EAT-NAS: Elastic Architecture Transfer
for Accelerating Large-scale Neural Architecture Search

Jiemin Fang[1*†], Yukang Chen[3†], Xinbang Zhang[3], Qian Zhang[2]
Chang Huang[2], Gaofeng Meng[3], Wenyu Liu[1], Xinggang Wang[1]
[1]*School of EIC, Huazhong University of Science and Technology* [2]*Horizon Robotics*
[3]*National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences*
{*jaminfong, xgwang, liuwy*}@*hust.edu.cn* {*qian01.zhang, chang.huang*}@*horizon.ai*
{*yukang.chen, xinbang.zhang, gfmeng*}@*nlpr.ia.ac.cn*

Abstract

Neural architecture search (NAS) methods have been proposed to release human experts from tedious architecture engineering. However, most current methods are constrained in small-scale search due to the issue of computational resources. Meanwhile, directly applying architectures searched on small datasets to large datasets often bears no performance guarantee. 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 neural architecture search (EAT-NAS). In our implementations, 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 the 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. What we propose is not only a NAS method but a mechanism for architecture-level transfer.

In our experiments, we obtain two final models EATNet-A and EATNet-B that achieve competitive accuracies, 74.7% and 74.2% on ImageNet, respectively, which also surpass the models searched from scratch on ImageNet under the same settings. For the computational cost, EAT-NAS takes only less than 5 days on 8 TITAN X GPUs, which is significantly less than the computational consumption of the state-of-the-art large-scale NAS methods. [1]

1. Introduction

Designing neural network architectures by human experts often requires tedious trials and errors. To make this process more efficient, many neural architecture search (NAS) methods [30, 18, 17] have been proposed. Despite their remarkable results, most NAS methods require expensive computational resources. For example, 800 GPUs across 28 days are used by NAS [29] on the task of CIFAR-10 [10] image classification. Real-world applications involve lots of large-scale datasets. However, directly carrying out the architecture search on large-scale datasets, *e.g.*, ImageNet [5], requires much more computation cost which limits the wide application of NAS. Although some accelerating methods have been proposed [30, 18, 17], few of them directly explore on large-scale tasks.

From lots of previous works, *e.g.* VGGNet [21], GoogLeNet [22], ResNet [8], *etc.*, a neural architecture that has good performance on one dataset usually performs well on other datasets or tasks. PNAS [12] suggests the transfer capability of the searched architectures by measuring the correlation between performance on CIFAR-10 and ImageNet for different neural architectures. Most existing NAS methods [30, 18, 12] search for architectures on a small dataset, *e.g.*, CIFAR-10 [10], and then apply these architectures directly on a large dataset, *e.g.*, ImageNet, with the architectures adjusted manually. Normally, when transferring to the large dataset, both the number of stacked cells / layers and filters in the network will be enlarged. The searched cell structures or layer operations remain unchanged. However, due to the dataset bias [24] between the small dataset and the large dataset, the best representation of the neural architecture differs between datasets. Moreover, because of the lower resolution and the limited number of training images and data categories in the small dataset, the effectiveness of the architecture or the cells / layers of it searched on the small dataset degrades, when being directly applied on the large dataset or transferred with only the depth and width adjusted by handcraft.

