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-scale tasks 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 (the width and depth of architectures are taken into consideration as well), e.g.*, CIFAR-10, and the best is chosen as the basic architecture. Then the whole architecture is transferred with elasticity. We accelerate the search process on a large-scale dataset,* e.g.*, the whole ImageNet dataset, 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, 73.8% and 73.7% on ImageNet, respectively, which also surpass the models searched from scratch on ImageNet under the same settings. For 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. 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 [16, 26, 15] 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 [25] on the task of CIFAR-10 [9] image classification. Real-world applications often involve large-scale datasets or tasks. However, directly applying the architecture search to large-scale datasets, *e.g.*, ImageNet [5], requires much more computation costs which limit the wide application of NAS. Although some accelerating methods have been proposed [26, 16, 15], few of them directly explore on large-scale tasks.

Most existing NAS methods [26, 16, 11] search architectures in small datasets, *e.g.*, CIFAR-10 [9], and then apply these architectures directly on ImageNet with the depth and width adjusted manually. This mechanism has been widely used in the area of NAS. However, the architectures searched on small datasets have no performance guarantee on large datasets because of the large gap between different domains. For example, the optimal factors of architectures like depth and width might be different in other datasets. Besides, other architecture factors including the topological structure, operations, might heavily depend on the image size of datasets. It also leads to the poor performance of this kind of direct use on large datasets.

In this work, we propose a reasonable solution to these limitations. What we propose is not only a method but a common mechanism for architecture-level transfer. Specially, our method transfers architectures learned from small datasets to large ones and fine-tunes these architectures on large tasks. We employ an EA-based NAS method, *tournament selection*, where neural architectures are first searched on small datasets as in previous works [17]. Then we carry on the second search stage on large datasets and use the architectures searched on small datasets as initial seeds. This mechanism is a kind of architecture level transfer and would

* The work was done during an internship at Horizon Robotics.
† Equal contributions.

be usable for various tasks and datasets.

Our contribution can be summarized as follows:

1) We propose an elastic architecture transfer mechanism that can bridge small and large datasets for neural architecture search.

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.

3) Our searched architectures achieve remarkable ImageNet performance that is comparable to MnasNet which searches directly on the full dataset with huge computational cost (73.8% vs 74.0%).

2. Related work and Background

2.1. Neural Architecture Search

Generating neural architectures automatically has arouse great interests in recent years. In NAS [25], a RNN network trained with reinforcement learning is utilized as a controller to determine the type, parameter and connection for every layer in the architecture. Although NAS [25] 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 NAS [25], 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 [26, 24], progressive search with performance predictor [11], early stopping strategy in [24] or parameter sharing [15]. Though they achieve impressive results, the search process is still computation hunger and extremely hard for large datasets.

Another stream of NAS works utilize the evolutionary algorithm to generate coded architectures [14, 16, 13]. Modifications to the architecture (filter sizes, layer numbers, and connections) serve as mutation in the search process. Though they achieve state-of-the-art results, the computation cost is also far beyond affordable. Recently a novel method, DARTS [1], discards the black-box searching method and introduces architecture parameters, which are updated on the validation set, for every path of the network. A *softmax* classifier is utilized to select the path and the operation for each node.

Recently, MnasNet [21] proposes to search directly on large-scale datasets with latancy optimization of the architecture. Although MnasNet [21] successfully generates high-performance architectures with promising inference speed, it requires huge computational resources as every sampled model needs to be trained and evaluated on the full ImageNet dataset during the whole search process.

