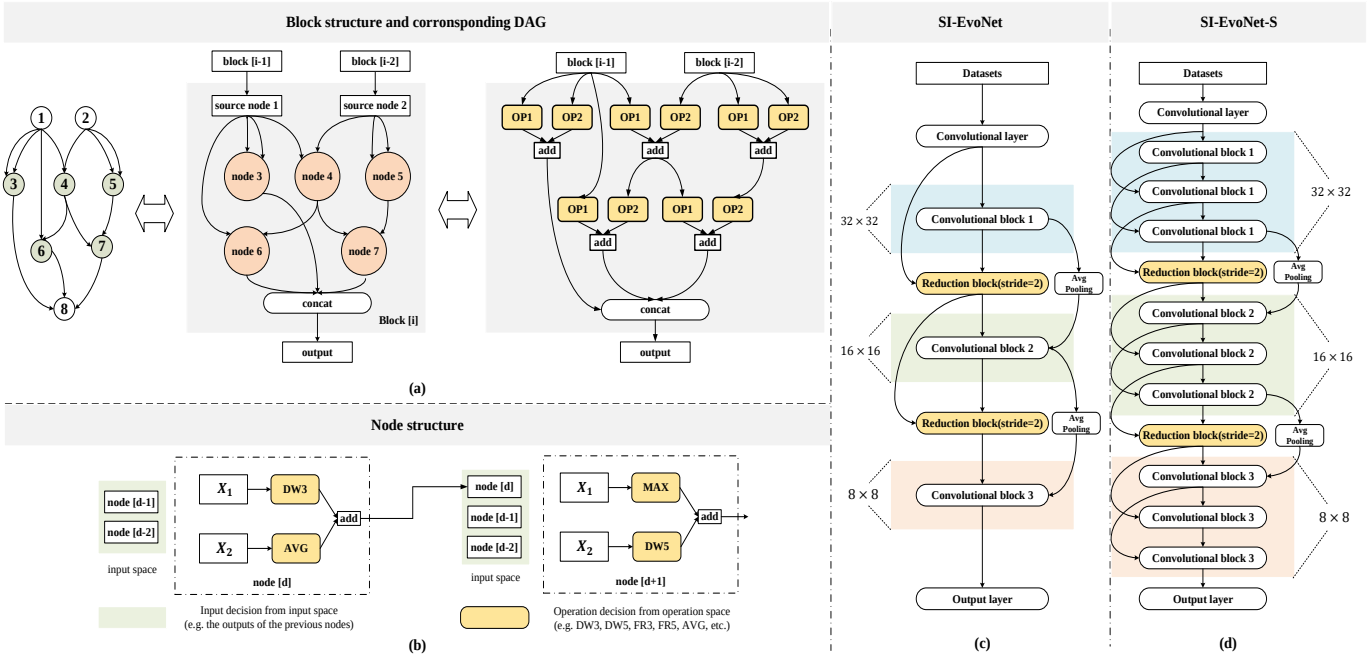

Fig. 1. Search space of SI-EvoNAS: a) An example block represented by a DAG with 8 nodes including two source nodes (node 1 and node 2), five computation nodes (node 3, 4, 5, 6, 7), and the output node (node 8). The output node concatenates the computation nodes. b) The structure of a computation node represented by a binary tree, where two computation operations (e.g., depthwise separable convolutional layer or average pooling layer) are applied on two inputs (X_1, X_2), and then combined by a merging operation. c) The SI-EvoNet consisting of 5 blocks. d) The SI-EvoNet-S made up of 11 blocks.

of the feature maps is halved to maintain a roughly constant hidden state dimension.

Each block contains three types of nodes: the source node, the computation node, and the output node. The source node is the block's input, which is the output of the previous blocks or the input of the overall network. The computation node is structured by a binary tree representation and ends with an element-wise addition operation. The results outputted from the addition operation is a new hidden state, which is available for the subsequent nodes in the same block. As illustrated in Figs. 1(a) and (b), a node d in block i is then depicted by a 5-tuple, (X_1, X_2, O_1, O_2, M), where, $X_1, X_2 \in \mathcal{X}$ specify the input of the current node, $O_1, O_2 \in \mathcal{O}$ specify the operation to be applied on the input tensor. M specifies the element-wise addition operation that sums up the two operation's results of a node to generate the feature map corresponding to the output of this node. The possible inputs $\mathcal{X}$ are composed of the set of hidden states generated in any of the previous nodes inside the block, the output of the previous block $i-1$ (source node 1) and the previous-previous block $i-2$ (source node 2). All nodes without a successor in the block are concatenated together to provide the final output of block.

$\mathcal{O}$ is the operation space consisting of a set of possible basic components of network architectures, known successful modules designed by human experts, and feature reconstruction (FR) convolutional operation. The nine available operations in $\mathcal{O}$ and corresponding genotype-phenotype mappings used in this work are presented in Table I. In the table, depthwise separable convolution (DW) and local binary convolution (LBC) are parameter-efficient layers, which are able to reduce network parameters without sacrificing the network performance.

TABLE I
NETWORK OPERATION CODE SPACE. THE SPACE CONTAINS SEVEN TYPES OF COMMONLY USED COMPUTATION OPERATION.

Operation type	Kernel size	Short name	Code
Identity mapping	–	Identity	1
Depthwise separable convolution	3	DW3	2
Depthwise separable convolution	5	DW5	3
FR convolution	3	FR3	4
FR convolution	5	FR5	5
Average pooling	3	AVG	6
Max pooling	3	MAX	7
Local binary convolution	3	LBC3	8
Local binary convolution	5	LBC5	9

Here we use two DW operations and two LBC operations with a kernel size 3×3 and 5×5, DW3 and DW5, LBC3, and LBC5 for short. More details about the FR will be given Subsection III.C.

B. From blocks to neural networks

Based on the above-defined blocks, we build the entire neural network by stacking several blocks sequentially with residual connections. As shown in Figs. 1 (c) and (d), respectively, we first construct a lightweight version of network, termed SI-EvoNet, used for architecture search. After that, we construct a standard version of the network, termed SI-EvoNet-S, to evaluate the performance of SI-EvoNAS. To accomplish this, we construct the SI-EvoNet-S by repeating each convolutional block discovered by search (the output of SI-EvoNAS) for three times and stacking these blocks sequentially. Meanwhile, to avoid the representational bottleneck as discussed in [61], we use two parallel modules, i.e., a reduction

block and an average pooling layer, to decrease the grid size of the feature maps. This design is motivated by successful hand-crafted CNN architectures, e.g., Inception-V3 and Inception-V4, which design different block structures for the feature maps of different resolutions in CNNs. Finally, the output layers of the designed architectures consist of a global average pooling layer (GAP) followed by a softmax layer.

Although the block construction process is identical to that in [31, 59, 62], the construction of the overall network is different. Specifically, the search space in this work is an expanded version of the micro search space proposed in [8, 52]. In order to process intermediate information, our method designs a new convolutional block after down-sampling instead of repeating one normal block multiple times, as shown in Figs.1(c) and (d). This scheme overcomes the limitation of repeating the same normal block throughout the whole network, since one single normal block may not be optimal for feature maps with different resolutions.

C. Multi-scale feature reconstruction convolutional operation

Inspired by the previous work on efficient channel attention mechanism [37], we propose a feature reconstruction convolutional operation as a building block in the neural architecture search, FR for short. FR consists of a dilated convolutional layer [63] followed by an efficient channel attention module.

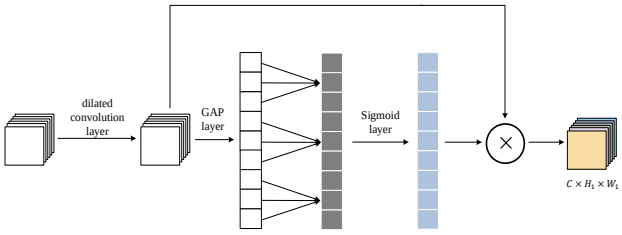

Fig. 2. The structure of the FR convolutional operation. An array of the square block represents the input features. The dilated convolutional layer expands the input feature maps from $x \in \mathbb{R}^{C\times H\times W}$ to feature maps $T(x) \in \mathbb{R}^{C\times H_1\times W_1}$. After a GAP layer is the channel attention module followed by a $\sigma_{sigmoid}$ function layer that computes weights using a $1D$ convolutional of size θ. The rectangles represent feature maps of channels. Gray rectangles represent the output of $1D$ convolution. Blue rectangles represent the weight of channels. An array of the colorful square block represents the output feature maps of the FR operation.