In this work, we propose a transfer learning solution to

---

[1]Our pretrained models and the evaluation code are released at
https://github.com/JaminFong/EAT-NAS
  * The work was done during an internship at Horizon Robotics.
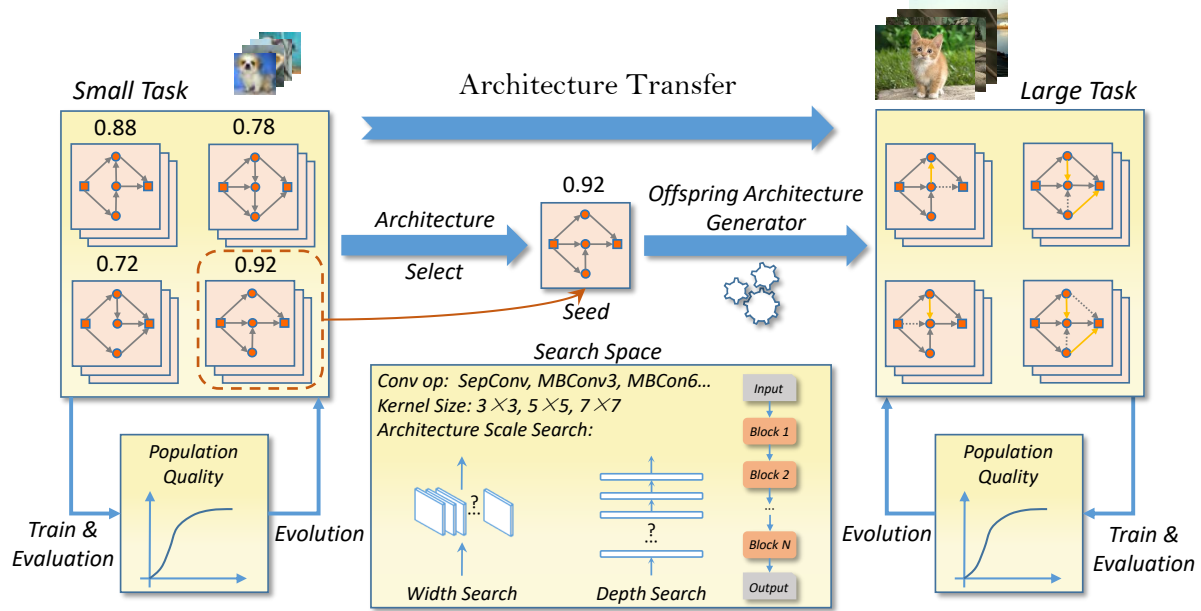 † Equal contributions.

Figure 1: The framework of Elastic Architecture Transfer for Neural Architecture Search (EAT-NAS). We firstly search for the basic architecture on the small-scale task and then search on the large-scale task with the basic architecture as the seed of the new population initialization.

the above problem. What we propose is not only a NAS method but a common mechanism for architecture-level transfer learning. We define the elements in the neural architecture design as *architecture primitives*, *e.g.* the number of filters, the number of layers, operation types, the connection mode, kernel sizes. Our proposed elastic architecture transfer (EAT) is a mechanism which automatically transfers the neural architecture to a large-scale dataset, as is shown in Fig. 1. It is elastic because all the architecture primitives are adjusted or fine-tuned automatically when transferring the architecture to another dataset.

In our implementation, we choose *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 is easy to transfer to another architecture population on the large dataset. We adopt a two-stage search process to achieve the architecture transfer between different datasets. Firstly, we carry out the first search stage on a small dataset. When the first search process is finished, we choose the best architecture in the population as the seed which is called *basic architecture*. Secondly, 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 carry on 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 architec-

ture 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 mechanism of the architecture-level transfer supplies a new aspect for transfer learning. The transfer mechanism could not only be deployed to the EA-based NAS method, but also the gradient or RL based ones. Moreover, EAT could be available for various tasks and datasets in practical problems.

Our contribution can be summarized as follows:

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

2) Through the experiments on the large-scale dataset, *i.e.*, ImageNet, we show the efficiency of our method by cutting the search cost to only less than 5 days on 8 TITAN X GPUs, about 106x lower than the cost for MnasNet estimated based on [23].

3) Our searched architectures achieve remarkable ImageNet performance that is comparable to MnasNet which searches directly on the full dataset with huge computational cost (74.7% vs 74.0%).

2. Related Work and Background

2.1. Neural Architecture Search

Generating neural architectures automatically has aroused great interests in recent years. In NAS [29], an RNN network trained with reinforcement learning is utilized as a

controller to determine the operation type, parameter, and connection for every layer in the architecture. Although NAS [29] achieves impressive results, the search process is incredibly computation hunger and hundreds of GPUs are required to generate a high-performance architecture on CIFAR-10 datasets. Based on the NAS method in [29], many novel methods have been proposed to improve the efficiency of architecture search like finding out the blocks of the architecture instead of the whole network [30, 28], progressive search with performance predictor [12], early stopping strategy in [28], and parameter sharing in [17]. Though they have achieved impressive results, the search process is still computation hunger and extremely hard when the searched datasets are in large-scale, *e.g.*, ImageNet.

Another stream of NAS works utilizes the evolutionary algorithm to generate coded architectures [16, 18, 15]. Modifications to the architecture (filter sizes, layer numbers, and connections) serve as the mutation in the search process. Though they have achieved state-of-the-art results, the computation cost is also far beyond affordable.