Algorithm 1: Evolutionary Algorithm

input : model population $\mathbb{P}$, population size P, population quality Q, sample size S, model M, dataset D

output: the best model M_{best}

$\mathbb{P}^{(0)} \leftarrow$ `initialize(`P`)`
while $i < P$ **do**
  $M_i.acc \leftarrow$ `train-eval(`M_i`,` D`)`
  $M_i.score \leftarrow$ `comp-score(`$M_i, M_i.acc$`)`
end
$Q^{(0)} \leftarrow$ `comp-quality(`$\mathbb{P}^{(0)}$`)`
while $Q^{(i)}$ *not converge* **do**
  $S^{(i)} \leftarrow$ `sample(`$\mathbb{P}^{(i)}$`,` S`)`
  $M_{best}, M_{worst} \leftarrow$ `pick(`$S^{(i)}$`)`
  $M_{mut} \leftarrow$ `mutate(`M_{best}`)`
  $M_{mut}.acc \leftarrow$ `train-eval(`M_{mut}`,` D`)`
  $M_{mut}.score \leftarrow$ `comp-score(`M_{mut}`,` $M_{mut}.acc$`)`
  $\mathbb{P}^{(i+1)} \leftarrow$ `remove(`$\mathbb{P}^{(i)}$`,` M_{worst}`)`
  $\mathbb{P}^{(i+1)} \leftarrow$ `add(`$\mathbb{P}^{(i+1)}$`,` M_{mut}`)`
  $Q^{(i+1)} \leftarrow$ `comp-quality(`$\mathbb{P}^{(i+1)}$`)`
end
$M_{best} \leftarrow$ `rerank-topk(`$\mathbb{P}_{best}$`,` k`)`

Our proposed elastic architecture transfer method focuses on transferring the architecture searched on the small-scale dataset to large-scale dataset fast and precisely. EAT-NAS could 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 [14, 16, 13]. As is summarized in Algorithm 1, the search process is based on the population of various models. 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.

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. We obtain the mutated model 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 strategy is based on evolutionary algorithm as well.

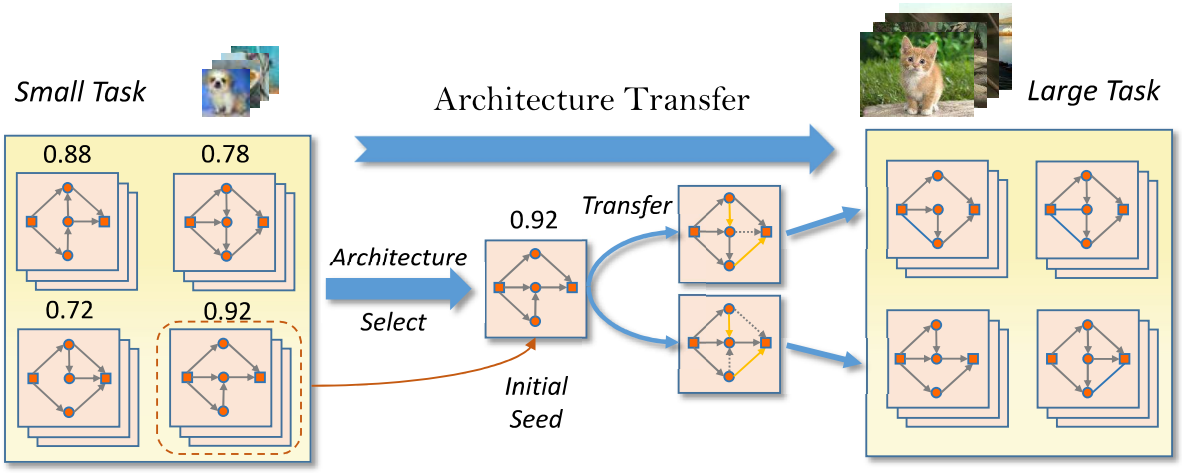


Figure 1: The framework of Elastic Architecture Transfer for Neural Architecture Search (EAT-NAS). We firstly search for the basic model on the small-scale task and then search on the large-scale task with the basic model as the initialization seed of the new population.