Fig. 2 plots the components of the FR convolutional operation. Specifically, given the input features, FR first uses a dilated convolutional layer to exponentially expand the receptive field without losing resolution or coverage, which contains more global information. Then a GAP layer is employed for each channel independently. After that, a $1D$ convolutional layer of size θ followed by a $\sigma_{sigmoid}$ function layer is utilized to generate the weight of each channel. The $1D$ convolutional layer of size θ is designed to capture the non-linear cross-channel interaction for each channel with its θ neighboring channels, where the kernel size θ represents how many neighboring channels take part in the attention prediction of the channel. Hyperparameter θ is adaptively determined by an exponential function proposed in [37].

IV. Proposed Algorithm

As discussed above, the search space of the proposed SI-EvoNAS is represented using a single DAG, where a neural network architecture can be realized by taking a subgraph of the DAG. Different connection relationships between the nodes will result in a large number of neural networks with different architectures. We use SI-EvoNAS to learn the connection relationship between nodes and to find better topologies for deep neural networks. Except for the source nodes, each node in a DAG represents some local computation, which is specified by the weights and bias. Since an offspring individual (a new neural network model) generated by applying genetic operations on parent individuals (existing neural network models in the parent population) can be seen as the recombination of the nodes of the parent models, the parameters of the offspring individual can be directly inherited from the parent networks. We call this method node inheritance. Moreover, because each node is repeatedly used throughout the evolutionary optimization, all parameters of the nodes are trained and updated in the evolutionary search process. Consequently, SI-EvoNAS is able to completely avoid training offspring individuals with the help of node inheritance, thereby effectively reducing the high computational costs usually required by EvoNAS.

A. Overall framework

Algorithm 1 The framework of SI-EvoNAS

Input: population size K, maximum number of generations G, training data D_{train}, validation data D_{valid}
Output: the best neural network architecture
1: $P_0 \leftarrow$ Initialize a population with a size of K by Algorithm 2
2: $t \leftarrow 0$
3: **while** $t < G$ **do**
4: $P_t, w_t \leftarrow$ Train individuals of P_t by Algorithm 3 on D_{train}
5: Evaluate the fitness of trained individuals in P_t on D_{valid}
6: Generate offspring $|Q_t| = K$ by Algorithm 4
7: Evaluate the fitness of individuals in Q_t on D_{valid}
8: $R_t \leftarrow P_t \cup Q_t$
9: p_{best},$\delta_{best} \leftarrow$ Select the individual with the best fitness from R_t
10: Set δ_{best} as the fitness of R_t
11: $P_{t+1} \leftarrow$ Select K individuals from R_t by environment selection
12: **if** p_{best} is not in P_{t+1} **then**
13: Replace the worst individual in P_{t+1} by p_{best}
14: **end if**
15: $t \leftarrow t + 1$
16: **end while**
17: Return the best individual from P_t and decode it into the corresponding neural network

Algorithm 1 lists the main components of the SI-EvoNAS. It starts with an initial population of P_0 consisting of K randomly generated individuals (line 1). Then, we repeat the following steps (lines 4-15) for G generations. Each generation is composed of training and evaluation of parent individuals (lines 4-5), generation and evaluation of offspring individuals (lines 6-7), combination of the parent and offspring populations (line 8), and environmental selection (lines 9-15).

First, an initial population is randomly generated with a given size of K (line 1). Then, each individual in the parent population P_t is trained by proposed sampled training method and then evaluated (lines 4-5). Subsequently, the binary tournament selection is adopted to select two parents and crossover

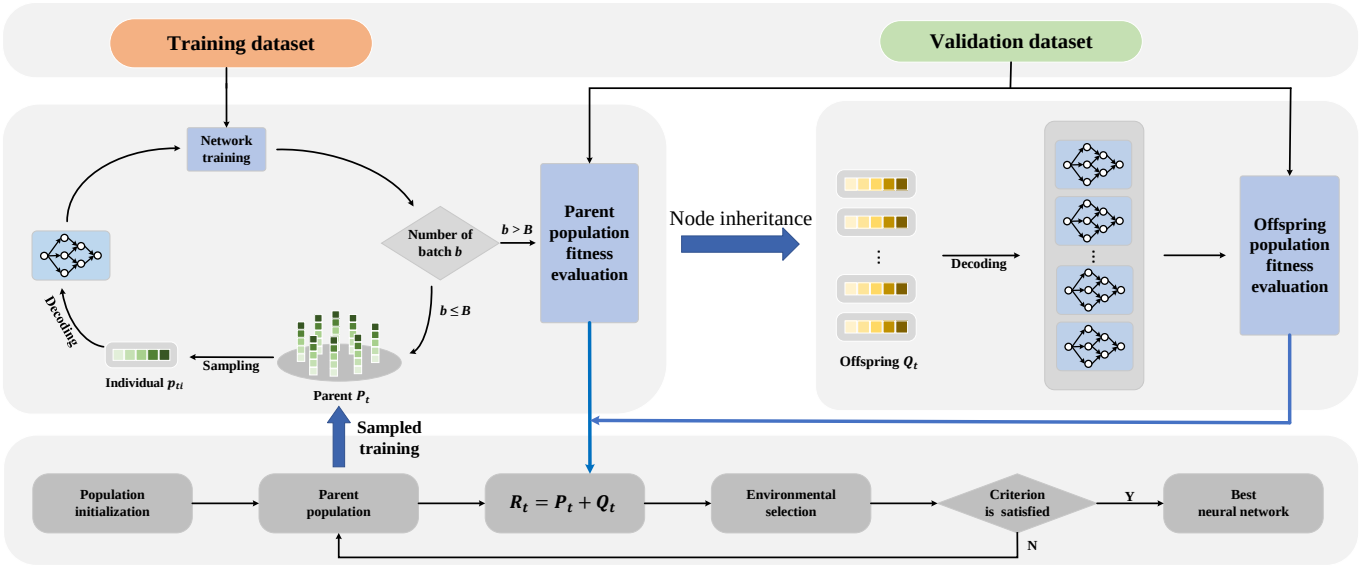

Fig. 3. The overall framework of SI-ENAS

and mutation are applied on the two selected parent individuals to generate two offspring. This process repeats until K offspring individuals are generated to form the offspring population Q_t. Fitness evaluation of the offspring population is achieved by the proposed node inheritance method, which will be detailed in Subsection IV.C. In the environmental selection, P_t and Q_t are merged to form a combined population R_t of size $2K$, from which the best K individuals are selected as the parent population of the next generation (P_{t+1}). Finally, SI-EvoNAS outputs the best individual and decodes it into the corresponding neural architecture (which is the optimal neural architecture found by the SI-EvoNAS), when the maximum number of generations is reached. This optimized neural network will be fully trained and then evaluated on the test data.

The framework of the overall SI-EvoNAS is illustrated in Fig. 3. Note that the proposed SI-EvoNAS is significantly different from most existing EvoNAS algorithms in that in SI-EvoNAS, the parent individuals are randomly sampled and trained on mini-batches of training data, while the offspring individuals are generated using a node inheritance strategy and do not need to be trained for fitness evaluations. This way, SI-EvoNAS is able to significantly reduce the computation time while avoiding biases towards either parent or offspring individuals.

In the following, we elaborate the main components of Si-EvoNAS.

B. Neural Architecture Encoding and Decoding

The encoding strategy defines the genotype-phenotype, which is required for an EA to be employed to optimize the architecture of neural networks. Here, the phenotypes are different neural network architectures and the genotypes are the genetic encoding. In our work, we employ a fixed-length integer encoding scheme, although both fixed or variable lengths have been used in EvoNAS [17, 19, 41]. As shown in Fig.4, each node is encoded by four integer genes. The first two numbers stand for the node indexes, which represent the inputs of the node. The last two represent the operation types of each input. The genes of different nodes are stacked sequentially to encode a block, and consequently the chromosome of a whole network, each consisting of four blocks, is composed of 80 genes.

