SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

September 2021, Vol. 64 192106:1–192106:13
https://doi.org/10.1007/s11432-020-3112-8

EAT-NAS: elastic architecture transfer for accelerating large-scale neural architecture search

Jiemin FANG[1,2†], Yukang CHEN[4†], Xinbang ZHANG[4], Qian ZHANG[3], Chang HUANG[3], Gaofeng MENG[4], Wenyu LIU[2] & Xinggang WANG[2*]

[1]*Institute of Artificial Intelligence, Huazhong University of Science and Technology, Wuhan* 430074*, China;*
[2]*School of Electronic Information and Communications, Huazhong University of Science and Technology, Wuhan* 430074*, China;*
[3]*Horizon Robotics, Beijing* 100089*, China;*
[4]*National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing* 100190*, China*

Received 28 February 2020/Revised 19 May 2020/Accepted 4 August 2020/Published online 6 August 2021

Abstract Neural architecture search (NAS) methods have been proposed to relieve human experts from tedious architecture engineering. However, most current methods are constrained in small-scale search owing to the issue of huge computational resource consumption. Meanwhile, the direct application of architectures searched on small datasets to large datasets often bears no performance guarantee due to the discrepancy between different datasets. This limitation impedes the wide use of NAS on large-scale tasks. To overcome this obstacle, we propose an elastic architecture transfer mechanism for accelerating large-scale NAS (EAT-NAS). In our implementations, the architectures are first searched on a small dataset, e.g., CIFAR-10. The best one is chosen as the basic architecture. The search process on a large dataset, e.g., ImageNet, is initialized with the basic architecture as the seed. The large-scale search process is accelerated with the help of the basic architecture. We propose not only a NAS method but also a mechanism for architecture-level transfer learning. In our experiments, we obtain two final models EATNet-A and EATNet-B, which achieve competitive accuracies of 75.5% and 75.6%, respectively, on ImageNet. Both the models also surpass the models searched from scratch on ImageNet under the same settings. For the computational cost, EAT-NAS takes only fewer than 5 days using 8 TITAN X GPUs, which is significantly less than the computational consumption of the state-of-the-art large-scale NAS methods.

Keywords architecture transfer, neural architecture search, evolutionary algorithm, large-scale dataset

Citation Fang J M, Chen Y K, Zhang X B, et al. EAT-NAS: elastic architecture transfer for accelerating large-scale neural architecture search. Sci China Inf Sci, 2021, 64(9): 192106, https://doi.org/10.1007/s11432-020-3112-8

1 Introduction

Deep neural networks have achieved significant success in a wide range of computer vision applications, including image classification [1–3], semantic segmentation [4–7], object detection [8–10], and super-resolution [11]. However, the designing of neural network architectures by human experts often requires tedious trial and error. To make this designing process more efficient, many neural architecture search (NAS) methods [12–14] have been proposed. However, despite their remarkable results, most NAS methods require expensive computational resources. For example, 800 GPUs are used by NAS [15] over 28 days for the task of CIFAR-10 [16] image classification. The real-world applications of NAS involve many large-scale datasets. However, directly performing an architecture search on large-scale datasets, e.g., ImageNet [17], requires significantly high computation cost, which limits the wide application of NAS. Although some search accelerating methods [12–14] have been proposed, few of them directly explore large-scale tasks.

From many previous studies, e.g., VGGNet [18], GoogleNet [19], ResNet [2], it can be said that a neural architecture with good performance on one dataset usually performs well on other datasets

* Corresponding author (email: xgwang@hust.edu.cn)
† Fang J M and Chen Y K have contributed equally, and the work was performed during the internship at Horizon Robotics.

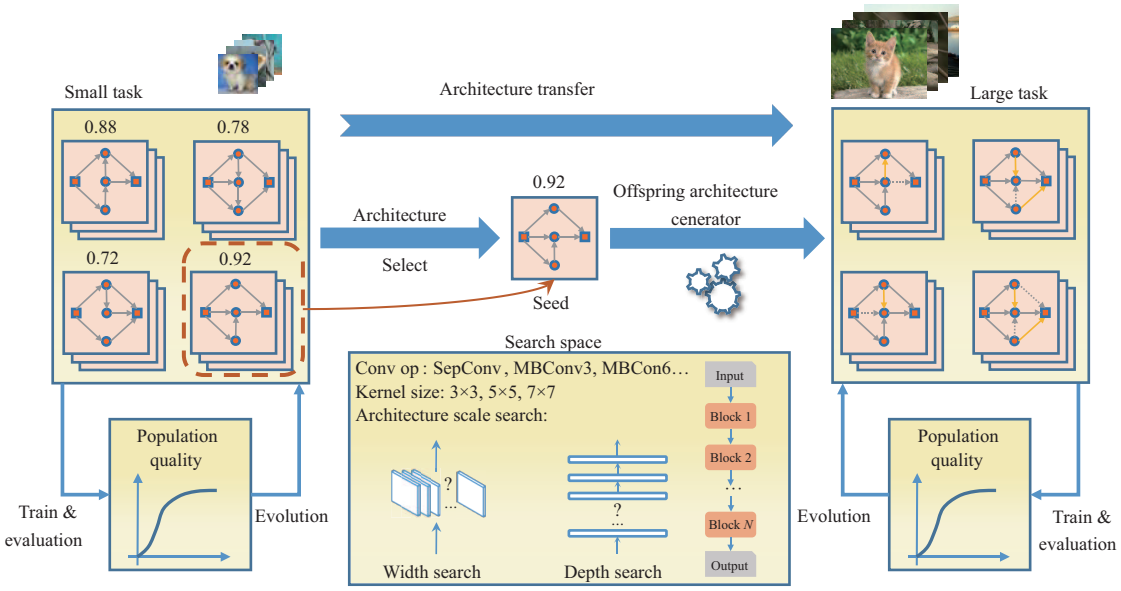

Figure 1 (Color online) Framework of the elastic architecture transfer for NAS (EAT-NAS). We first search for the basic architecture on a small-scale task and then search on a large-scale task with the basic architecture as the seed of the new population initialization.

or tasks. Most ImageNet [17] pre-trained models can be used as the backbones of object detection networks. PNAS [20] demonstrates the transfer capability of the searched architectures by measuring the correlation between the performances on CIFAR-10 and ImageNet for different neural architectures. Most existing NAS methods [12, 13, 20] search for architectures on a small dataset, e.g., CIFAR-10 [16], and they then directly apply these architectures on a large dataset, e.g., ImageNet, with the architectures adjusted manually. Generally, when transferring a neural architecture to a large dataset, the number of stacked cells/layers and filters in the network will increase. Although the searched cell structures or layer operations remain unchanged. However, because of the dataset bias [21] between small and large datasets, the best representation of a neural architecture differs between small and large datasets. Moreover, because of the lower resolution and limited number of training images and data categories in small datasets, the effectiveness of an architecture or the cells/layers of it searched on a small dataset degrades, when the architecture is directly applied on a large dataset or transferred with only the depth and width of the architecture adjusted by handcraft.