3. Elastic Architecture Transfer

When applying an architecture to large-scale tasks, most architecture search methods [16, 15, 12, 26] merely rely on prior knowledge of human experts. They transfer an architecture manually, such as expanding the depth and width by a multiplication or direct addition. Different from these conventional transfer methods, we propose an elastic architecture transfer (EAT) method. In EAT, all elements of the architecture on the new task, including the structure, scale, operations, *etc*., are fine-tuned from the basic architecture. EAT accelerates the large-scale search process by making use of the basic architecture searched on the small-scale task and adjusting it to the large-scale task with elastic transfer.

3.1. Framework

As Fig. 1 illustrates, we firstly search for a set of top-performing architectures on the small dataset, such as CIFAR-10. We design a criterion *population quality* (Section 3.3) to better evaluate the model population during evolution. And we search the *architecture scale* (Section 3.4) under our *search space* (Section 3.2) as well. Then we retrain the top-performing models and choose the best one as the basic model in the elastic transfer. Secondly, we start the large-scale task architecture search with the basic architecture as the initialization seed of the new model population. We design the *offspring architecture generator* (Section 3.5) to produce the new architectures. And then we continue architecture search on the large-scale task based on the generated population. In this way, the process of search on the new task converges obviously faster than carrying it out from scratch, which is benefited from the useful information of the basic architecture. Finally, the best one is selected from the top-k performing models in the population by retraining them on the full large-scale dataset. Following MnasNet [22], we use the Pareto-optimal method to perform multi-objective optimization for both the accuracy and model size.

3.2. Search Space

A well-designed search space is essential for the architecture search. Inspired by MnasNet [22], we employ an architecture search space with MobileNetV2 [18] as the backbone. As Fig. 2 shows, the network is divided into several blocks which can be different with each other. Each block consists of several layers. Each layer represents one type of operation which is repeated for particular times in one block. Specifically, 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} [18].
- *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, [0.5, 1.0, 1.5, 2.0].
- *Depth factor*: the number of layers per block, [1, 2, 3, 4].

The down sampling and expand width operation are carried out in the first layer of blocks.

To manipulate the neural architecture more conveniently, we encode every architecture in our method following the format defined in the search space. As a network could be

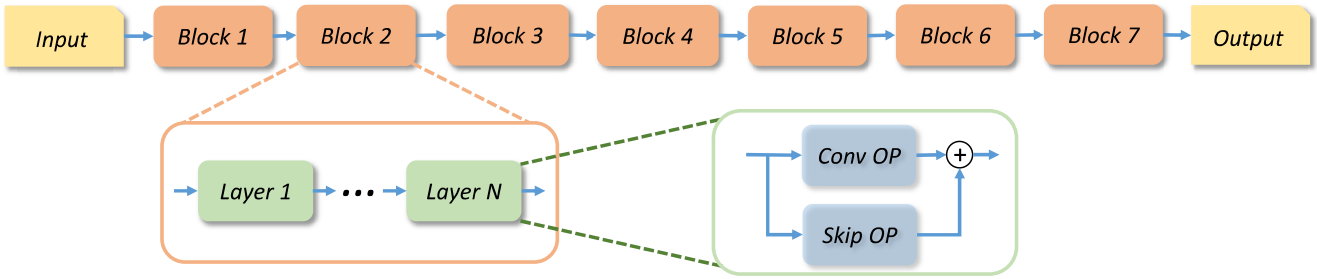


Figure 2: Search Space. During search, all the blocks are concatenated to constitute the whole neural architecture. Each block consists of several layers. The representation of one block includes five elements: convolutional operation, kernel size, skip connection, width and depth.

separated 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 elements, which is encoded by a tuple $B^i = (conv, kernel, skip, width, depth)$. Every manipulation on the neural architecture is carried out based on the model code.