The pseudocode for neural architecture encoding and population initialization is given in Algorithm 2. Encoding a neural network is composed of two steps. First, two integers are randomly selected from the input space $\mathcal{X}$ as the inputs to the current node (line 10). Then, two integers are also randomly selected from the operation space $\mathcal{O}$ as the corresponding operations (line 11). This process repeats until all computation nodes of an individual are configured.

Algorithm 2 Population initialization

Input: population size K, $block\ type\ index$, node space $\mathcal{X}$, operation space $\mathcal{O}$
Output: the initial population P_0

```
1: P_0 ← ∅
2: block type index = 4
3: node number index = 7
4: for k ← 1 to K do
5:     individual [k] ← ∅
6:     for b ← 1 to block type index do
7:         block code [b] ← ∅
8:         for s ← 3 to node number index do
9:             node code [s] ← ∅
10:            X_1,X_2 ← Randomly select two integers from 𝒳
11:            O_1,O_2 ← Randomly select two integers from 𝒪
12:            node code [s] ← X_1,X_2 ∪ O_1,O_2
13:            block code [b] ← block code [b] ∪ node code [s]
14:        end for
15:        individual [k] ← individual [k] ∪ block code [b]
16:    end for
17:    P_0 ← P_0 ∪ individual [k]
18: end for
19: Return P_0
```

To evaluate the performance of the evolved architectures, each chromosome (the genotype of an individual) is decoded

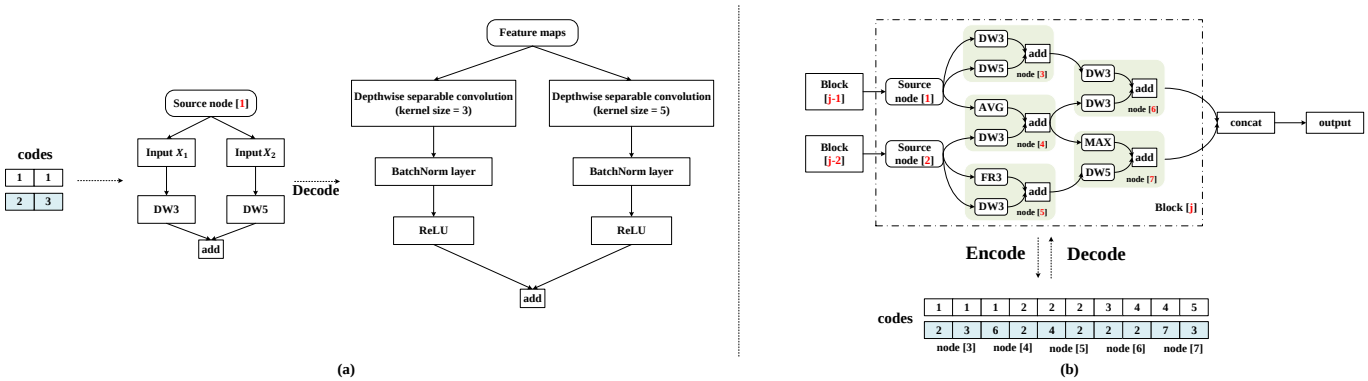

Fig. 4. a) An example of an encoded node and its corresponding decoded network structure. b) An example of a block structure and its corresponding codes

into a complete neural network architecture (phenotype). As shown in Fig.4, when decoding a neural network, we need to pay attention to the following practices of modern CNNs [5] : (i) a batch normalization operation followed by a ReLU activation function is added to the output of the convolutional layer; (ii) the zero-padding operation is used to make sure that the size of the input feature map is consistent with that of the output feature map. By repeating the above steps, we can decode each chromosome into different neural network.

C. Sampled training and node inheritance

Algorithm 3 Sampled training based node inheritance

Input: parent P_t, training data D_{train}, batch size b, initial learning rate (LR) r with a MultiStepLR schedule, objective function l
Output: The trained networks of P_t and corresponding parameters w

1: **if** generation = 0 **then**
2: **for** each $individual$ in parent P_t **do**
3: $net \leftarrow$ Decode $individual$ into the corresponding neural network
4: $w \leftarrow$Initialize the parameters w with He initialization [64] in net
5: **end for**
6: **else**
7: **for** each $batch$ data in D_{train} **do**
8: Randomly sample an $individual$ $\mathcal{P}$ from P_t
9: $net_{\mathcal{P}} \leftarrow$ Decode $individual$ $\mathcal{P}$ into the corresponding network
10: $[net_{\mathcal{P}}, w_{\mathcal{P}}] \leftarrow$ Load the parameters w of all computation nodes in $net_{\mathcal{P}}$
11: $\nabla w_{\mathcal{P}} \leftarrow$ Compute the gradient by ∂l / $\partial w_{\mathcal{P}}$
12: $w_{\mathcal{P}} \leftarrow w_{\mathcal{P}}$ - $r\nabla w_{\mathcal{P}}$
13: **end for**
14: **end if**
15: Return the trained parent P_t and corresponding parameters w

The main computational bottleneck of EvoNAS methods resides in the fitness evaluation. In traditional EvoNAS, each individual is decoded into the corresponding network and then trained on the given data, as done in a common CNN. Such a scheme is computationally prohibitive, seriously limiting the allowed maximum number of generations in evolution. To reduce the computational burden, low fidelity estimates (also denoted as proxy metrics) of the actual performance are commonly used. For example, the candidate networks are trained for a small number of iterations or on a subset of the training data. Although these approximation approaches can address the problem to a certain degree, the performance of the networks evolved by these approaches may be limited [65].

In this work, we aim to reduce the computational costs required by EvoNAS by taking advantage of the working mechanisms of evolutionary algorithms. On the one hand, the extraordinary large computational cost is caused by the fact that EAs are population-based search methods and at each generation, a number of candidate architectures (equals to the population size) needs to be trained. On the other hand, there are close relationships between the offspring and parent architectures. Based the above two observations, we propose a concept of node inheritance and a sampled training method, which are called sampled training based node inheritance.

The main idea of the sampled training is, for each mini-batch iteration, one individual in the parent population is randomly chosen and trained on this mini-batch data, which repeats until the maximum number of iterations for each epoch is reached. As a result, the number of training iterations needed at each generation equals the number of iterations in each epoch. Most importantly, each parent architecture will be trained on many mini-batches at each generation, due to the fact that the number of mini-batch iterations (B) is usually much larger than the population size (K) and the sampled training is also repeated for a large number of generations. For example, in this work, $K = 25$ and $B = 312$ for CIFAR10, and a maximum nubmer of 300 generations will be carried out. Once the training is completed, all parameters of computation nodes are updated and saved. After that, all parent individuals are tested on the validation dataset to calculate their fitness. The details of the proposed training method are described in Algorithm 4.

Note that the success of the sampled training strategy heavily relies on the encoding strategy adopted in this work and the node inheritance proposed here. By node inheritance, we mean that offspring neural network architectures are generated from parents using crossover, exchange mutation, and weight inheritance. Consequently, no new operations are generated during reproduction, which is different to most existing EvoNAS methods and plays a critical role in the success of the sampled training.

As shown in Algorithm 4, the first part is one-point crossover (lines 3-15). Two parents are selected from population P_t using the binary tournament selection to create two offspring by applying one-point crossover on the node and