In this study, we propose a transfer learning solution to the above-mentioned problem. What we propose is not only a NAS method but also a common mechanism for architecture-level transfer learning. We define the elements in the neural architecture design as architecture primitives, namely the number of filters, the number of layers, operation types, the connection mode, and kernel size. Our proposed elastic architecture transfer (EAT) method automatically transfers a neural architecture to a large-scale dataset, as shown in Figure 1. It is elastic because all the architecture primitives are automatically adjusted or fine-tuned when transferring the architecture to another dataset. The similarities between datasets/tasks facilitate the model/architecture transfer, offering benefits including high-level information/feature extraction and spatial information caption demands between the small and large tasks. Whereas the conventional transfer learning methods mostly focus on the parameter level, our EAT-NAS performs transfer learning on the architecture level.

In our implementation, we choose the evolutionary algorithm (EA) to design the basic NAS method. Benefitting from the population-based search process, the information of the architecture searched on the small dataset can be easily transferred to another architecture population on the large dataset. We adopt a two-stage search process to achieve the architecture transfer between different datasets. First, we perform the first search stage on a small dataset. Upon the completion of the first search process, we choose the best architecture of the population as the seed which is called basic architecture. Second, we use the seed to initialize the architecture population of the second search stage on the large dataset. We obtain the new architectures of the new population by adding some perturbations to the basic architecture. Finally, we proceed with the search process on the large dataset for fewer epochs. The population resulting from

the deformed representations of the basic architecture can evolve faster towards the direction that fits the new dataset.

EAT narrows the gap between datasets on the architecture level automatically. The search process on the large dataset is accelerated by taking advantage of the information from the architecture searched on the small dataset. Our proposed architecture-level transfer mechanism provides a new aspect for transfer learning. The transfer mechanism could not only be deployed to the EA-based NAS method, but also to the gradient or reinforcement learning (RL) based ones. Moreover, EAT is readily available for various tasks and datasets in practical problems.

Our contribution in this study can be summarized as follows:

(1) We propose an elastic architecture transfer mechanism which automates the architecture transfer between datasets and enables NAS on large datasets with low computation cost.

(2) Through experiments on the large-scale dataset, i.e., ImageNet, we prove the efficiency of our method by reducing the search cost to only fewer than 5 days on 8 TITAN X GPUs, about 106x lower than the cost for MnasNet estimated based on [22].

(3) Our searched architectures achieve remarkable ImageNet performance that is comparable to MnasNet which searches directly on the full dataset incurring huge computational cost (75.6% vs. 74.0%).

2 Related work and background

2.1 Neural architecture search

Generating neural architectures automatically has aroused significant interest in recent years. In NAS [15], an RNN network trained via reinforcement learning is utilized as a controller to determine the operation type, parameter, and connection for every layer in the architecture. Although NAS [15] achieves impressive results, its search process is incredibly computation hungry and requires hundreds of GPUs to generate a high-performance architecture on CIFAR-10 datasets. On the basis of the NAS method in [15], many novel methods have been proposed to improve the efficiency of architecture search; these novel methods include finding out the blocks of the architecture instead of the whole network [12, 23], progressive search with a performance predictor [20], an early stopping strategy [23], and parameter sharing [14]. Although these methods have achieved impressive results, their search processes are still computation hungry and become extremely tedious when the searched datasets are large-scale, e.g., ImageNet.

Another stream of studies on NAS utilizes the evolutionary algorithm to generate coded architectures [13, 24, 25]. Modifications to the architecture (filter sizes, layer numbers, and connection patterns) serve as the mutation in the search process. Though they have achieved state-of-the-art results, the computation cost required is also far beyond affordable.

Gradient-based NAS methods [26–32] have also become popular. They discard the black-box searching method and introduce architecture parameters, which are updated on the validation set by using gradient descent, for every path of the network. A softmax classifier is utilized to select the path and operation for each node, while some studies [33, 34] use Gumbel-Softmax for better optimization of the architecture parameters. The search space is relaxed to be continuous so that the architecture can be optimized with respect to its validation set performance by gradient descent. Although gradient-based NAS methods perform remarkably with high efficiency, the search space relies on the super network. All the possible sub-architectures should be included in the delicately designed super network. Therefore, the scalability of the search space is suppressed, e.g., it is not easy to search for the widths of the architecture by gradient-based methods. However, in our evolutionary algorithm based EAT method over the discrete search space, the architecture can be encoded and is easy to be extended to diverse search spaces.

Recently, MnasNet [22] proposes to search directly on large-scale datasets with accuracy and latency co-optimization of the architecture based on RL. MnasNet successfully generates high-performance architectures with promising inference speed, but it requires huge computational resources. In total, 8K models are sampled to be trained on nearly the entire training set for 5 epochs and then evaluated on a 50K validation set. It takes about 91K GPU hours, as estimated according to the description in [22].

There is a work [35] that combines transfer learning with the RL-based NAS method. They transfer the controller by reloading the parameters of the pretrained controller and add a new randomly initialized embedding for the new task. Our proposed elastic architecture transfer method focuses on transferring

on the architecture-level automatically. The architecture searched on the small dataset can be transferred to the large dataset fast and precisely. Compared with searching from scratch, EAT-NAS obtains models with competitive performances and significantly fewer computational resources.

2.2 Evolutionary algorithm based NAS

EA is widely utilized in NAS [13, 24, 25]. As summarized in Algorithm 1, the search process is based on the population of various models. Conventionally, the population $\mathbb{P}$ is first initialized with randomly generated P models which are within the setting range of the search space. Each model is trained and evaluated on the dataset to get its accuracy.