3.3. Population Quality

During the evolution of the population, we hope that the performance of models could get more and more desirable. Nevertheless, it is insufficient to evaluate whether a population meet the demands merely depending on the mean accuracy of models or the variance of accuracies. Especially when parameter sharing is used in the search process, it is hard to judge the degree of convergence by only observing the accuracy tendency, because accuracy gains could derive from both parameters sharing and better model performance. For comprehensive assessment, we introduce the variance of accuracies rather than merely considering the mean accuracy.

Thus until the objective of the population converges to an optimal solution, the mean accuracy of models should be as high as possible while the variance of model accuracies should be as low as possible. This issue could be treated as a Pareto-optimal problem [4]. To approximate Pareto optimal solutions, we utilize a target function to describe our optimized objective, which is referred to in MnasNet [22]:

$$obj = obj_{main} \times \left[\frac{obj_{sub}}{target}\right]^{\omega} \tag{1}$$

where ω is the weight factor defined as:

$$\omega = \begin{cases} \alpha, & \text{if } obj_{sub} < target \\ \beta, & \text{otherwise} \end{cases} \tag{2}$$

where α and β could be hyper-parameters in this equation and we could set different values towards different applications.

Algorithm 2: Sharing on the width level

input : layer l kernel K_l, original kernel K_o

$ch^s_{in} \leftarrow \texttt{min}(ch^l_{in}, ch^o_{in})$

$ch^s_{out} \leftarrow \texttt{min}(ch^l_{out}, ch^o_{out})$

$K_l \leftarrow K_o(w^o, h^o, ch^s_{in}, ch^s_{out})$

output: layer l kernel K_l

To assess the quality of the model population precisely, we design a customized function based on Eq. (1) to approximate the quality of the population as follow:

$$Q = acc_{mean} \times \left[\frac{std}{target_{std}}\right]^{\omega} \tag{3}$$

where acc_{mean} denotes the mean accuracy of models in the population and std denotes the standard deviation of model accuracies. We set α = β = -0.07 to assign the value to ω. After the evolution process, we pick up the best quality population and retrain top-k models.

3.4. Architecture Scale Search

Most neural architecture search methods [15, 26, 11, 16] treat the scale of architecture as a fixed element based on the prior knowledge of human beings. The best representation of the architecture changes with different network scales. To obtain better performance architectures, our search method includes the scale search of the architecture, the width and depth, as well.

As mentioned in Section 3.2, the search space includes the width and depth factors. The width factor denotes the expansion ratio of filter numbers: $factor_{width} = \frac{N_o}{N_i}$, and depth factor denotes the number of layers per block.

To accelerate the architecture search process, we employ the parameter sharing method on each model during searching. Net2Net [2] proposed two specific function-preserving transformations, namely Net2WiderNet and Net2DeeperNet, which accelerate the training of a significantly larger neural network. The concept of Net2Net could

Algorithm 3: Architecture Transformation Function

input : basic architecture $Arch_b$, search space $\mathbb{S}$, number of blocks N_{blocks} and number of elements N_{elem}

output: transformed architecture $Arch_t$

$Arch_t \leftarrow$ `copy` $(Arch_b)$

for $j < N_{\text{blocks}}$ and $B_t^j \in Arch_t$ **do**

  type $\leftarrow$ `rand-select` (N_{elem})

  value $\leftarrow$ `generate` (type, $\mathbb{S}$)

  B_t^j[type] $\leftarrow$ value

end

be consider as parameters sharing. Different from Net2Net, we adopt a verified parameters sharing method on the model training. For an untrained model, we traverse each layer of it. If the operation and the kernel size of the layer consist with that of the shared model, then we apply parameters sharing on this layer. We introduce two parameter sharing behaviors on the width and depth respectively.

Sharing on the width level By sharing the parameters, we desire to inherit as more information as possible from the former model. Especially for the convolutional layer, we suppose the convolutional kernel of the l^{th} 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, 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 as Algorithm 2 shows. In addition to the shared parameters, the rest part of $\mathbf{K}_l$ is randomly initialized.