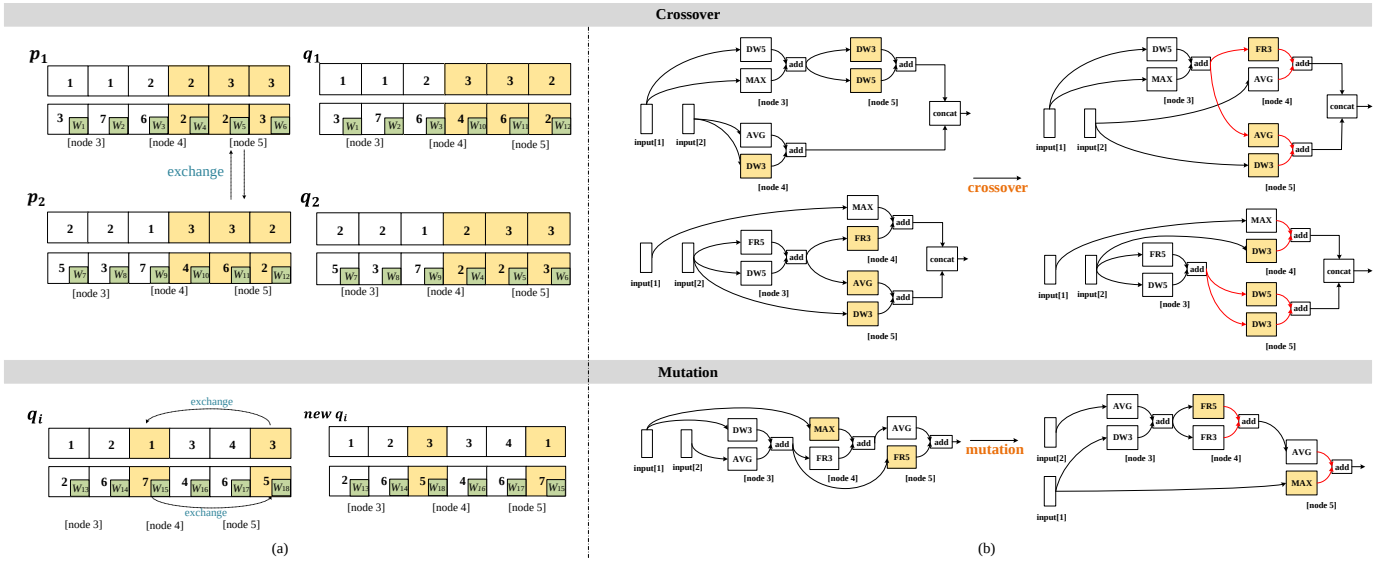

Fig. 5. An example of offspring generation by means of node inheritance, including crossover, exchange mutation and parameter inheritance. a) Given two parents, p_1 and p_2, two offspring individuals, q_1 and q_2 are created by a crossover between the last three genes. Then, individual q_i has a mutation by exchanging the third gene with the sixth gene. All parameters w_* are directly inherited from their parents. b) A graphical visualization of node inheritance.

Algorithm 4 Node inheritance

Input: population P_t, probability of crossover P_c, probability of mutation P_m.
Output: offspring population Q_t

1: $Q_t \leftarrow \varnothing$
2: **while** $|Q_t| < |P_t|$ **do**
3: p_1,$p_2 \leftarrow$ Select two individuals from population P_t by the binary tournament selection
4: $\gamma \leftarrow$ Uniformly generate a number from (0,1]
5: **if** $\gamma < P_c$ **then**
6: Randomly choose a position from p_1, p_2, respectively.
7: Separate p_1 and p_2 based on the chosen positions
8: $q_1 \leftarrow$ Combine the first part of p_1 and the second part of p_2
9: $q_2 \leftarrow$ Combine the first part of p_2 and the second part of p_1
10: **else**
11: $q_1 \leftarrow p_1$
12: $q_2 \leftarrow p_2$
13: **end if**
14: $Q_t \leftarrow Q_t \cup q_1 \cup q_2$
15: **end while**
16: **for** each individual q in Q_t **do**
17: $\gamma \leftarrow$ Uniformly generate a number from (0,1]
18: **if** $\gamma < P_m$ **then**
19: $q_i \leftarrow$ Randomly select two genes and swap their positions
20: **end if**
21: **end for**
22: **for** each $individual$ in Q_t **do**
23: $net_* \leftarrow$ Decode the individual * into the corresponding net_*
24: net_*, $w_* \leftarrow$ Reload the parameters trained in Algorithm 3 to each computation node in net_*
25: **end for**
26: Return Q_t

operation strings. This process is repeated until K offspring individuals are generated. The second part is the node/operation exchange mutation (lines 16-21), which can be seen as a type of mutation in which an individual randomly exchanges the order of two computation nodes or operations in its chromosome. Accordingly, the crossover is used to exchange substructures between two parent individuals, and then the mutation is applied to exchange the operations between two computation nodes within a single individual.

After the architecture of an offspring individual is generated by crossover based on its parents, we directly assign the trained parameters from its parents as the initial weights of the corresponding nodes in the offspring individual. Since no new nodes are generated in crossover and exchange mutation, all node parameters of the offspring individuals can be inherited from the parents, and therefore no random initialization of weights is needed. Subsequently, the offspring individuals are able to be directly evaluated without training on the validation dataset.

Figure 5 provides an illustrative example of offspring generation by means of node inheritance. Suppose p_1 and p_2 are selected by the tournament selection as the parents individuals, and then the position between the third and fourth genes is chosen for one-point crossover to generate the two offspring individuals, denoted by q_1 and q_2. Let w_* be the inheritable parameters that the offspring individuals receive. We can clearly see that all their operations are inherit from their parents. As shown in Fig. 5, offspring q_1 inherits three operations (indicated by white boxes) from p_1, and the rest three operations (yellow boxes) are inherited from p_2. Then, w_1,w_2, w_3 from p_1 and w_{10},w_{11}, w_{12} from p_2 are assigned to the corresponding operations of q_1. After crossover, an exchange mutation is applied on q_i, in which its third gene and sixth gene exchange their position. Meanwhile, the corresponding weights, w_{15} and w_{18}), are also exchanged.

As mentioned before, at each generation, each network is trained only on a subset of the mini-batches of the training data, however, all nodes of each network will have been trained on different mini-batches over the generations through sampled training of the parents and node inheritance. Therefore, the estimated performance of the evolved networks at the end of evolution is much more accurate than other proxy metrics, which will be experimentally confirmed in Subsection VI.B.

V. EXPERIMENTAL SETTINGS

The final goal of SI-EvoNAS is to efficiently find the optimal neural network architecture which achieves promising classification accuracy based on the benchmark dataset. To this end, a series of experiments are designed in this section to demonstrate the advantage of the proposed method compared to state-of-the-art. First, we evaluate the performance of the proposed algorithm by investigating the overall classification performance of the evolved neural networks. Second, we investigate the efficiency of the proposed node inheritance strategy in terms of the accuracy of the estimated fitness and the computational costs in comparison with four existing proxy-based EvoNAS algorithms. Following this, we examine the effectiveness of the proposed FR operation. Finally, we transfer the optimized network architecture evolved on CIFAR10 to CIFAR100 and SVHN to evaluate the transferability of the evolved network architecture.

We first perform the SI-EvoNAS algorithm over the SI-EvoNet. When the evolutionary process terminates, the best individual is enlarged to the SI-EvoNet-S, which is retrained from scratch and evaluated on the validation dataset. Finally, the trained SI-EvoNet-S is tested on the test dataset. The test classification accuracy is reported as the best result of our experiments, following the practice in deep learning. Note that the SI-EvoNet-S requires much more computation memories than SI-EvoNet. For example, it consumes more than 90% GPU memories in one run of the SI-EvoNet-S with a batch size of 96 on a computer with RTX 2080Ti GPU. Therefore, experiments for sensitivity analysis regarding the population size (Subsection V.D), comparison with parameter sharing (Subsection VI.C), the effectiveness of the FR operation (Subsection VI.D), and neural architecture transferring (Subsection VI.E), are all run based on the SI-EvoNet.

In the following, we introduce the peer competitors chosen to compare with the proposed algorithm, the used benchmark datasets, and finally the parameter settings of the SI-EvoNAS.

A. Peer competitors

In order to demonstrate the superiority of the proposed algorithm, various peer competitors are selected for comparison. The selected competitors are divided into three different groups.

The first group includes state-of-the-art CNN architectures which are manually designed by human experts, including DenseNet[4], ResNet(101) [5], Wide ResNet [66], ResNet (pre-activation) [6] VGG [7], SENet [36], MobileNetV2 [67], ShuffleNet [68], IGCV3-D ($G_1 = 4$, $G_2 = 2$) [69].

The second group comprises various non-evolutionary NAS methods (mainly RL or GD), such as NAS [8], MetaQNN [51], DPP-Net[70], Block-QNN-S [71], ENAS [52], Proxyless NAS [56], PNASNet [72], PPP-Net-A [73], MdeNAS [59], DARTs(first order) [31], RC-DARTS-C42 [62], RC-DARTS-C14 [62], SNAS [55] .

The third group represents state-of-the-art evolutionary NAS for CNN architecture design, including, Large-scale Evolution [16], Hierarchical evolution (Hier-evo) [74], AmoebaNet + cutout [42], LEMONADE evolution [46], NSGA-Net [47] CGP-CNN [17], CNN-GA [50], AE-CNN [19], and AE-CNN+E2EPP [18].