Algorithm 1 Evolutionary algorithm

Input: Population size P, sample size S, dataset $\mathbb{D}$.
Output: The best model M_{best}.
$\mathbb{P}^{(0)} \leftarrow$ `initialize`(P);
for $1 \leqslant j \leqslant P$ **do**
  M_j.acc $\leftarrow$ `train-eval`$(M_j, \mathbb{D})$;
  M_j.score $\leftarrow$ `comp-score`$(M_j, M_j.\text{acc})$;
end for
$Q^{(0)} \leftarrow$ `comp-quality`$(\mathbb{P}^{(0)})$;
while $Q^{(i)}$ not converge **do**
  $S^{(i)} \leftarrow$ `sample`$(\mathbb{P}^{(i)}, S)$;
  $M_{\text{best}}, M_{\text{worst}} \leftarrow$ `pick`$(S^{(i)})$;
  $M_{\text{mut}} \leftarrow$ `mutate`(M_{best});
  M_{mut}.acc $\leftarrow$ `train-eval`$(M_{\text{mut}}, \mathbb{D})$;
  M_{mut}.score $\leftarrow$ `comp-score`$(M_{\text{mut}}, M_{\text{mut}}.\text{acc})$;
  $\mathbb{P}^{(i+1)} \leftarrow$ `remove` M_{worst} `from` $\mathbb{P}^{(i)}$;
  $\mathbb{P}^{(i+1)} \leftarrow$ `add` M_{mut} `to` $\mathbb{P}^{(i)}$;
  $Q^{(i+1)} \leftarrow$ `comp-quality`$(\mathbb{P}^{(i+1)})$;
  $i++$;
end while
$M_{\text{best}} \leftarrow$ `rerank-topk`$(\mathbb{P}_{\text{best}}, k)$.

Following the Pareto-optimal problem [36], we use $\text{acc} \times [\text{size}/T]^{\omega}$ to compute the score of the model, where acc denotes the accuracy of the model, size denotes the model size (i.e., the number of parameters or multiply-add operations), T the target model size, and ω a hyperparameter for controlling the trade-off between accuracy and model size. At each evolution cycle, S models are randomly sampled from the population. The models with the best and worst scores, respectively, are selected. A mutated model is then obtained by adding some transformation to the best scoring model. The mutated model is trained, evaluated and added to the population with its score. Meanwhile, the worst model is removed. The aforementioned search process is called tournament selection [37]. Finally, the top-k performing models are retrained and the best one is selected. Our architecture search method is based on the evolutionary algorithm.

3 Method

To apply an architecture to large-scale tasks, most architecture search methods [12–14, 38] merely rely on the prior knowledge of human experts. They manually transfer an architecture with only expanding the depth and width by multiplication or direct addition. Different from these conventional transfer methods, we propose an EAT method. EAT automatically transfers the neural architecture to a large-scale task by fine-tuning the architecture primitives searched on a small-scale task. It is elastic for the transfer capability of all the architecture primitives, e.g., operator types, structure, and the depth and width of the architecture. EAT accelerates the large-scale search process by making use of the knowledge from the basic architecture searched on the small-scale task. EAT adjusts the basic architecture to the large-scale task with all the architecture primitives fine-tuned.

3.1 Framework

Figure 1 illustrates the process of EAT. The two search processes on the small and the large datasets, respectively, are based on the same search space (see Subsection 3.2). We first search for a set of top-performing architectures on the small dataset, such as CIFAR-10. To obtain better performing

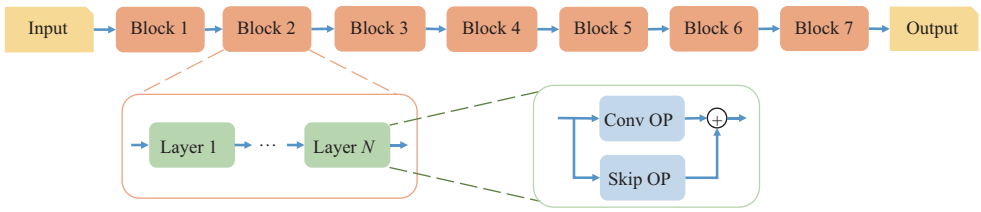

Figure 2 (Color online) Search space. During the search, all the blocks are concatenated to constitute the whole network architecture. Each block comprises several layers and is represented by the following five primitives: convolutional operation type, kernel size, skip connection, width and depth.

architectures, we search for the architecture scale (see Subsection 3.3) with the help of the width and depth factor. We design a criterion called population quality (see Subsection 3.4) to better evaluate the model population. Subsequently, we retrain the top-performing models and choose the best one as the basic architecture. Second, we start the architecture search on the large-scale task with the basic architecture as the seed to initialize the new architecture population. We design an architecture perturbation function (see Subsection 3.5) to produce the architectures of the new population. We then continue the architecture search on the large-scale task based on the population derived from the basic architecture. In this way, the search on the new task is accelerated, and better-performing models are obtained than those obtained via searching from scratch, which is benefited from the useful information of the basic architecture. Subsection 4.4 displays the results of contrast. Finally, we select the best one from the top-k performing models in the population by retraining them on the full large-scale dataset. Algorithm 2 displays the entire procedure of the elastic architecture transfer.

Algorithm 2 Elastic architecture transfer

Input: Datasets $\mathbb{D}_1$, $\mathbb{D}_2$, population size P.
Output: The target architecture $\text{Arch}_{\text{target}}$.

// Initialize the population on $\mathbb{D}_1$.
$\mathbb{P}_1 \leftarrow$ initialize(P);
evolve($\mathbb{P}_1$, $\mathbb{D}_1$);
$\text{Arch}_{\text{basic}} \leftarrow$ rerank-topk & select($\mathbb{P}_1$, k);
// Initialize the population on $\mathbb{D}_2$.
for $1 \leqslant i \leqslant P$ **do**
    $\text{Arch}_i \leftarrow$ arch-perturbation($\text{Arch}_{\text{basic}}$);
    $\mathbb{P}_2$.append(Arch_i);
end for
evolve($\mathbb{P}_2$, $\mathbb{D}_2$);
$\text{Arch}_{\text{target}} \leftarrow$ rerank-topk & select($\mathbb{P}_2$, k).

3.2 Search space

A well-designed search space is essential for NAS. Inspired by MnasNet [22], we employ an architecture search space with MobileNetV2 [3] as the backbone. As shown in Figure 2, the network is divided into several blocks which can be different from each other. Each block consists of several layers, whose operations are determined by a per-block sub search space. Specifically, the sub search space for each block could be parsed as follows:

• **Conv operation.** Depthwise separable convolution (SepConv) [39], mobile inverted bottleneck convolution (MBConv) with diverse expansion ratios {3, 6} [3].