Besides, gradient-based NAS methods [14, 26, 1] become popular. Gradient-based methods discard the black-box searching method and introduce architecture parameters, which are updated on the validation set by gradient descent, for every path of the network. A *softmax* classifier is utilized to select the path and the operation for each node. 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. Though gradient-based NAS methods achieve great performance 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. This suppresses the expansibility of the search space, *e.g.*, it is not easy to search for the width of the architecture by gradient-based methods. For our evolutionary algorithm based EAT method over the discrete search space, the architecture can be encoded and is easier to be extended to diverse search spaces.

Recently, MnasNet [23] proposes to search directly on large-scale datasets with latency 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 the nearly whole training set for 5 epochs and evaluated on the 50K validation set. It takes about 91K GPU hours as is estimated according to the description in [23].

There is a work [25] that combines transfer learning with 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

---

Algorithm 1: Evolutionary Algorithm

input : population size P, sample size S, dataset D
output: the best model M_{best}

1   $\mathbb{P}^{(0)} \leftarrow$ initialize(P)
2   **for** $j < P$ **do**
3     $M_j.acc \leftarrow$ train-eval(M_j, D)
4     $M_j.score \leftarrow$ comp-score($M_j, M_j.acc$)
5   **end**
6   $Q^{(0)} \leftarrow$ comp-quality($\mathbb{P}^{(0)}$)
7   **while** $Q^{(i)}$ *not converge* **do**
8     $S^{(i)} \leftarrow$ sample($\mathbb{P}^{(i)}$, S)
9     $M_{best}, M_{worst} \leftarrow$ pick($S^{(i)}$)
10    $M_{mut} \leftarrow$ mutate(M_{best})
11    $M_{mut}.acc \leftarrow$ train-eval(M_{mut}, D)
12    $M_{mut}.score \leftarrow$ comp-score(M_{mut}, $M_{mut}.acc$)
13    $\mathbb{P}^{(i+1)} \leftarrow$ remove M_{worst} from $\mathbb{P}^{(i)}$
14    $\mathbb{P}^{(i+1)} \leftarrow$ add M_{mut} to $\mathbb{P}^{(i)}$
15    $Q^{(i+1)} \leftarrow$ comp-quality($\mathbb{P}^{(i+1)}$)
16   i ++
17   **end**
18   $M_{best} \leftarrow$ rerank-topk($\mathbb{P}_{best}$, k)

---

architecture searched on the small dataset can be transferred to the large dataset fast and precisely. EAT-NAS obtain models with competitive performance and much less computational resources than search from scratch.

2.2. Evolutionary Algorithm based NAS

Evolutionary algorithm (EA) is widely utilized in NAS [16, 18, 15]. As is 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 the accuracy of the model.

Following the Pareto-optimal problem [4], we use $acc \times [size/T]^{\omega}$ 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 is the target model size and ω is 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 model with the best score and the worst one are picked up. The mutated model is obtained by adding some transformation to the one with the best score. The mutated model is trained, evaluated and added to the population with its score. The worst model is removed meanwhile. The above search process is called *tournament selection* [7]. Finally, the top-k performing models are retrained to select the best one. Our architecture search method is based on the evolutionary algorithm.
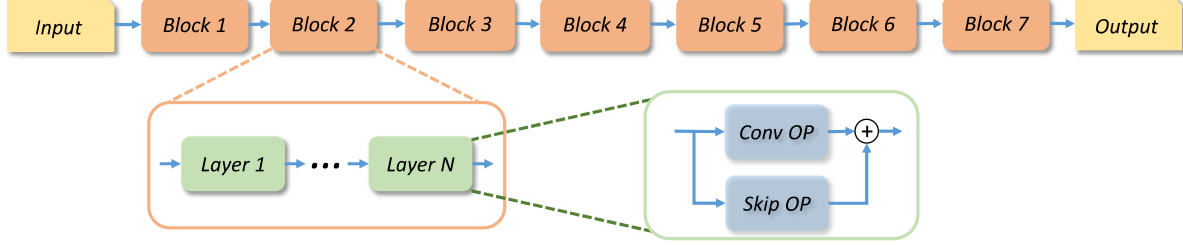
Figure 2: Search Space. During search, all the blocks are concatenated to constitute the whole network architecture. Each block consists of several layers and is represented by the following five primitives: convolutional operation type, kernel size, skip connection, width and depth.