TABLE II
SUMMARY OF HYPER-PARAMETER SETTINGS

Categories	Parameters	Evolution	Evaluation
SI-EvoNAS	Initial channels	20 (CIFAR10) 32 (CIFAR100)	20 (CIFAR10) 32 (CIFAR100)
	Weight dacay	1×10^{-4}	3×10^{-4}
	Dropout	0.2	0.2
	Momentum	0.9	0.9
	Batch size	128	96
	Generation/Epoch	300	1000
	Initial LR	0.05	0.05
	LR schedule	MultiStepLR	Cosine Annealing

B. Benchmark datasets

In our experiments, the benchmark datasets include CIFAR10 [75], CIFAR100 [75], SVHN [76], ImageNet [77] datasets , which are all widely adopted in experimental studies of state-of-the-art CNNs and NAS algorithms.

CIFAR10 is a 10-category classification dataset consisting of 50,000 training datasets and 10,000 test datasets, and each image has a dimension of 32×32. CIFAR100 has the same number of images in the training datasets and test datasets as those of CIFAR10, except that it is 100-category classification problem. SVHN (Street View House Number) dataset is composed of 630,420, 32×32 RGB color images in total, of which 73,257 samples are used for training, 26,032 for validation, and the rest 531,131 for test.The task of this dataset is to classify the digits located at the center of each image. ImageNet [77] is a large-scale dataset including more than 1.4 million images with 1,000 different classes and each image has a much larger resolution. We employ the mobile setting, where the size of the input image size fixed to 224×224.

For CIFAR10 and CIFAR100, the first 80% of the training data is used as the training dataset and the remaining 20% is used for validation for calculating the fitness value. It should be noted that the test dataset is never used to guide the evolutionary search process. The details of the data augmentation procedure are provided in Section I of the Supplementary materials.

C. Parameter settings

In this subsection, the parameter settings for SI-EvoNAS are detailed.

Our algorithm parameters are divided into two parts: evolutionary search and best individual evaluation. During the evolutionary search phase, the parameter settings follow the practices in evolutionary computation. Although a larger population size and a larger number of generations will in principle lead to better performance, the computational costs will also become prohibitive. Hence, we investigate the impact of the population size and the maximum number of generations on the performance and computational cost of SI-EvoNAS in Subsection V.D. The probabilities of crossover and mutation are set to 0.95 and 0.05, respectively. In full training of the

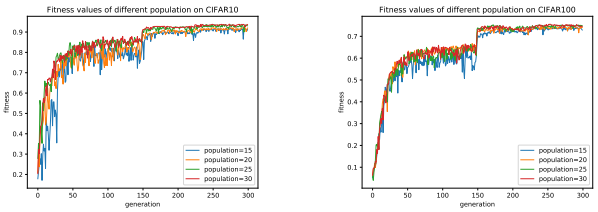

Fig. 6. The best fitness of the population over the generations for different population sizes (15, 20, 25, 30) on the CIFAR10 (left) and CIFAR100 (right) datasets.

TABLE III
THE ACCURACY OF THE BEST SI-EVONET FOUND BY SI-EVONAS AND ITS RUNTIME IN GPU DAYS ON THE CIFAR10 AND CIFAR100 BENCHMARK DATASETS WHEN THE POPULATION SIZE IS SET TO 15, 20, 25, AND 30.

Search dataset	Population size	Fitness value of population	Search cost (GPU days)
CIFAR10	15	91.3	0.350
	20	92.6	0.417
	25	93.5	0.458
	30	93.5	0.542
CIFAR100	15	73.4	0.594
	20	74.6	0.721
	25	77.1	0.813
	30	76.7	0.904

neural architectures in the evaluation phase, the parameter settings that are taken from the peer competitors [31]. Note that in our experiments, we use the SGD optimizer with an initial learning rate of 0.05 for a maximum number of 1000 epochs with a single period cosine decay learning rate schedule [78]. We employ the scheduled path dropout [29] regularization technique of with a probability of 0.2 and an auxiliary classifier [79] with a weight of 0.4 as done in [31]. Dropout is adopted in training the softmax layer. All weights are initialized according to [64]. Other hyperparameter settings used in SI-EvoNAS, such as weight decay rate, momentum, and the number of filters of the first convolutional blocks before the first reduction block (also called initial channels), are also summarized in Table II. Note that these experimental settings are constrained by the computational resources available to us. All experiments are performed on two Nvidia GeForce RTX 2080Ti card. For convenience, the search cost is compared in terms of "GPU days" calculated by multiplying the number of GPU cards deployed with the execution time in days, as suggested in [18, 19, 50].

D. Population sizing

To investigate the influence of the population size on the performance and computational cost of the proposed algorithm, we set the population size $K = 15, 20, 25, 30$ and run the evolutionary search for 300 generations on CIFAR10 and CIFAR100. The change of the classification accuracy of the best individual in the population over the generations for different population sizes are plotted in Fig. 6. The final classification accuracy and runtime of SI-EvoNAS with different population sizes are listed in Table III. From these results, we can conclude that the best performance (93.5% on CIFAR10 and 77.1% CIFAR100) is achieved when the population size is set to 25. When the population size is increased to 30, no performance improvement is observed, although the runtime will become longer. Hence, the population size is set to 25 in the remaining experiments.

TABLE IV
COMPARISON OF SI-EVONET, SI-EVONET-S AND THE PEER COMPETITORS IN TERMS OF THE CLASSIFICATION ACCURACY (%) AND THE CONSUMED COMPUTATIONAL COST ON THE CIFAR10(C10) AND CIFAR100(C100) DATASETS

Architectures	Search method	GPU Days	params	C10	C100
VGG [7]	manual	–	28.05	93.34	67.95
ResNet-101[5]	manual	–	1.7M	93.57	74.84
Pre-ResNet [6]	manual	–	10.3M	95.36	77.29
Wide-ResNet [66]	manual	–	36.5M	95.83	79.5
DenseNet [4]	manual	–	25.6M	96.54	82.82
SENet [36]	manual	–	11.2M	95.95	–
IGCV3-D [69]	manual	–	2.2M	94.96	77.95
MobileNetV2 [67]	manual	–	2.1M	94.56	77.09
ShuffleNet [68]	manual	–	1.06M	90.87	77.14
NAS V3 [8]	RL	22400	7.1M	95.53	–
ENAS [52]	RL	0.5	4.2M	97.06	–
Block-QNN-S [30]	RL	90	6.1M	96.70	82.95
MetaQNN [51]	RL	90	11.2M	93.08	72.86
DPP-Net [70]	RL	8	0.45M	94.16	–
PPP-Net-A [73]	RL	–	0.45M	94.72	–
Proxyless NAS [56]	RL	1500	5.7M	97.92	–
DARTS [31]	GD	1	3.4M	97.18	82.46
RC-DARTS-C42 [62]	GD	1	3.3M	97.19	–
RC-DARTS-C14 [62]	GD	1	0.43M	95.83	–
SNAS [55]	GD	1.5	2.8M	97.15	–
PNAS [72]	SMBO	150	3.2M	96.37	80.47
MdeNAS [59]	MDL	0.16	3.61M	97.45	–
Hier-EA [74]	EA	300	15.7M	96.37	–
large-scale Evo [16]	EA	2750	5.4M	94.60	–
large-scale Evo [16]	EA	2750	40.4M	–	77.00
Amoebanet-A [42]	EA	3150	3.3M	96.66	81.07
LEMONADE [46]	EA	90	13.1M	97.42	–
NSGA-Net [47]	EA	4/8	3.3M	97.25	79.26
CGP-CNN [17]	EA	27	1.7M	94.02	–
CNN-GA [50]	EA	35	2.9M	96.78	–
CNN-GA [50]	EA	40	4.1M	–	79.47
AE-CNN [19]	EA	27	2.0M	95.3	–
AE-CNN [19]	EA	36	5.4M	–	77.6
AE-CNN+E2EPP [18]	EA	7	4.3M	94.70	–
AE-CNN+E2EPP [18]	EA	10	20.9M	–	77.98
SI-EvoNet	**EA**	**0.458**	**0.51M**	**96.02**	**–**
SI-EvoNet	**EA**	**0.813**	**0.99M**	**–**	**79.16**
SI-EvoNet-S	**EA**	**0.458**	**1.84M**	**97.31**	**–**
SI-EvoNet-S	**EA**	**0.813**	**3.32M**	**–**	**84.30**

VI. EXPERIMENTAL RESULTS

A. Overall results

The experimental results in terms of the classification accuracy and consumed GPU days of all compared algorithms are presented in Table IV. In the table, symbol "–" means that the corresponding result was not published. Note that all the results of the competitors in this table are extracted from the papers the methods were published.