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 $l_u > l_w$:

$$\mathbf{U}[i] = \begin{cases} \mathbf{W}[i], & \text{if } i < l_w \\ \Gamma(i), & \text{otherwise} \end{cases} \tag{4}$$

ii $l_u < l_w$:

$$\mathbf{U}[1, 2, ..., l_u] = \mathbf{W}[1, 2, ..., l_u] \tag{5}$$

where Γ is a random weight initializer using a normal distribution.

3.5. Offspring Architecture Generator

When transferring the architecture, we design an offspring architecture generator to produce new architectures of the model population.The generator takes the basic architecture as the initialization seed of the new model population. We define a transformation function to derive new architectures by adding some perturbation to the input architecture homogeneously and slightly. Algorithm 3 illustrates the process of the transformation function. In each block of the architecture, there are total five architecture elements (*conv operation, kernel size, skip connection, width factor, depth factor*) to be manipulated as described in Section 3.2. We randomly select one type of the five elements to transform. For the selected element, we generate a new value of the element stochastically within the limit of our search space and replace the existing one.

In this generator, we produce every new architecture of the population by applying the architecture transformation function to the basic one. In other word, each initial architecture of the new population is the deformed representation of the basic one. After generating offspring architectures, the evolution starts the same procedure described in Algorithm 1. Moreover the architecture transformation function is utilized to mutate models as well.

4. Experiments and Results

Our experiments mainly consist of two stages, searching for the basic model on CIFAR-10 and then transfer to ImageNet. In this section, we introduce the implementation details in EAT-NAS and the experimental results. We analyze the results of some contrast experiments which demonstrate the effectiveness of our EAT-NAS method.

4.1. Search on CIFAR-10

The experiment on CIFAR-10 could be 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. The original CIFAR-10 testing set is only utilized in the evaluation process of the final searched models. We use standard data pre-processing and augmentation steps. All images are whitened with the channel mean subtracted and the channel standard deviation divided. Then we crop 32 x 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 to measure the accuracy. We mutate about 1,400 models during the total evolution and only top-8 models are retrained on the full training dataset and evaluated on

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(%)	Type
MobileNet-v1 [8]	4.2	575	70.6 / 89.5	manual design
MobileNet-v2 [18]	3.4	300	71.7 / -	manual design
MobileNet-v2 (1.4)[19]	6.9	585	74.7 / -	manual design
ShuffleNet-v1 2x [23]	≈ 5	524	73.7 / -	manual design
NASNet-A [26]	5.3	564	74.0 / 91.6	handcraft transfer
NASNet-B [26]	5.3	488	72.8 / 91.3	handcraft transfer
NASNet-C [26]	4.9	558	72.5 / 91.0	handcraft transfer
AmoebaNet-A [16]	5.1	555	74.5 / 92.0	handcraft transfer
AmoebaNet-B [16]	5.3	555	74.0 / 91.5	handcraft transfer
AmoebaNet-C [16]	5.1	535	75.1 / 92.1	handcraft transfer
AmoebaNet-C (more filters) [16]	6.4	570	75.7 / 92.4	handcraft transfer
PNASNet-5 [11]	5.1	588	74.2 / 91.9	handcraft transfer
MnasNet [22]	4.2	317	74.0 / 91.8	search from scratch
MnasNet*	4.2	317	73.3 / 91.3	search from scratch
DARTS [1]	4.7	574	73.3 / 91.3	handcraft transfer
EATNet-A	5.1	563	73.8 / 91.7	elastic transfer
EATNet-B	5.3	551	73.7 / 91.5	elastic transfer
EATNet-S	4.6	414	72.8 / 90.9	elastic transfer

* To avoid any discrepancy between different implementations or training settings, we incorporate MnasNet [22] into our training framework. The result of MnasNet* was obtained by training under the same settings as ours. For further proofs of our effectiveness, we make ablation studies with comparisons to handcraft transfer in the following sections.