• **Kernel size.** 3×3, 5×5, 7×7.

• **Skip connection.** Whether to add a skip connection for every layer.

• **Width factor.** The expansion ratio of the output width to input width, $\text{factor}_{\text{width}} = N_o/N_i$, [0.5, 1.0, 1.5, 2.0].

• **Depth factor.** The number of layers per block, [1, 2, 3, 4].

Besides, down-sampling and width-expansion operations are applied in the first layer of each block.

To manipulate the neural architecture more conveniently, every architecture is encoded following the format defined in the search space. As a network could be separated into several blocks, the whole architecture is presented as a block set $\text{Arch} = \{B^1, B^2, \ldots, B^n\}$. Each block consists of the above-mentioned five primitives, which is encoded by a tuple $B^i = (\text{conv}, \text{kernel}, \text{skip}, \text{width}, \text{depth})$. Every manipulation for the neural architecture is performed based on the model code.

3.3 Architecture scale search

Most NAS methods [12–14, 20] treat the scale of the architecture as a fixed element based on the prior knowledge from human experts. The scale, depth, and width of the architecture usually affect the architecture performance. To obtain better performing architectures, we search for the architecture scale by manipulating the width and depth factors.

To accelerate the architecture search process, we employ the parameter sharing method on each model during the search. Inspired by the function-preserving transformations, namely Net2WiderNet and Net2DeeperNet, in Net2Net [40], we propose a modified parameter sharing method used for model training. When initializing the parameters for a network, the proposed algorithm traverses the operation type of each layer. If the operation type and kernel size of the layer are consistent with those of the shared model, then parameter sharing is applied on this layer; otherwise, the parameters are randomly initialized. We introduce two parameter sharing behaviors on the network width and depth, respectively.

Parameter sharing on the width-level. By sharing the parameters, we desire to inherit as much information as possible from the former model. For the convolutional layer, we assume that the convolutional kernel of the l^{th} layer $\boldsymbol{K}_l$ has the shape of $(w^l, h^l, \text{ch}_{\text{in}}^l, \text{ch}_{\text{out}}^l)$, where w^l and h^l denote the filter width and height respectively, while ch_{in}^l and ch_{out}^l denote the number of input and output channels respectively. If the original convolutional kernel $\boldsymbol{K}_o$ has the shape of $(w^o, h^o, \text{ch}_{\text{in}}^o, \text{ch}_{\text{out}}^o)$, we perform the sharing strategy as described in Algorithm 3. In addition to the shared parameters, the rest part of $\boldsymbol{K}_l$ is randomly initialized.

Algorithm 3 Parameter sharing on the width-level

Input: Kernel $\boldsymbol{K}_l$ in layer l, the original kernel $\boldsymbol{K}_o$.
Output: Kernel $\boldsymbol{K}_l$ in layer l.
1: $\text{ch}_{\text{in}}^s \leftarrow \texttt{min}(\text{ch}_{\text{in}}^l, \text{ch}_{\text{in}}^o)$;
2: $\text{ch}_{\text{out}}^s \leftarrow \texttt{min}(\text{ch}_{\text{out}}^l, \text{ch}_{\text{out}}^o)$;
3: $\boldsymbol{K}_l \leftarrow \boldsymbol{K}_o(w^o, h^o, \text{ch}_{\text{in}}^s, \text{ch}_{\text{out}}^s)$.

Parameter sharing on the depth-level. The parameters are shared on the depth level in a way similar to that on the width level. Suppose that $\boldsymbol{U}[1, 2, \ldots, l_u]$ denotes the parameter matrix of one block which has l_u layers. Additionally, assume that $\boldsymbol{W}[1, 2, \ldots, l_w]$ denotes the parameter matrix of the corresponding block, which has l_w layers, from the shared model. The parameter sharing process is illustrated in two cases as follows:

(i) $l_u > l_w$:

$$\boldsymbol{U}[i] = \begin{cases} \boldsymbol{W}[i], & \text{if } i < l_w, \\ \Gamma(i), & \text{otherwise;} \end{cases} \tag{1}$$

(ii) $l_u \leqslant l_w$:

$$\boldsymbol{U}[1, 2, \ldots, l_u] = \boldsymbol{W}[1, 2, \ldots, l_u], \tag{2}$$

where Γ denotes a random weight initializer based on the normal distribution.

3.4 Population quality

During the evolution process, we design a criterion called population quality to evaluate the model population. With the search proceeding, the scores of the models in the population improve. To ascertain whether the population evolution converges, the variance of the model scores needs to be taken into consideration. Merely depending on the mean score of the models in the population may result in imprecision, because accuracy gains could derive both from parameter sharing and model performance promotion.

Therefore, until the population converges to an optimal solution, the mean score of the models should be as high as possible while the variance of the model scores as low as possible. This issue could be treated as a Pareto-optimal problem [36]. To approximate the Pareto optimal solution, we utilize a target function, population quality, as follows:

$$Q = \text{score}_{\text{mean}} \times \left[\frac{\text{std}}{\text{target}_{\text{std}}}\right]^{\omega}, \tag{3}$$

where ω denotes the weight factor defined as follows:

$$\omega = \begin{cases} \alpha, & \text{if std} < \text{target}_{\text{std}}; \\ \beta, & \text{otherwise}, \end{cases} \tag{4}$$

where α and β denote the hyperparameters for controlling the trade-off between the mean score of the models and the standard deviation of the model scores.

In Eq. (3), $\text{score}_{\text{mean}}$ denotes the mean score of the models in the population, std the standard deviation of model scores, and $\text{target}_{\text{std}}$ the target std of model scores. The term $\text{target}_{\text{std}}$ is a preset parameter, 0.1, in our experiments, for minimizing the std of model scores during search and obtain a population of models with similar performance. Following MnasNet [22], we set $\alpha = \beta = -0.07$ to assign the value to ω. After the evolution, we select the best-quality population and retrain top-k models of the population.