From the results in Table IV, we can see that SI-EvoNet-S is able to achieve better performance than all state-of-the-art

manually designed DNNs. The performance enhancement of SI-EvoNet-S on CIFAR100 is larger than 10% compared to VGG and larger than 5% compared to ResNet (depth=101), IGCV3-D, and ResNet (pre-activation, depth=1001).

Compared with twelve non-evolutionary NAS methods under comparison, SI-EvoNet-S achieves better performance than NAS V3, ENAS, Block-QNN-S, MetaQNN, DARTS, RC-DARTS-C42, SNAS and PNAS, but a little worse than Proxyless NAS (0.61%) and MdeNAS (0.14%) on CIFAR10. On the CIFAR100, SI-EvoNet-S has achieved 84.30% classification accuracy, which outperforms all compared non-evolutionary NAS algorithms.

Compared with the nine EvoNAS methods, SI-EvoNet-S performs better than Hierarchical Evolution, Large-scale evolution, AmoebaNet, CGP-CNN, AE-CNN, AE-CNN+E2EPP, but slightly worse than LEMONADE (0.11%) on CIFAR10. On CIFAR100, SI-EvoNet-S has achieved the highest classification accuracy among all compared EvoNAS algorithms. Furthermore, SI-EvoNAS consumes the least search cost among all peers in this category.

Table IV also presents the performance of the SI-EvoNet. On the CIAFR10 and CIFAR100 datasets, SI-EvoNet achieved comparable results with the other lightweight hand-craft network architectures and the lightweight NAS algorithms including MobileNetV2, ShuffleNet, DPP-Net, PPP-Net-A, and compressed DARTS (RC-DARTS-C14). Moreover, we also provide a visualization and analysis of the optimized architecture obtained on CIFAR10 in Section III of the Supplementary materials.

B. Efficiency of Node Inheritance

To overcome the computational bottleneck of EvoNAS, a commonly used approach is to estimate the performance of the network candidates using different proxy metrics. Such proxy metrics aim to reduce the computational costs of networks to accelerate the search process. However, most existing proxy metrics exhibit a low correlation between the estimated performance and the true performance after full training. To demonstrate the efficiency of the proposed node inheritance method for reducing computatonal cost, we compare the estimated performance and the true performance of the best network found by SI-EvoNAS and four EvoNAS algorithms using proxy metrics, including AmoebaNet [42], Economical evolutionary-based NAS (EcoNAS) [80], NSGA-Net [47], and Hier-evo [74]. AmoebaNet uses a heuristics to generate the proxy model, EcoNAS evaluates the less promising networks using a faster proxy and more promising ones using an expensive proxy, NSGA-Net [47] employs a Bayesian Network to assist the search by learning inherent correlations between the layers of the network architecture, and Hier-evo designs a hierarchical genetic representation scheme for network architecture to reduce the search space.

The final performance and the computational cost for the search are presented in Fig. 7 (a), SI-EvoNAS achieves 10x to 1000x search time reduction in comparison to the considered peer EvoNAS algorithms while maintaining comparable performance. In addition, SI-EvoNAS can explore more candidate

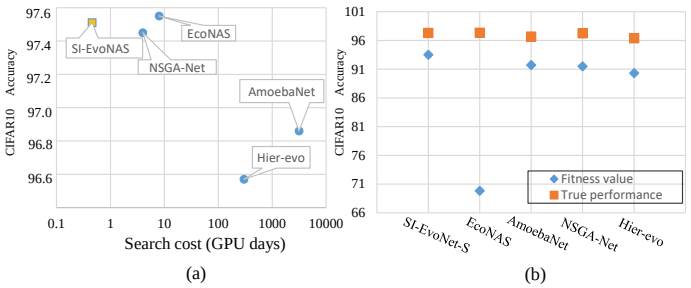

Fig. 7. Comparison of search efficiency between SI-EvoNAS and other four EvoNAS algorithms on CIFAR10 dataset. a) The required computational cost (GPU days). b) The validation and test performance of the best individual.

architectures in fewer search costs. SI-EvoNAS evaluates 15k architectures in 0.458 GPU days. As a comparison, EcoNAS evaluates 1k models in 8 GPU days, and AmoebaNet evaluates 20k models in 3150 GPU days.

Fig. 7 (b) plots the estimated performance of the best network against the final performance after the network is fully trained. The difference between the estimated and true performance of SI-EvoNet-S is only 3.81%, while the performance difference of EcoNAS, AmoebaNet, NSGA-Net, Hier-evo is 27.54%, 4.96%, 5.75%, and 6.07% respectively, indicating that the proposed node inheritance method is able to more precisely estimating the performance during the search process, potentially making the search more efficient.

C. Comparison with Parameter Sharing

Parameter sharing is an effective and popular method for reducing the computational cost in RL based NAS [52]. To compare the performance of parameter sharing and node inheritance, we replace the proposed node inheritance with the parameter sharing method in the evolutionary search and all other settings are kept unchanged. The best performance of the population of SI-EvoNAS using parameter sharing and node inheritance on CIFAR10 and CIFAR100 are plotted in Fig. 8 (a), (c), and the final test performance of the fully trained best networks are plotted in Fig. 8 (b) (d), respectively. From these results, we observe that there is a big difference between the estimated validation performance. Although the final performance of the two found networks is similar on CIFAR10, the final performance of the network found by SI-EvoNAS using node inheritance is better on CIFAR100.

D. Effectiveness of the FR operation

In the deep learning community, the channel attention mechanism is manually incorporated into a neural network. In this work, the channel attention module is part of the search operation in the encoded search space and will be adaptively incorporated into the network if it helps improve the learning performance. To check the effectiveness of the FR operation, we compare the neural network evolved by SI-EvoNAS with other three neural networks. One is a SI-EnoNet evolved by SI-EvoNAS that switches off the FR operation, which is denoted by SI-EvoNet-NA. The other two networks are obtained by manually stacking a different channel attention module, i.e.,

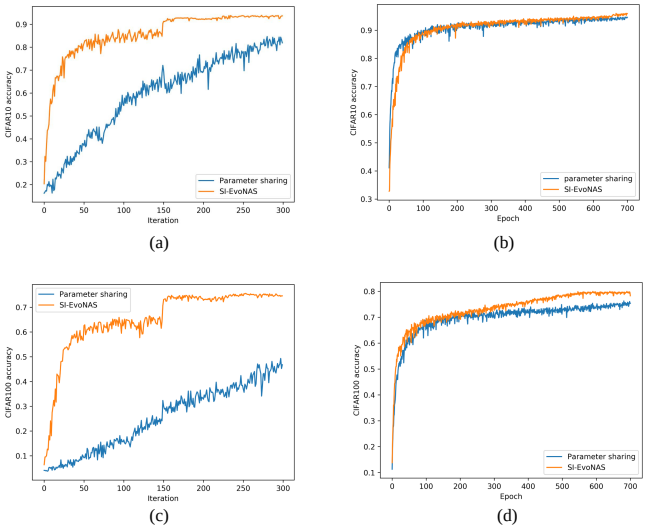

Fig. 8. Comparing the effectiveness of SI-EvoNAS using node inheritance and parameter sharing on CIFAR10 and CIFAR100. In the a) and c), the orange line shows the best fitness in each generation during the evolution iteration. The blue line marks the best validation accuracy of the models that were sampled by RNN-controller in each iteration of the parameter sharing method. In the b) and d), we plot validation accuracy of the two networks found by both methods during the training. Each network is trained for 700 epochs.

SE-block (SE) [36] and the residual channel attention module (RCAM) [35], respectively, into SI-EvoNet-NA, which are denoted by SI-EvoNet+SE and SI-EvoNet+RCAM. For a fair comparison, all other settings are kept the same as described in Subsection V.C.