the testing dataset. The number of model parameters is the sub-optimizing objective during evolution. We set the target number of model parameters as 3.0M. Each model on CIFAR-10 consists of 7 blocks and the down sampling operations are carried out in the third and fifth block. 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 there is not any limitation. Some constraints are added to the scale of the model within an acceptable range to avoid the memory running out of control during 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 learning rate of 0.0256 (fixed during 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 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 the ones used for architecture search. Following existing works [15, 26, 11, 16], additional enhancements include cutout [6] with length of 16, and auxiliary towers with weight 0.4. We don't use the path dropout during training. The training of the searched model takes around 13 hours on two GPUs.

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 [12], we report the mean and standard deviation of 5 independent runs for our full model. The basic model achieves 96.42% mean test accuracy with only 2.04M parameters and the standard deviation of 0.05. The architecture of the basic model is shown in Fig. 3.

4.2. Transfer to ImageNet

We use the architecture of the basic model searched on CIFAR-10 as the initialization seed to generate models of the population on ImageNet. The search process is carried out on the whole ImageNet dataset. We use the offspring architecture generator to produce 64 new architectures based on the basic architecture. During architecture search, we train every model with batch 128 and learning rate 0.05. The data augmentation only keeps randomly resized cropping with scale of [0.2, 1.0] and randomly horizontal flipping for the training dataset. The number of multiply-add operations is set as the sub-optimizing objective during evolution. We set the target number of multiply-add operations

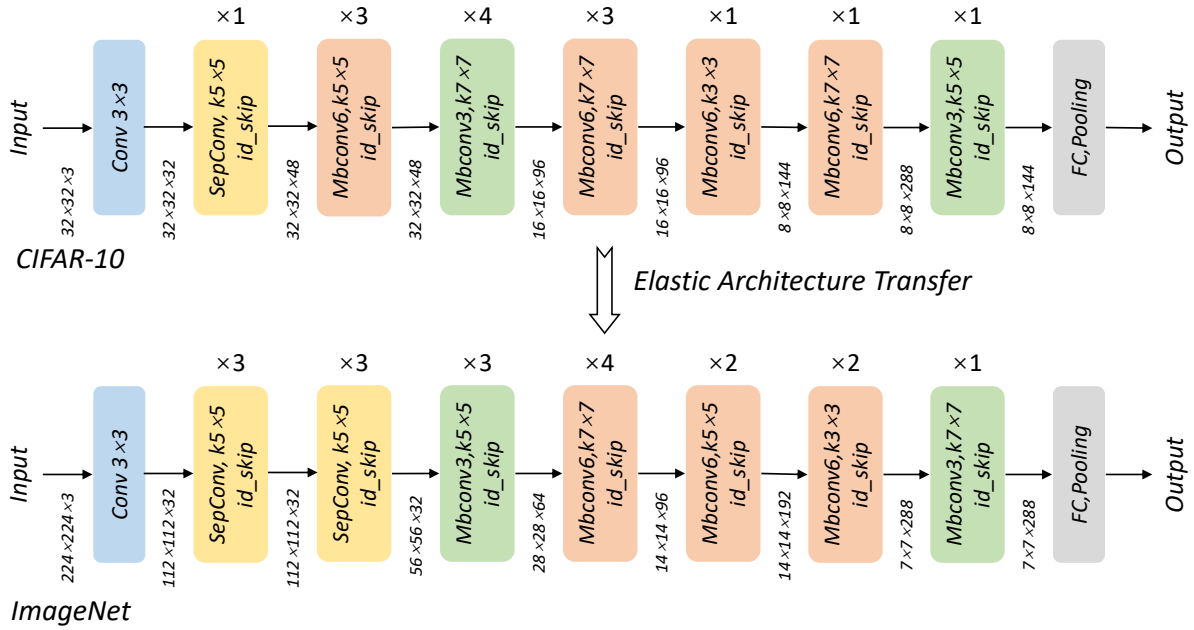


Figure 3: The architectures searched by EAT-NAS. The upper one is the basic architecture searched on CIFAR-10. And the nether one is the architecture searched on ImageNet which is transferred from the basic architecture.

as 500M. Other hyperparameters of 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 down sampling 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 expansion ratio is limited within [8, 16].