3. Method

To apply an architecture to large-scale tasks, most architecture search methods [18, 17, 13, 30] merely rely on prior knowledge of human experts. They transfer an architecture manually with only expanding the depth and width by multiplication or direct addition. Different from these conventional transfer methods, we propose an elastic architecture transfer (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, the structure, the depth and width of the architecture. EAT accelerates the large-scale search process by making use of the knowledge from the base 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

Fig. 1 illustrates the process of EAT. The two search process on the small and the large datasets are based on the same *search space* (Section 18). We firstly search for a set of top-performing architectures on the small dataset, such as CIFAR-10. To get better performing architectures, we search for the *architecture scale* (Section 3.3) with the help of the width and depth factor. We design a criterion as *population quality* (Section 3.4) to better evaluate the model population. Then we retrain the top-performing models and choose the best one as the basic architecture. Secondly, 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* (Section 3.5) to produce architectures of the new population. Then we 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 obtains better performing models than carrying it out from scratch, which is benefited from the useful information of the basic architecture. Section 4.3 displays the results of contrast. Finally, we select the best one from the top-k performing models in the population by retraining

---

Algorithm 2: Elastic Architecture Transfer

> **input** : datasets D_1, D_2, population size P
> **output:** the target architecture $Arch_{target}$

1 // initialize the population on D_1
2 $\mathbb{P}_1 \leftarrow$ initialize(P)
3 evolve($\mathbb{P}_1, D_1$)
4 $Arch_{basic} \leftarrow$ rerank-topk & select($\mathbb{P}_1, k$)
5 // initialize the population on D_2
6 **for** $i < P$ **do**
7 $\quad Arch_i \leftarrow$ arch-perturbation($Arch_{basic}$)
8 $\quad \mathbb{P}_2$.append($Arch_i$)
9 **end**
10 evolve($\mathbb{P}_2, D_2$)
11 $Arch_{target} \leftarrow$ rerank-topk & select($\mathbb{P}_2, k$)

---

them on the full large-scale dataset. Algorithm 2 displays the whole procedure for the elastic architecture transfer.

3.2. Search Space

A well-designed search space is essential for NAS. Inspired by MnasNet [23], we employ an architecture search space with MobileNetV2 [19] as the backbone. As Fig. 2 shows, 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 one block could be parsed as follows:

- *Conv operation*: depthwise separable convolution (SepConv) [3], mobile inverted bottleneck convolution (MBConv) with diverse expansion ratios {3,6} [19].
- *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 the input width, $factor_{width} = N_o/N_i$, [0.5, 1.0, 1.5, 2.0].
- *Depth factor*: the number of layers per block, [1, 2, 3, 4].

Besides, the 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

Algorithm 3: Parameter sharing on the width-level

input : kernel $\mathbf{K}_l$ in layer l, the original kernel $\mathbf{K}_o$
output: kernel $\mathbf{K}_l$ in layer l
1 $ch_{in}^s \leftarrow \min(\ ch_{in}^l,\ ch_{in}^o\ )$
2 $ch_{out}^s \leftarrow \min(ch_{out}^l,\ ch_{out}^o)$
3 $\mathbf{K}_l \leftarrow \mathbf{K}_o(w^o,\ h^o,\ ch_{in}^s,\ ch_{out}^s)$

into several blocks, a whole architecture is presented as a block set $Arch = \{B^1, B^2, ..., B^n\}$. Each block consists of the above five primitives, which is encoded by a tuple $B^i = (conv, kernel, skip, width, depth)$. Every manipulation for the neural architecture is performed based on the model code.

3.3. Architecture Scale Search

Most NAS methods [30, 17, 12, 18] treat the scale of the architecture as a fixed element based on the prior knowledge from human experts. The scale, the depth and width, of the architecture, usually affects the performance of the architecture. 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 in Net2Net [2], namely Net2WiderNet and Net2DeeperNet, we propose a modified parameter sharing method on model training. When initializing parameters for a network, the proposed algorithm traverses the operation for each layer. If the operation type and the kernel size of the layer consist with that of the shared model, the parameter sharing is applied on this layer. We introduce two parameter sharing behaviors on the width and depth respectively.

Parameter sharing on the width-level By sharing the parameters, we desire to inherit as more information as possible from the former model. For the convolutional layer, we suppose the convolutional kernel of the lth layer $\mathbf{K}_l$ has the shape of $(w^l,\ h^l,\ ch_{in}^l,\ ch_{out}^l)$, 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 $\mathbf{K}_o$ has the shape of $(w^o,\ h^o,\ ch_{in}^o,\ ch_{out}^o)$, we carry out sharing strategy in Algorithm 3. In addition to the shared parameters, the rest part of $\mathbf{K}_l$ is randomly initialized.