3.5 Architecture perturbation function

To transfer the architecture, we initialize the new population on the large scale dataset with the the basic architecture searched on the small dataset as the seed. We design an architecture perturbation function to derive new architectures by adding some perturbation to the input architecture code homogeneously and slightly. Algorithm 4 illustrates the process of the perturbation function. In each block of the architecture, there are a total of five architecture primitives (conv, kernel, skip, width, depth) to be manipulated as described in Subsection 3.2. We randomly select one type of the five primitives to perturb. We then stochastically generate a new value of the selected primitive within the restriction of our search space and replace the existing one.

Algorithm 4 Architecture perturbation function

Input: Basic architecture Arch_b, search space $\mathbb{S}$, number of blocks N_{blocks}, and primitives prims.
Output: Perturbed architecture Arch_p.

$\text{Arch}_p \leftarrow$ `copy`(Arch_b);
for $1 \leqslant j \leqslant N_{\text{blocks}}$ **do**
  prim $\leftarrow$ `rand-select`(prims);
  value $\leftarrow$ `rand-generate`(prim, $\mathbb{S}$);
  $B_t^j \leftarrow$ `get-block`(Arch_t, j);
  B_t^j[type] $\leftarrow$ value;
end for

When initializing the population on the large dataset, we produce every new architecture by applying the architecture perturbation function to the basic architecture until the number of architectures meets the population size. In another word, each initial architecture of the new population is a deformed representation of the basic one. After initializing the new population, the evolution begins as the same procedure described in Algorithm 1.

Randomized perturbations are introduced into the initial population for more possible architectures to be searched. The proposed perturbations are slight, and thus the similarity among the architectures is maintained. The original similarity between the datasets is unchanged, as the perturbations are only performed on the architecture level. Moreover, the architecture perturbation function is utilized as the mutation operation in the evolution.

4 Experiments

Our experiments mainly consist of two stages, searching for the basic architecture on CIFAR-10 and then transferring it to ImageNet. In this section, we introduce some implementation details in EAT-NAS and report the experimental results. We analyze the results of some ablation experiments to demonstrate the effectiveness of the proposed EAT-NAS method.

4.1 Search on CIFAR-10

The experiments on CIFAR-10 are divided into two steps including architecture search and evaluation. CIFAR-10 consists of 50000 training images and 10000 testing images. We split the original training set (80%–20%) to create our training and validation sets for the search process. The original CIFAR-10

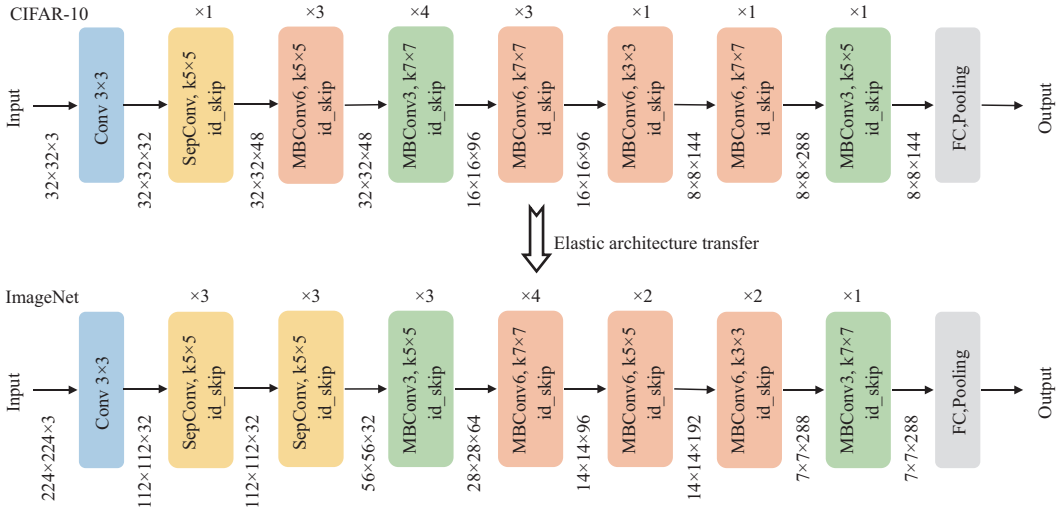

Figure 3 (Color online) The architectures searched by EAT-NAS. The upper one is the basic architecture searched on CIFAR-10. And the lower one is the architecture, namely EATNet-A, searched on ImageNet which is transferred from the basic architecture.

testing set is only utilized in the evaluation process of the final searched models. All images are whitened with the channel mean subtracted and the channel standard deviation divided. Then we crop 32×32 patches from images padded to 40×40 and randomly flip them horizontally.

During the search process, we set the population size to 64 and the sample size to 16. Every model generated during the evolution is trained for 1 epoch and is evaluated on the separate validation set to measure its accuracy. We mutate about 1400 models during the total evolution and only top-8 models are retrained on the full training dataset and evaluated on the testing dataset. The number of model parameters is the sub-optimizing objective during the evolution with the target as 3.0M. Each model on CIFAR-10 consists of 7 blocks and the downsampling operations are performed in the third and fifth blocks. The initial number of channels is 32. The depth and width of the mutated model would vary in an extremely wide range within our search space if no restriction is set. Some constraints are added to the scale of the model within an acceptable range to avoid the memory running out of control during the search. On CIFAR-10, the total expansion ratio is limited within $[4, 10]$.

For training during the search process, the batch size is set as 128. We use the stochastic gradient descent (SGD) optimizer with the learning rate of 0.0256 (fixed during the search), momentum of 0.9, and weight decay of 3×10^{-4}. The search experiment is performed on 4 GPUs, taking about 22 h. For evaluation, every model is trained for 630 epochs with a batch size of 96. The initial learning rate is 0.0125, and the learning rate follows the cosine annealing restart schedule [41]. Other hyperparameters remain the same as that in the search process. Following existing studies [12–14, 20], additional enhancements include cutout [42] with the length of 16, and auxiliary towers with weight 0.4. The training of the searched model takes around 13 h on two GPUs.

Since the CIFAR-10 results are subject to high variance even with exactly the same training setup [38], we report the mean and standard deviation of 5 independent runs for our model. The basic model achieves 96.42% mean test accuracy (the standard deviation of 0.05) with only 2.04M parameters. The architecture of the basic model is shown in Figure 3.

4.2 Transferring to ImageNet

We use the architecture of the basic model searched on CIFAR-10 as the seed to generate the model population on ImageNet. The search process is carried out on the whole ImageNet training dataset. To avoid overfitting the original ImageNet validation set, we have a separate validation set containing 50K images randomly selected from the training set to measure the accuracy. We use the architecture perturbation function to produce 64 new architectures based on the basic architecture.