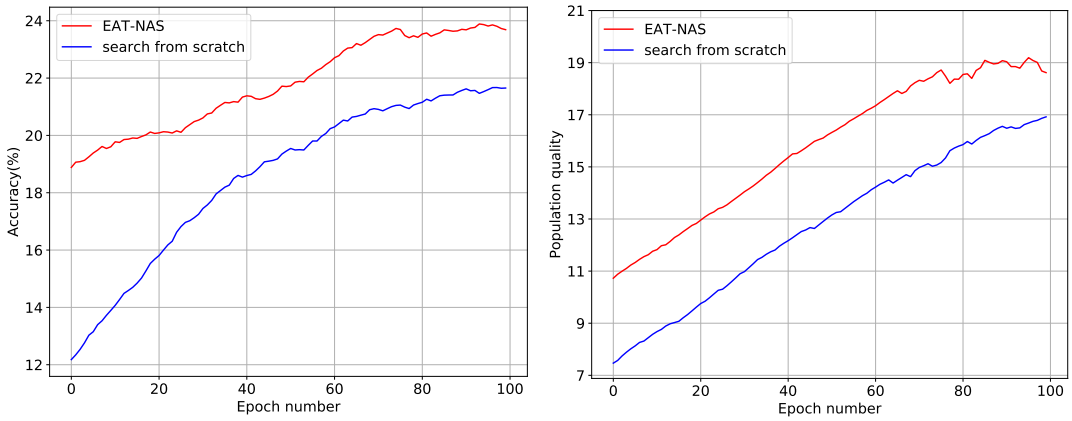
For evaluating the model performance on ImageNet, we retrain the final top-8 models on 224×224 images of the training dataset for 200 epochs, using standard SGD optimizer with momentum rate set to 0.9, weight decay 4×10^{-5}. Our batch size is 64 on each GPU and we use 4 GPUs. The initial learning rate is 0.1 and it decays in a polynomial schedule to 1×10^{-4}. For data augmentation, we resize the original input images with its shorter side randomly sampled in [256, 480] for scale augmentation [20]. A 224×224 patch is randomly cropped from an image or its horizontal flip, with the per-pixel mean subtracted [10]. The standard color augmentation in [10] is used. For the last 20 epochs, we only keep the random crop and horizontal flip preprocessings on $256 \times N$ resized images to fine-tune the model. All the other data augmentation strategies are withdrawn.

As shown in Fig. 4, the evolution process takes about 100 mutating epochs to converge. In other 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 architecture search, 50 times the amount of ours. With much less computational resources, EAT-NAS achieves comparable results on ImageNet which are illustrated in Table 1. The architecture of EATNet-A is displayed in Fig. 3.

Table 2: The results of contrast experiments on ImageNet. SS-A and SS-B denote the models searched from scratch on ImageNet. The basic model searched on CIFAR-10 is directly applied on ImageNet without any modification. The Model-B in the table denotes the best model searched on ImageNet with a bad basic architecture.

Model	#Params (M)	#Mult-Adds (M)	Top-1/Top-5 Acc(%)
SS-A	5.53	447	71.6 / 90.4
SS-B	2.88	263	69.4 / 88.7
Basic Model	3.14	886	73.9 / 91.8
Model-B	3.20	405	71.8 / 90.3
EATNet-A	5.08	563	73.8 / 91.7
EATNet-B	5.28	551	73.7 / 91.5
EATNet-S	4.63	414	72.8 / 90.9

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 the basic architecture on CIFAR-10 and 4 days on 8 GPUs to transfer to ImageNet.