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

i $l_u > l_w$:

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

ii $l_u \leq l_w$:

$$\mathbf{U}[1, 2, ..., l_u] = \mathbf{W}[1, 2, ..., l_u] \quad (2)$$

where Γ is a random weight initializer based on a normal distribution.

3.4. Population Quality

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

Therefore, until the objective of the population converges to an optimal solution, the mean score of models should be as high while the variance of model scores as low as possible. This issue could be treated as a Pareto-optimal problem [4]. To approximate the Pareto optimal solution, we utilize a target function, population quality, as follows,

$$Q = score_{mean} \times \left[\frac{std}{target_{std}} \right]^\omega \quad (3)$$

where ω is the weight factor defined as follows,

$$\omega = \begin{cases} \alpha, & \text{if } std < target_{std} \\ \beta, & \text{otherwise} \end{cases} \quad (4)$$

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

In Eq. 3, $score_{mean}$ denotes the mean score of models in the population, std denotes the standard deviation of model scores and $target_{std}$ is the target std. We set $\alpha = \beta = -0.07$ to assign the value to ω. After the evolution, we pick up the best-quality population and retrain top-k models.

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
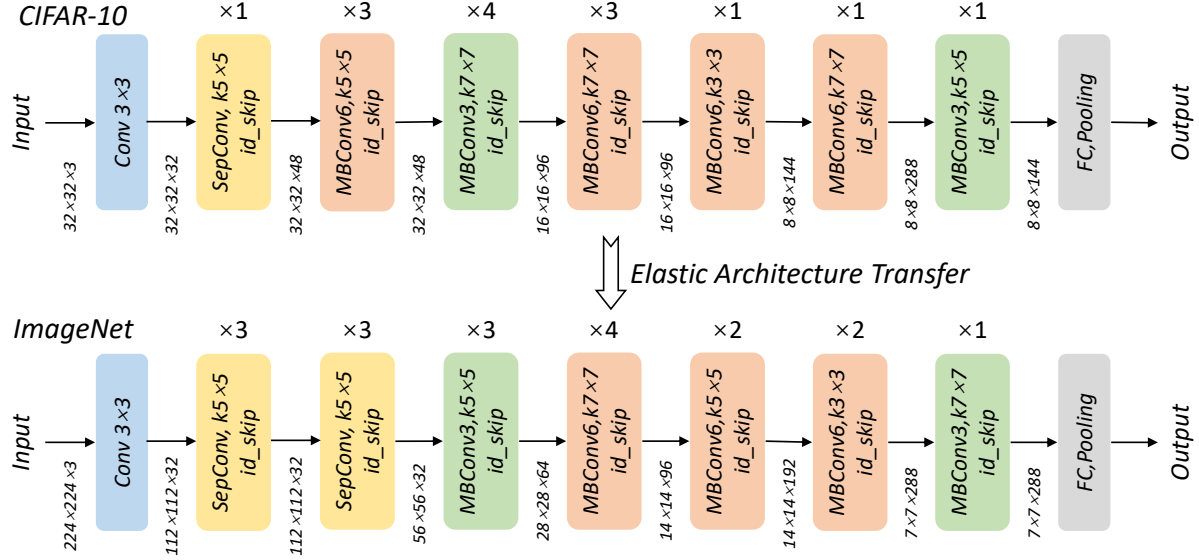
Figure 3: The architectures searched by EAT-NAS. The upper one is the basic architecture searched on CIFAR-10. And the nether one is the architecture EATNet-A searched on ImageNet which is transferred from the basic architecture.

Algorithm 4: Architecture Perturbation Function

input : basic architecture $Arch_b$, search space $\mathbb{S}$, number of blocks N_{blocks}, primitives $prims$

output: perturbed architecture $Arch_p$

1   $Arch_p \leftarrow \texttt{copy}(Arch_b)$
2   **for** $j < N_{\text{blocks}}$ **do**
3     $prim \leftarrow \texttt{rand-select}(prims)$
4     $value \leftarrow \texttt{rand-generate}(prim, \mathbb{S})$
5     $B_t^j \leftarrow \texttt{get-block}(Arch_t, \texttt{j})$
6     $B_t^j[\text{type}] \leftarrow value$
7   **end**

as described in Section 18. We randomly select one type of the five primitives to perturb. For the selected primitive, we generate a new value of the primitive stochastically within the restriction of our search space and replace the existing one.