During the architecture search, we train every model for one epoch with a batch size of 128 and a learning rate of 0.05. Following GoogleNet [19], the input images are sampled as various sized patches whose size is distributed between 20% and 100% of the image area with aspect ratio constrained to the range of $[\frac{3}{4}, \frac{4}{3}]$. The number of multiply-add operations is set as the sub-optimization objective during

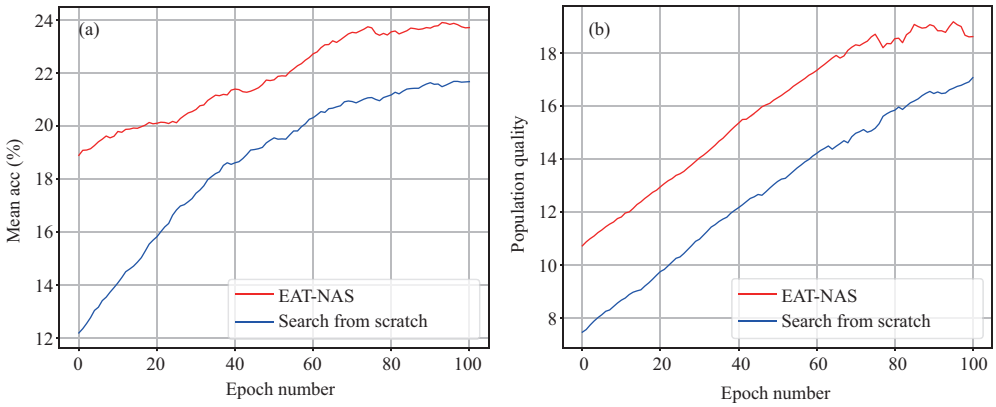

Figure 4 (Color online) Comparing the evolution processes between EAT-NAS and search from scratch on ImageNet. (a) Mean accuracy of models in the population; (b) population quality.

Table 1 ImageNet classification results in the mobile setting. The results of manually designed models are provided in the top section, other NAS results in the middle section, and the result of our models in the bottom section[a)]

Model	#Params (M)	#Mult-Adds (M)	Top-1/Top-5 Acc (%)	Search dataset	Search time (GPU hours)
MobileNet-v1 [43]	4.2	575	70.6/89.5	–	–
MobileNet-v2 [3]	3.4	300	71.7/ –	–	–
MobileNet-v2 (1.4) [3]	6.9	585	74.7/ –	–	–
ShuffleNet-v1 2x [44]	≈ 5	524	73.7/ –	–	–
NASNet-A [12]	5.3	564	74.0/91.6	CIFAR-10	48K
NASNet-B [12]	5.3	488	72.8/91.3	CIFAR-10	48K
NASNet-C [12]	4.9	558	72.5/91.0	CIFAR-10	48K
AmoebaNet-A [13]	5.1	555	74.5/92.0	CIFAR-10	76K
AmoebaNet-B [13]	5.3	555	74.0/91.5	CIFAR-10	76K
AmoebaNet-C [13]	5.1	535	75.1/92.1	CIFAR-10	76K
PNASNet-5 [20]	5.1	588	74.2/91.9	CIFAR-10	6K
MnasNet [22]	4.2	317	74.0/91.8	ImageNet	91K
DARTS† [26]	4.7	574	73.3/91.3	CIFAR-10	96
P-DARTS† [45]	4.9	557	75.6/92.6	CIFAR-10	7.2
PC-DARTS† [46]	5.3	597	75.8/92.7	ImageNet	91
Proxyless (GPU)† [28]	7.1	465	75.1/92.5	ImageNet	200
SNAS† [47]	4.3	522	72.7/90.8	CIFAR-10	36
EATNet-A	5.1	563	75.5/92.5	CIFAR-10 to ImageNet	856
EATNet-B	5.2	545	75.6/92.4	CIFAR-10 to ImageNet	856
EATNet-C	4.6	417	73.9/91.8	CIFAR-10 to ImageNet	856

a) † denotes the gradient-based search method.

the evolution, with the target as 500M. The other hyperparameters of the search process are the same as that on CIFAR-10. Each model is composed of 7 blocks and the number of input channels is 32 as well. The downsampling operations are carried out in the input layer and the 2nd, 3rd, 4th, 6th block. The number of layers is limited within $[16, 18]$ and the total width expansion ratio is limited within $[8, 16]$.

For evaluating the model performance on ImageNet, we retrain the final top-8 models on 224×224 images of the training dataset for 240 epochs, using the standard SGD optimizer with 0.9 momentum rate, 4×10^{-5} weight decay and 0.1-weighted label smoothing. The batch size is 128 on 8 GPUs. The initial learning rate is 0.5 and it decays with a cosine annealing schedule [41] to 1×10^{-4}. We use the standard GoogleNet [19] data augmentation.

As shown in Figure 4, the evolution process takes about 100 evolution epochs to converge. In another word, taking the initial 64 models into account, we only sample around 164 models to find out the best one based on the basic architecture. In MnasNet [22], the controller samples about 8K models during

Table 2 Results on CIFAR-100[a)]

Model	#Params (M)	Top-1 Acc (%)
ResNet [2]	1.7	72.8
LS Evo [48]	40.4	77.0
SS	2.2	77.4
EATNet	1.9	**78.1**

a) The comparison results are from [48]. "LS Evo": large-scale evolution. "SS": the model searched from scratch on CIFAR-100.

architecture search, 50 times the amount of ours. With much less computational resources, EAT-NAS achieves comparable results on ImageNet as in Table 1 [3,12,13,20,22,26,28,43–47]. Though the gradient-based methods [26, 28, 45, 46] are faster in principal than the EA based one, our EAT-NAS still achieves competitive results compared with the gradient-based ones.

Figure 3 shows the basic architecture and the architecture of EATNet-A. From the figure, we observe that compared with the basic architecture, there are some transformations in EATNet-A. During the elastic architecture transfer process, all architecture primitives in the basic architecture are likely to be modified. For example, the operation type in the second block has changed from Mbconv6 to SepConv, and the kernel sizes have also changed in some blocks. The depth and the width of the architecture have changed as well. The modifications to the architecture primitives adapt the architecture to the new dataset.