(a) The mean accuracy of the models in the population.

(b) The population quailty.

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

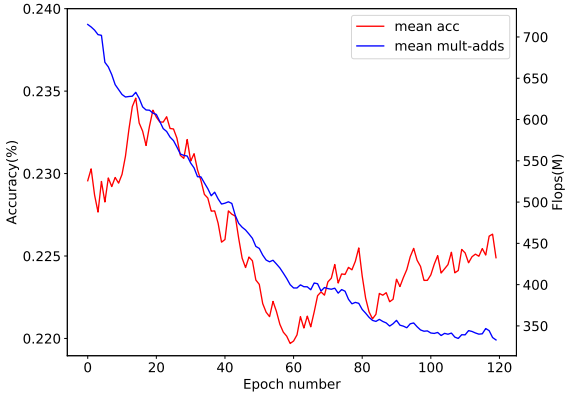


Figure 5: The mean accuracy and the mean number of multiply-add operations of the models during the search on ImageNet whose basic architecture has worse performance on CIFAR-10.

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 set the same as EAT-NAS from search to evaluation. The search process takes the same GPU days 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 of EAT-NAS. In Table 2, we compare two models of top-8 searched from scratch with that from EAT-NAS, which are both selected and retrained in the 100th search epoch. 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 to ImageNet without any modification. 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 an high validation accuracy but with a large number of multiply-add operations. Its high validation accuracy demonstrates the superior quality of the initialization seed in transfer process. Our EAT-NAS method adjusts the scale of the model to an acceptable range with comparable performance.

Impact of Basic Model Performance We select one worse model as the basic model 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 model with bad performance will have a negative impact on transfer. It takes extra search epochs to optimize the size of the model at the expense of the accuracy. This experiment demonstrates the importance of the search process for a well-performing basic model. We retrain the searched top-8 models under the same settings as well. We select the best model to compare with one searched by EAT-NAS which has a similar model size. As Table 2 displays, models searched by EAT-NAS surpasses those searched on the bad basic. We attribute the results to the performance of the basic model.

5. Conclusion and Future Work

In this paper, we propose an elastic architecture transfer mechanism for accelerating large-scale neural architecture search (EAT-NAS). Rather than spending a lot of computational resources to directly search the neural architecture 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, which are comparable to the very computational expensive method MnasNet [22].

In future, we would try to combine the proposed mechanism with other search methods, such as reinforcement learning based NAS and gradient based NAS. In addition, EAT-NAS can also be utilized to search for neural architectures in other computer version tasks like detection, segmentation and tracking, which we also leave for future work.

Acknowledgement

We thank Liangchen Song and Guoli Wang for the discussion and assistance.

References

[1] Anonymous. Darts: Differentiable architecture search. In *Submitted to International Conference on Learning Representations*, 2019. under review. 2, 6

[2] T. Chen, I. J. Goodfellow, and J. Shlens. Net2net: Accelerating learning via knowledge transfer. *CoRR*, abs/1511.05641, 2015. 4

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

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

[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. 6

[7] D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In *FGA*, pages 69–93. 1990. 2

[8] 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. 6

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

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012. 7

[11] 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, 2, 4, 6

[12] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. *CoRR*, abs/1711.00436, 2017. 3, 6

[13] 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. 2

[14] 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. 2

[15] 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, 2, 3, 4, 6

[16] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. *CoRR*, abs/1802.01548, 2018. 1, 2, 3, 4, 6

[17] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *ICML*, pages 2902–2911, 2017. 1

[18] 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. 3, 6

[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. *CoRR*, abs/1801.04381, 2018. 6

[20] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 7

[21] 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

[22] 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. 3, 4, 6, 7, 9

[23] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017. 6

[24] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu. Practical block-wise neural network architecture generation. In *CVPR*, 2018. 2

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

[26] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017. 1, 2, 3, 4, 6