Table V presents the classification results of the four neural networks under comparison on CIFAR10 and CIFAR100. As we can see from Table V, SI-EvoNet-NA, SI-EvoNet+SE, and SI-EvoNet+RCAM obtain the best classification accuracies of 93.88%, 94.76%, 95.16% on CIFAR10, and 74.82%, 76.88%, 77.42%, on CIFAR100, respectively, while SI-EvoNet obtains the best classification accuracies of 96.02% on CIFAR10 and 79.16% on CIFAR100, respectively. We also note that SI-EvoNet+RCAM achieves better classification performance than SI-EvoNet+SE, indicating that the performance of the residual channel attention module is betters than the channel attention mechanism.

We also incorporate the FR operation into the search space of DARTS [31] and ENAS [52]. The experimental results are shown in Fig. S1 of the Supplementary materials, indicating that both DARTS and ENAS are able to benefit from the FR operation.

TABLE V
CLASSIFICATION ACCURACY (%) OF SI-EVONET, SI-EVONET+SE, SI-EVONET+RCAM AND SI-EVONET-NA ON CIFAR10 AND CIFAR100

Algorithm	CIFAT10	CIFAR100
SI-EvoNet	96.02	79.16
SI-EvoNet+SE	94.76	76.88
SI-EvoNet+RCAM	95.16	77.42
SI-EvoNet-NA	93.88	74.82

E. Neural Architecture Transferring

1) Small-Scale Transfer Experiments: Here, we examine if SI-EvoNet optimized on CIFAR10, a relatively small dataset, can be directly used to effectively learn bigger datasets such as CIFAR100 and SVHN.

To observe the change of the transferring performance of the neural networks during the optimization, we divide the evolutionary process into three stages based on the changes of the learning rate. Specifically, the initial population is denoted as stage 0, generations 1 to 149 is denoted as stage 1, generations 150 to 224 is denoted as stage 2, generations 225 to 299 as stage 3. Then, we evaluate the performance of the best individual at each stage on all data in the SVHN and CIFAR100 datasets, respectively. Note that an individual is randomly picked from the initial population at stage 0.

TABLE VI
THE PERFORMANCE OF THE SI-EVONET OPTIMIZED ON CIFAR10 AND TESTED ON CIFAR100 AND SVHN, RESPECTIVELY, IN VARIOUS SEARCH STAGES. CIFAR100-CIFAR100 DENOTES THE NETWORK OPTIMIZED ON THE TRAINING AND VALIDATION SETS OF CIFAR100 AND EVALUATED ON THE TEST DATASET OF CFAR100, AND SVHN-SVHN THE NETWORK OPTIMIZED ON THE TRAINING AND VALIDATION DATASETS OF SVHN, AND TESTED ON THE VALIDATION DATA OF SVHN.

Test dataset	Search dataset	Individual	Accuracy
SVHN	CIFAR10	Stage 0	97.06
		Stage 1	98.06
		Stage 2	98.11
		Stage 3	98.23
	SVHN	SVHN-SVHN	98.37
CIFAR100	CIFAR10	Stage 0	71.12
		Stage 1	74.44
		Stage 2	76.99
		Stage 3	78.21
	CIFAR100	CIFAR100-CIFAR100	79.16

The experimental results are presented in Table VI. From the table, we can see that the best individual trained on CIFAR10 is able to achieve a classification accuracy of 78.21% and 98.23% on CIFAR100 and SVHN, respectively, which is slightly lower than the network trained on CIFAR100 (79.16%) and SVHN (98.37%). From these results, we can conclude that the neural network structure optimized by the proposed SI-EvoNAS has a promising capability to be transferred to different datasets.

2) Large-Scale Transfer Experiments: We also transfer the SI-EvoNet searched on CIFAR10 to ImageNet. Since each image in ImageNet has a much larger resolution, we first use three 3×3 convolutional layers of stride 2 to reduce the input image resolution from 224×224 to 28×28. And then, the SI-EvoNet-S is stacked after the above-mentioned convolutional layers to construct the whole network. We train the network from scratch until convergence and test it on the test dataset. The test accuracy is considered as the transfer performance. The training settings and the detailed results on ImageNet are provided in Subsection II.A and II.C of the Supplementary materials.

The comparative results on the ImageNet database are presented in Table VII. Overall, SI-EvoNet-S achieves a top 1 and top 5 accuracies of 75.83% and 92.59%, respectively. The results also demonstrate that SI-EvoNet-S has achieved

TABLE VII
COMPARISON OF SI-EVONET-S AND THE PEER COMPETITORS IN TERMS OF THE CLASSIFICATION ACCURACY (%) AND THE CONSUMED COMPUTATIONAL COST ON THE IMAGENET DATASET

Architectures	Top1	Top5	params	search cost	method
Inception-V1 [79]	69.8	89.9	6.6M	–	manual
MobileNet-V2 [67]	70.6	89.5	4.2M	–	manual
ShuffleNet [68]	73.6	89.8	5M	–	manual
NASNet-A [29]	74.0	91.6	5.3M	2000	RL
MnasNet-92 [81]	74.8	92.0	4.4M	–	RL
AmoebaNet-C [42]	75.7	92.4	7.6M	3150	EA
EcoNAS [80]	74.8	–	4.3M	8	EA
PNAS [72]	74.2	91.3	4.7M	225	SMBO
DARTS [31]	73.3	91.3	4.7M	4	GD
ProxylessNAS [56]	75.1	92.5	7.1M	8.3	GD
SNAS [55]	72.7	90.8	4.3M	1.5	GD
SI-EvoNet-S	**75.83**	**92.59**	**4.7M**	**0.458**	**EA**

highly competitive results compared with the popular human-designed architectures, and state-of-the-art NAS methods, including MobileNet [67], MnasNet-92 [81], AmoebaNet-C [42], and ProxylessNAS [56].

VII. CONCLUSION AND FUTURE WORK

This paper proposes a fast EvoNAS framework that is well suited for implementation on devices with limited computation resources. The computational costs of fitness evaluations is dramatically reduced by two related strategies, sampled training of the parent individuals and node inheritance for the weights of the offspring individuals. To further improve the expression ability in evolving large neural networks, the multi-scale feature reconstruction convolutional operation is encoded into search space. Our experimental results demonstrate that SI-EvoNAS can effectively speed up the evolutionary architecture search and achieve promising classification accuracy compared with state-of-the-art NAS algorithms.

Although SI-EvoNAS is competitive to design a high-performance neural network architectures, one main limitation it has is, similar to many other NAS methods based on proxy metrics, that some promising neural architectures may get lost, since their fitness value is evaluated based on the inherited weights without complete training. Thus, one of our future work will be to design a novel EvoNAS framework to protect architecture innovations. In addition, more scalable encoding methods such as generative or developmental encoding are of great importance to achieve a good balance between compactness and flexibility. Finally, network properties apart from classification accuracy, such as complexity, robustness and explanability, will also be considered using multi-objective evolutionary algorithms.

VIII. ACKNOWLEDGMENT

The first author would like to thank S. Huang for his kind support in the experimental studies.

REFERENCES

[1] T. Sainath, A. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for lvcsr," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 8614–8618.
[2] O. Abdelhamid, A. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 22, no. 10, pp. 1533–1545, 2014.
[3] I. Sutskever, O. Vinyals, and Q. Le, "Sequence to sequence learning with neural networks," *Advances in NIPS*, 2014.
[4] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.
[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
[6] ——, "Identity mappings in deep residual networks," pp. 630–645, 2016.
[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[8] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.
[9] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *arXiv preprint arXiv:1808.05377*, 2018.
[10] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," *arXiv preprint arXiv:1901.06032*, 2019.
[11] T. Back, *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.
[12] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.
[13] L. D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: optimizing connections and connectivity," *parallel computing*, vol. 14, no. 3, pp. 347–361, 1990.
[14] L. Xie and A. Yuille, "Genetic CNN," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1379–1388.
[15] X. Wang, Y. Jin, and K. Hao, "Evolving local plasticity rules for synergistic learning in echo state networks," *IEEE Transactions on Neural Networks*, pp. 1–12, 2019.
[16] 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," *arXiv: Neural and Evolutionary Computing*, 2017.
[17] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2017, pp. 497–504.
[18] 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 Transactions on Evolutionary Computation*, 2019.
[19] 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, 2020.
[20] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.
[21] H. Wang, L. Jiao, and X. Yao, "Two_Arch2: An improved two-archive algorithm for many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 4, pp. 524–541, 2014.
[22] Y. Sun, G. G. Yen, and Z. Yi, "Improved regularity model-