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 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 starts the same procedure as described in Algorithm 1. Moreover, the architecture transformation function is utilized as the mutation operation in 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 which demonstrate the effectiveness of the proposed EAT-NAS method. Other more implementation details are displayed in the appendix of the Supplementary Materials.

4.1. Search on CIFAR-10

The experiments on CIFAR-10 are divided into two steps including architecture search and architecture evaluation. CIFAR-10 consists of 50,000 training images and 10,000 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 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 x 40 and randomly flip them horizontally.

During the search process, we set the population size as 64 and the sample size as 16. Every model generated during evolution is trained for 1 epoch and is evaluated on the separate validation set. We mutate about 1,400 models during the total evolution. The number of model parameters is the sub-optimizing objective with the target as 3.0M. Each model on CIFAR-10 consists of 7 blocks and the downsampling operations are carried out in the third and fifth blocks. The initial number of channels is 32. For evaluation, we retrain top-8 models searched on CIFAR-10 and select the one with the best accuracy as the basic model. Since the CIFAR-10 results are subject to high variance even with exactly the same setup [13], 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

Table 1: ImageNet classification results in the mobile setting. The results of manual-design models are in the top section, other NAS results are presented in the middle section, and the result of our models are in the bottom section.

Model	#Params (M)	#Mult-Adds (M)	Top-1/Top-5 Acc(%)	Search Dataset	Search Time (GPU hours)
MobileNet-v1 [9]	4.2	575	70.6 / 89.5	-	-
MobileNet-v2 [19]	3.4	300	71.7 / -	-	-
MobileNet-v2 (1.4)[20]	6.9	585	74.7 / -	-	-
ShuffleNet-v1 2x [27]	$\approx$ 5	524	73.7 / -	-	-
NASNet-A [30]	5.3	564	74.0 / 91.6	CIFAR-10	48K
NASNet-B [30]	5.3	488	72.8 / 91.3	CIFAR-10	48K
NASNet-C [30]	4.9	558	72.5 / 91.0	CIFAR-10	48K
AmoebaNet-A [18]	5.1	555	74.5 / 92.0	CIFAR-10	76K
AmoebaNet-B [18]	5.3	555	74.0 / 91.5	CIFAR-10	76K
AmoebaNet-C [18]	5.1	535	75.1 / 92.1	CIFAR-10	76K
PNASNet-5 [12]	5.1	588	74.2 / 91.9	CIFAR-10	6K
MnasNet [23]	4.2	317	74.0 / 91.8	ImageNet	91K
MnasNet (our impl.)	4.2	317	73.3 / 91.3	ImageNet	91K
DARTS [14]	4.7	574	73.3 / 91.3	CIFAR-10	96
SNAS [26]	4.3	522	72.7 / 90.8	CIFAR-10	36
EATNet-A	5.1	563	74.7 / 92.0	CIFAR-10 to ImageNet	856
EATNet-B	5.3	551	74.2 / 91.8	CIFAR-10 to ImageNet	856
EATNet-S	4.6	414	72.7 / 91.1	CIFAR-10 to ImageNet	856

[*] To avoid any discrepancy between different implementations or training settings, we try our best to reproduce the performance of MnasNet [23]. The highest accuracy we could reproduce is 73.3% for MnasNet. All the training settings we use are the same as that we reproduce MnasNet for the fair comparison.

deviation of 0.05) with only 2.04M parameters. The architecture of the basic model is shown in Fig. 3.

4.2. Transferring to ImageNet

We use the architecture of the basic model searched on CIFAR-10 as the seed to generate models of the 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 which contains 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 basic architecture. During architecture search, we train every model for one epoch. The number of multiply-add operations is set as the sub-optimizing objective during evolution with the target as 500M. The input resolution of the network is set to 224×224. 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.

As shown in Fig. 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 [23], the controller samples about 8K models during architecture search, 50 times the amount of

ours. With much less computational resources, EAT-NAS achieves comparable results on ImageNet as in Table 1.

Fig. 3 displays the basic architecture and the architecture of EATNet-A. From the figure, we find that though based on the basic architecture, there are some transformations on 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 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 hours on 4 GPUs to search for the basic architecture on CIFAR-10 and 4 days on 8 GPUs to transfer to ImageNet. Though DARTS [14] and SNAS [26] take less search time, they only search on the small dataset, CIFAR-10. And our ImageNet performances surpass them clearly.