In summary, our EAT-NAS includes two stages, search on CIFAR-10 and transfer to ImageNet. It takes 22 h on 4 GPUs to search for the basic architecture on CIFAR-10 and 4 days on 8 GPUs to transfer to ImageNet. Though DARTS [26] and SNAS [47] take less search time, they only search on the small dataset, CIFAR-10. And our ImageNet performances clearly surpass them.

4.3 Transferring to CIFAR-100

We further perform an experiment by transferring the architecture searched on CIFAR-10 to CIFAR-100. All the search and retraining settings and hyper-parameters on CIFAR-100 are the same as the experiments on CIFAR-10 in Subsection 4.1. The search process on CIFAR-100 takes 100 epochs in total. The results are shown in Table 2 [2, 48]. Compared with the handcrafted architecture ResNet [2], EATNet achieves a 5.3% higher accuracy with similar Params and a 1.1% higher accuracy with only 4.7% Params of LS Evo [48]. Compared with the model searched from scratch on CIFAR-100, EATNet obtains a 0.7 higher accuracy with 0.3M fewer Params.

4.4 Ablation study

Efficiency of EAT. To demonstrate the efficiency of our proposed method EAT-NAS, we carry out the search process on ImageNet from scratch. All the settings are the same as EAT-NAS both in the search and evaluation processes. The search process takes the same GPU hours as EAT-NAS as well. Figure 4 shows the mean accuracy of population models and the population quality of EAT-NAS compared with the search from scratch on ImageNet. The search epochs are set equal for the fair comparison. We observe that after initialization, the mean accuracy of models is obviously higher of EAT-NAS than that of search from scratch all through the search process. And the evolution process converges faster for EAT-NAS. In Table 3, we compare the best performing model of top-8 searched from scratch with that from EAT-NAS. The compared models we select are guaranteed to have similar model sizes. Obviously, the model EATNet-C surpasses that searched from scratch.

Effectiveness of EAT. To verify the effectiveness of our elastic architecture transfer method, we apply our basic architecture searched on CIFAR-10 directly on ImageNet without any modification as the handcrafted transfer does. We train the basic model on ImageNet under the same settings as those of EAT-NAS. The performance of the basic model is shown in Table 3. The basic model achieves a worse validation accuracy with a much larger number of multiply-add operations. Compared with the basic model, not only does EAT promote the accuracy, but also optimizes the computation cost of the model.

Impact of basic architecture performance. We select one architecture with worse performance as the basic architecture in our transfer process, whose validation accuracy on CIFAR-10 is 96.16% and the number of parameters is 1.9M. As shown in Figure 5, the basic architecture with worse performance has a negative impact on the transfer process. The mean accuracy deteriorates in the preliminary epochs. This experiment demonstrates the importance of a well-performing basic architecture for transfer. We retrain

Table 3 Results of the contrast experiments on ImageNet[a)]

Model	#Params (M)	#Mult-Adds (M)	Top-1/Top-5 Acc (%)
SS	5.55	465	72.5/90.7
Basic model	3.27	934	75.2/92.5
Model-B	3.22	408	72.7/91.0
EATNet-A	5.12	563	75.5/92.5
EATNet-B	5.20	545	75.6/92.4
EATNet-C	4.63	417	73.9/91.8

a) "SS" denotes the model searched from scratch on ImageNet. The basic model searched on CIFAR-10 is directly applied on ImageNet without any modification. Model-B denotes the best model searched on ImageNet with a poor-performing basic architecture. EATNet-C is a small model searched by EAT-NAS.

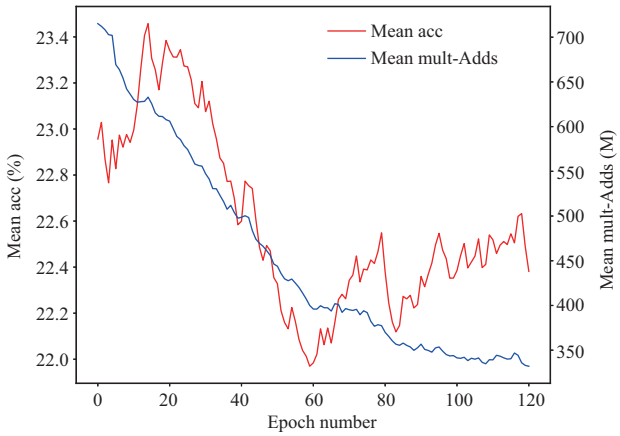

Figure 5 (Color online) Mean accuracy and the mean multiply-Adds of the models during the search on ImageNet whose basic architecture has worse performance on CIFAR-10.

the searched top-8 models under the same settings and compare the best one with that searched by EAT-NAS which has a similar model size. As shown in Table 3, models searched by EAT-NAS surpass those searched with the poor-performing basic architecture. We attribute the results to the poor performance of the basic architecture.

5 Conclusion and future work

In this paper, we propose an elastic architecture transfer mechanism for accelerating the large-scale neural architecture search (EAT-NAS). Rather than spending a lot of computation resources to directly search the neural architectures on large-scale tasks, EAT-NAS makes full use of the information of the basic architecture searched on the small-scale task. We transfer the basic architecture with elasticity to the large-scale task fast and precisely. With less computational resources, we obtain networks with excellent ImageNet classification results in mobile sizes.

In the future, we would try to combine the proposed mechanism with other search methods, including reinforcement learning and gradient-based NAS. Additionally, EAT-NAS can be utilized to search for neural architectures in other computer vision tasks like detection, segmentation, and tracking.

Acknowledgements This work was in part supported by National Natural Science Foundation of China (NSFC) (Grant Nos. 61876212, 61976208, 61733007), Zhejiang Lab (Grant No. 2019NB0AB02), and HUST-Horizon Computer Vision Research Center. We thank Liangchen SONG and Guoli WANG for the discussion and assistance.