based eda for many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 5, pp. 662–678, 2018.
[23] ——, "IGD indicator-based evolutionary algorithm for many-objective optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 2, pp. 173–187, 2019.
[24] K. Swersky, J. Snoek, and R. P. Adams, "Freeze-thaw bayesian optimization," *arXiv preprint arXiv:1406.3896*, 2014.
[25] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148 – 175, 2015.
[26] Y. Jin, H. Wang, T. Chugh, D. Guo, and K. Miettinen, "Data-driven evolutionary optimization: An overview and case studies," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 3, pp. 442–458, 2019.
[27] Y. Jin, "Surrogate-assisted evolutionary computation: Recent advances and future challenges," *Swarm and Evolutionary Computation*, vol. 1, no. 2, pp. 61–70, 2011.
[28] H. Wang, Y. Jin, C. Sun, and J. Doherty, "Offline data-driven evolutionary optimization using selective surrogate ensembles," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 2, pp. 203–216, 2018.
[29] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, 2018, pp. 8697–8710.
[30] 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*, 2020.
[31] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," 2019.
[32] J. Fu, H. Zheng, and T. Mei, "Look closer to see better: Recurrent attention convolutional neural network for fine-grained image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4438–4446.
[33] B. Zhao, X. Wu, J. Feng, Q. Peng, and S. Yan, "Diversified visual attention networks for fine-grained object classification," *IEEE Transactions on Multimedia*, vol. 19, no. 6, pp. 1245–1256, 2017.
[34] R. A. Rensink, "The dynamic representation of scenes," *Visual cognition*, vol. 7, no. 1-3, pp. 17–42, 2000.
[35] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang, "Residual attention network for image classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3156–3164.
[36] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.
[37] Q. Wang, B. Wu, P. Zhu, P. Li, W. Zuo, and Q. Hu, "ECA-net: Efficient channel attention for deep convolutional neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 534–11 542.
[38] M. Jaderberg, K. Simonyan, A. Zisserman *et al.*, "Spatial transformer networks," in *Advances in neural information processing systems*, 2015, pp. 2017–2025.
[39] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, "Draw: A recurrent neural network for image generation," *arXiv preprint arXiv:1502.04623*, 2015.
[40] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.
[41] W. Irwin-Harris, Y. Sun, B. Xue, and M. Zhang, "A graph-based encoding for evolutionary convolutional neural network architecture design," in *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2019, pp. 546–553.
[42] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, 2019, pp. 4780–4789.
[43] C. Wang, C. Xu, X. Yao, and D. Tao, "Evolutionary generative adversarial networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 6, pp. 921–934, 2019.
[44] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "Evolving deep convolutional neural networks for image classification," *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2019.
[45] 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 *ICML 2017 AutoML Workshop*, 2017.
[46] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," in *International Conference on Learning Representations*, 2018.
[47] Z. Lu, I. Whalen, V. N. Boddeti, Y. Dhebar, K. Deb, E. D. Goodman, and W. Banzhaf, "Nsga-net: neural architecture search using multi-objective genetic algorithm," pp. 419–427, 2019.
[48] H. Zhu and Y. Jin, "Multi-objective evolutionary federated learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 4, pp. 1310–1322, 2020.
[49] Y. Jin and B. Sendhoff, "Pareto-based multi-objective machine learning: An overview and case studies," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 38, no. 3, pp. 397–415, 2008.
[50] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing cnn architectures using the genetic algorithm for image classification," *IEEE Transactions on Cybernetics*, 2020.
[51] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," *arXiv preprint arXiv:1611.02167*, 2016.
[52] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *arXiv preprint arXiv:1802.03268*, 2018.
[53] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Uncertainty in Artificial Intelligence*. PMLR, 2020, pp. 367–377.
[54] X. Dong and Y. Yang, "Searching for a robust neural architecture in four gpu hours," in *Proceedings of the IEEE Conference on computer vision and pattern recognition*, 2019, pp. 1761–1770.
[55] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: stochastic neural architecture search," in *International Conference on Learning Representations*, 2019.
[56] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *International Conference on Learning Representations*, 2018.
[57] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Thirty-Second AAAI conference on artificial intelligence*, 2018.
[58] T. Chen, I. Goodfellow, and J. Shlens, "Net2net: Accelerating learning via knowledge transfer," *arXiv preprint arXiv:1511.05641*, 2015.
[59] X. Zheng, R. Ji, L. Tang, B. Zhang, J. Liu, and Q. Tian, "Multinomial distribution learning for effective neural architecture search," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1304–1313.
[60] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-first AAAI conference on artificial intelligence*, 2017.
[61] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.
[62] X. Jin, J. Wang, J. Slocum, M.-H. Yang, S. Dai, S. Yan, and J. Feng, "RC-DARTS: Resource constrained differentiable architecture search," *arXiv preprint arXiv:1912.12814*, 2019.
[63] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated

convolutions," *arXiv preprint arXiv:1511.07122*, 2015.
[64] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.
[65] A. Zela, A. Klein, S. Falkner, and F. Hutter, "Towards automated deep learning: Efficient joint neural architecture and hyperparameter search," *arXiv preprint arXiv:1807.06906*, 2018.
[66] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.
[67] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *arXiv preprint arXiv:1801.04381*, 2018.
[68] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.
[69] K. Sun, M. Li, D. Liu, and J. Wang, "Igcv3: Interleaved low-rank group convolutions for efficient deep neural networks," *arXiv preprint arXiv:1806.00178*, 2018.
[70] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun, "Dpp-net: Device-aware progressive search for pareto-optimal neural architectures," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 517–531.
[71] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2423–2432.
[72] 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 *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 19–34.
[73] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, and M. Sun, "Ppp-net: Platform-aware progressive search for pareto-optimal neural architectures," 2018.
[74] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," *arXiv preprint arXiv:1711.00436*, 2017.
[75] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.
[76] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.
[77] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.
[78] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.
[79] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.
[80] D. Zhou, X. Zhou, W. Zhang, C. C. Loy, S. Yi, X. Zhang, and W. Ouyang, "Econas: Finding proxies for economical neural architecture search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 396–11 404.
[81] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

Haoyu Zhang received the B.Sc. degree from the Taiyuan University of Technology, Taiyuan, China, in 2014, and the M.Sc. degree from Inner Mongolia University of Technology, Hohhot, China, in 2018. He is currently pursuing the Ph.D. degree, with a focus on evolutionary neural architecture search, with the College of Information Science and Technology, Donghua University, Shanghai, China.

Yaochu Jin (M'98-SM'02-F'16) 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 a Distinguished Chair Professor in Computational Intelligence, Department of Computer Science, University of Surrey, Guildford, U.K., where he heads the Nature Inspired Computing and Engineering Group. 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 presently the Editor-in-Chief of the IEEE TRANSACTIONS ON COGNITIVE AND DEVELOPMENTAL SYSTEMS and the Editor-in-Chief of Complex & Intelligent Systems. He is the recipient of the 2018 and 2021 IEEE Transactions on Evolutionary Computation Outstanding Paper Award, the 2015, 2017, and 2020 IEEE Computational Intelligence Magazine Outstanding Paper Award, and the Best Paper Award of the CIBCB 2010. He is recognized as a Highly Cited Researcher 2019 and 2020 by the Web of Science Group. He is a Fellow of IEEE.

Ran Cheng (M'2016) received the B.Sc. degree from the Northeastern University, Shenyang, China, in 2010, and the Ph.D. degree from the University of Surrey, Guildford, U.K., in 2016.

He is currently an Associate Professor with the Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China.

Dr. Cheng was a recipient of the 2018 IEEE Transactions on Evolutionary Computation Outstanding Paper Award, the 2019 IEEE Computational Intelligence Society (CIS) Outstanding Ph.D. Dissertation Award, and the 2020 IEEE Computational Intelligence Magazine Outstanding Paper Award. He is the founding Chair of IEEE Symposium on Model Based Evolutionary Algorithms (IEEE MBEA). He is an Associate Editor of *IEEE Transactions on Artificial Intelligence*.

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

She is currently a Professor at 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.