4.3. 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 in both search and evaluation process. The
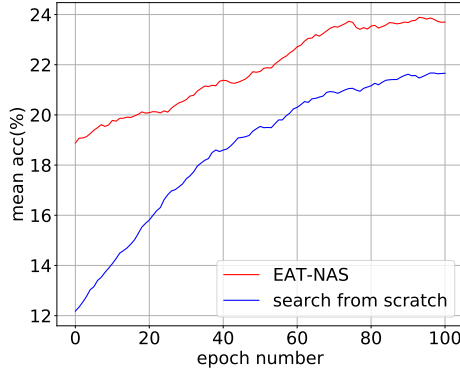
(a) The mean accuracy of the models in the population.
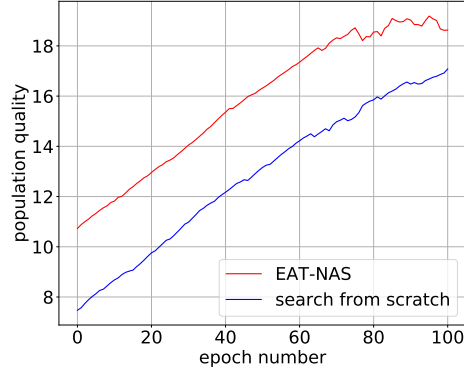
(b) The population quality.

Figure 4: Comparing the evolution process on ImageNet of EAT-NAS and search from scratch.
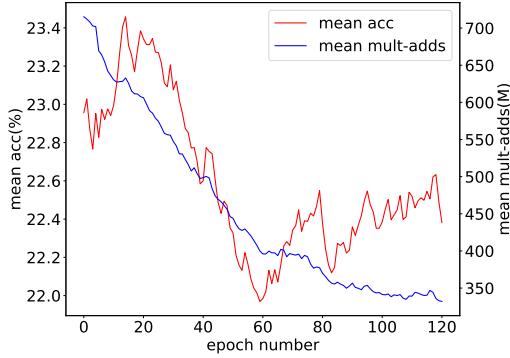

Figure 5: The mean accuracy and the mean multiply-adds of the models during the search on ImageNet whose basic architecture has worse performance on CIFAR-10.

search process takes the same GPU hours as well. Fig. 4 shows the mean accuracy of models in the population and the population quality of EAT-NAS and search from scratch on ImageNet for equal search epochs. We find out 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 2, 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. The model EATNet-S surpasses that searched from scratch obviously.

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 of EAT-NAS. The performance of the basic model is shown in Table 2 as well. The basic model achieved a high validation accuracy but with a very large number of multiply-add operations. Its high validation accuracy demonstrates the superior quality of the initialization seed in the transfer process. Comparing EATNet-A with Basic

Table 2: The results of contrast experiments on ImageNet. 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 worse-performing basic architecture.

Model	#Params (M)	#Mult-Adds (M)	Top-1/Top-5 Acc (%)
SS	5.53	447	71.8 / 90.6
Basic Model	3.14	886	74.3 / 92.0
Model-B	3.20	405	71.7 / 90.5
EATNet-A	5.08	563	74.7 / 92.0
EATNet-B	5.25	551	74.2 / 91.8
EATNet-S	4.63	414	72.7 / 91.1

Model, we find that not only does EAT promote the accuracy of the model, but also optimizes the size 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 Fig. 5 shows, the basic architecture with worse performance has a negative impact on transfer. The mean accuracy degrades in the preliminary epochs. This experiment demonstrates the importance of a well-performing basic architecture for transfer. We retrain 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 Table 2 displays, models searched by EAT-NAS surpass that searched with the bad-performing basic architecture. We attribute the results to the 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, such as reinforcement learning and gradient-based NAS. In addition, EAT-NAS can also be utilized to search for neural architectures in other computer vision tasks like detection, segmentation, and tracking, which we also leave for future work.