References

1 Szegedy C, Vanhoucke V, Ioffe S, et al. Rethinking the inception architecture for computer vision. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2016
2 He K M, Zhang X Y, Ren S Q, et al. Deep residual learning for image recognition. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2016
3 Sandler M, Howard A, Zhu M, et al. Mobilenetv2: inverted residuals and linear bottlenecks. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2018

4 Chen L C, Papandreou G, Kokkinos I, et al. DeepLab: semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. IEEE Trans Pattern Anal Mach Intell, 2018, 40: 834–848
5 Chen L C, Papandreou G, Schroff F, et al. Rethinking atrous convolution for semantic image segmentation. 2017. ArXiv:1706.05587
6 Huang Z L, Wang X G, Huang L C, et al. CCNet: criss-cross attention for semantic segmentation. In: Proceedings of International Conference on Computer Vision, 2019
7 Huang Z L, Wang X G, Wei Y C, et al. CCNet: criss-cross attention for semantic segmentation. IEEE Trans Pattern Anal Mach Intell, 2020. doi: 10.1109/TPAMI.2020.3007032
8 Ren S, He K, Girshick R, et al. Faster R-CNN: towards real-time object detection with region proposal networks. IEEE Trans Pattern Anal Mach Intell, 2017, 39: 1137–1149
9 Liu W, Anguelov D, Erhan D, et al. SSD: single shot multibox detector. In: Proceedings of European Conference on Computer Vision, 2016
10 Lin T Y, Goyal P, Girshick R, et al. Focal loss for dense object detection. In: Proceedings of International Conference on Computer Vision, 2017
11 Yi P, Wang Z Y, Jiang K, et al. Multi-temporal ultra dense memory network for video super-resolution. IEEE Trans Circ Syst Video Technol, 2020, 30: 2503–2516
12 Zoph B, Vasudevan V, Shlens J, et al. Learning transferable architectures for scalable image recognition. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2018
13 Real E, Aggarwal A, Huang Y, et al. Regularized evolution for image classifier architecture search. In: Proceedings of AAAI Conference on Artificial Intelligence, 2019
14 Pham H, Guan M Y, Zoph B, et al. Efficient neural architecture search via parameter sharing. In: Proceedings of International Conference on Machine Learning, 2018
15 Zoph B, Le Q V. Neural architecture search with reinforcement learning. In: Proceedings of International Conference on Learning Representations, 2017
16 Krizhevsky A, Hinton G. Learning Multiple Layers of Features From Tiny Images. Technical Report, 2009
17 Deng J, Dong W, Socher R, et al. Imagenet: a large-scale hierarchical image database. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2009
18 Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. 2014. ArXiv:1409.1556
19 Szegedy C, Liu W, Jia Y Q, et al. Going deeper with convolutions. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2015
20 Liu C X, Zoph B, Neumann M, et al. Progressive neural architecture search. In: Proceedings of European Conference on Computer Vision, 2018
21 Tommasi T, Patricia N, Caputo B, et al. A deeper look at dataset bias. In: Proceedings of Domain Adaptation in Computer Vision Applications, 2017
22 Tan M X, Chen B, Pang R M, et al. Mnasnet: platform-aware neural architecture search for mobile. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2019
23 Zhong Z, Yan J J, Wu W, et al. Practical block-wise neural network architecture generation. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2018
24 Miikkulainen R, Liang J, Meyerson E, et al. Evolving deep neural networks. In: Proceedings of Artificial Intelligence in the Age of Neural Networks and Brain Computing, 2019
25 Lu Z C, Whalen I, Boddeti V, et al. NSGA-Net: a multi-objective genetic algorithm for neural architecture search. 2018. ArXiv:1810.03522
26 Liu H, Simonyan K, Yang Y. DARTS: differentiable architecture search. In: Proceedings of International Conference on Learning Representations, 2019
27 Zhang X B, Huang Z H, Wang N Y. You only search once: single shot neural architecture search via direct sparse optimization. 2018. ArXiv:1811.01567
28 Cai H, Zhu L G, Han S. ProxylessNAS: direct neural architecture search on target task and hardware. In: Proceedings of International Conference on Learning Representations, 2019
29 Fang J M, Sun Y Z, Zhang Q, et al. Densely connected search space for more flexible neural architecture search. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2020
30 Fang J M, Sun Y Z, Peng K, et al. Fast neural network adaptation via parameter remapping and architecture search. In: Proceedings of International Conference on Learning Representations, 2020
31 Dong X Y, Yang Y. Searching for a robust neural architecture in four GPU hours. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2019
32 Mei J R, Li Y W, Lian X C, et al. Atomnas: fine-grained end-to-end neural architecture search. In: Proceedings of International Conference on Learning Representations, 2020
33 Wu B C, Dai X L, Zhang P Z, et al. FBNet: hardware-aware efficient convnet design via differentiable neural architecture search. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2019
34 Chang J L, Zhang X B, Guo Y W, et al. Data: differentiable architecture approximation. In: Proceedings of Conference on Neural Information Processing Systems, 2019
35 Wong C, Houlsby N, Lu Y F, et al. Transfer learning with neural automl. In: Proceedings of Conference on Neural Information

Processing Systems, 2018
36 Deb K. Multi-objective optimization. In: Proceedings of Search Methodologies, 2014
37 Goldberg D E, Deb K. A comparative analysis of selection schemes used in genetic algorithms. Found Genetic Algorithms, 1991, 1: 69–93
38 Liu H X, Simonyan K, Vinyals O, et al. Hierarchical representations for efficient architecture search. In: Proceedings of International Conference on Learning Representations, 2018
39 Chollet F. Xception: deep learning with depthwise separable convolutions. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2017
40 Chen T, Goodfellow I J, Shlens J. Net2Net: accelerating learning via knowledge transfer. In: Proceedings of International Conference on Learning Representations, 2016
41 Loshchilov I, Hutter F. SGDR: stochastic gradient descent with warm restarts. 2016. ArXiv:1608.03983
42 DeVries T, Taylor G W. Improved regularization of convolutional neural networks with cutout. 2017. ArXiv:1708.04552
43 Howard A G, Zhu M L, Chen B, et al. Mobilenets: efficient convolutional neural networks for mobile vision applications. 2017. ArXiv:1704.04861
44 Zhang X Y, Zhou X Y, Lin M X, et al. Shufflenet: an extremely efficient convolutional neural network for mobile devices. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2018
45 Chen X, Xie L X, Wu J, et al. Progressive differentiable architecture search: bridging the depth GAP between search and evaluation. In: Proceedings of International Conference on Computer Vision, 2019
46 Xu Y H, Xie L X, Zhang X P, et al. PC-DARTS: partial channel connections for memory-efficient architecture search. In: Proceedings of International Conference on Learning Representations, 2020
47 Xie S R, Zheng H H, Liu C X, et al. SNAS: stochastic neural architecture search. In: Proceedings of International Conference on Learning Representations, 2019
48 Real E, Moore S, Selle A, et al. Large-scale evolution of image classifiers. In: Proceedings of International Conference on Machine Learning, 2017