Acknowledgement

References

[1] H. Cai, L. Zhu, and S. Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019. 3

[2] T. Chen, I. J. Goodfellow, and J. Shlens. Net2net: Accelerating learning via knowledge transfer. *CoRR*, abs/1511.05641, 2015. 5

[3] F. Chollet. Xception: Deep learning with depthwise separable convolutions. pages 1800–1807, 2017. 4

[4] K. Deb. Multi-objective optimization. In *Search methodologies*, pages 403–449. 2014. 3, 5

[5] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 1

[6] T. Devries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017. 10

[7] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In *FGA*, pages 69–93. 1990. 3

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 1, 10

[9] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. 7

[10] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Technical report.*, 1(4):1–7, 2009. 1

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012. 10

[12] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. L. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *ECCV*, pages 19–35, 2018. 1, 3, 5, 7, 10

[13] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. *CoRR*, abs/1711.00436, 2017. 4, 6

[14] H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. 3, 7

[15] Z. Lu, I. Whalen, V. Boddeti, Y. D. Dhebar, K. Deb, E. D. Goodman, and W. Banzhaf. NSGA-NET: A multi-objective genetic algorithm for neural architecture search. *CoRR*, abs/1810.03522, 2018. 3

[16] R. Miikkulainen, J. Z. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat. Evolving deep neural networks. *CoRR*, abs/1703.00548, 2017. 3

[17] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. In *ICML*, pages 4092–4101, 2018. 1, 3, 4, 5, 10

[18] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. *CoRR*, abs/1802.01548, 2018. 1, 3, 4, 5, 7, 10

[19] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. volume abs/1801.04381, 2018. 4, 7

[20] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018. 7

[21] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 1, 10

[22] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 1, 10

[23] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le. Mnasnet: Platform-aware neural architecture search for mobile. *CoRR*, abs/1807.11626, 2018. 2, 3, 4, 7

[24] T. Tommasi, N. Patricia, B. Caputo, and T. Tuytelaars. A deeper look at dataset bias. In *Domain Adaptation in Computer Vision Applications*, pages 37–55. Springer, 2017. 1

[25] C. Wong, N. Houlsby, Y. Lu, and A. Gesmundo. Transfer learning with neural automl. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8356–8365. Curran Associates, Inc., 2018. 3

[26] S. Xie, H. Zheng, C. Liu, and L. Lin. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations*, 2019. 3, 7

[27] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017. 7

[28] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu. Practical block-wise neural network architecture generation. In *CVPR*, 2018. 3

[29] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016. 1, 2, 3

[30] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017. 1, 3, 4, 5, 7, 10

A. Experiment Details

We provide more experiment details in this appendix. Our experiments mainly consist of two stages, searching for the basic architecture on CIFAR-10 and then transferring it to ImageNet.

A.1. Search on CIFAR-10

For each model during the search ,the depth and width of the mutated model would vary in an extremely wide range within our search space if there is not any restriction. 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 SGD optimizer with the learning rate of 0.0256 (fixed during the search), momentum 0.9, and weight decay 3×10^{-4}. The search experiment is carried out on 4 GPUs which takes about 22 hours. 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. Other hyperparameters remain the same as that in the search process. Following existing works [17, 30, 12, 18], additional enhancements include cutout [6] with the length of 16, and auxiliary towers with weight 0.4. The training of the searched model takes around 13 hours on two GPUs.

A.2. Transferring to ImageNet

During architecture search, we train every model for one epoch with batch 128 and a learning rate of 0.05. Following GoogLeNet [22], the input images are sampled as various sized patches of the image whose size is distributed between 20% and 100% of the image area with aspect ratio constrained to the interval $[\frac{3}{4}, \frac{4}{3}]$. The input resolution of the network is set to 224×224. Other hyperparameters of the search process are the same as that on CIFAR-10. For each model, 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 200 epochs, using standard SGD optimizer with momentum rate set to 0.9, weight decay 4×10^{-5} and 0.1-weighted label smoothing. Our batch size is 256 on 4 GPUs. The initial learning rate is 0.1 and it decays in a polynomial schedule to 1×10^{-4}. We follow ResNet [8] to do the data augmentation. We resize the original input images with its shorter side randomly sampled in [256, 480] and $1 \pm \frac{1}{4}$ aspect ratio for scale augmentation [21]. A 224×224 patch is randomly cropped from an image or its horizontal flip, with the per-pixel mean subtracted [11]. For the last 20 epochs, we only keep the $256 \times N$ for the resizing scale to fine-tune the model. To avoid any discrepancy between different implementations or training settings, we try our best to reproduce the performance of MnasNet (74.0% on ImageNet). The highest accuracy we could reproduce is 73.3% for MnasNet. All the training settings we use are the same as that we reproduce MnasNet for the fair comparison.