Boosting Order-Preserving and Transferability for Neural Architecture Search: a Joint Architecture Refined Search and Fine-tuning Approach

Beichen Zhang, Xiaoxing Wang, Xiaohan Qin, Junchi Yan[*]
Department of Computer Science and Engineering & MoE Key Lab of AI, Shanghai Jiao Tong University
{zhangbeichen,figure1_wxx,galaxy-1,yanjunchi}@sjtu.edu.cn
https://github.com/beichenzbc/Supernet-shifting

Abstract

Supernet is a core component in many recent Neural Architecture Search (NAS) methods. It not only helps embody the search space but also provides a (relative) estimation of the final performance of candidate architectures. Thus, it is critical that the top architectures ranked by a supernet should be consistent with those ranked by true performance, which is known as the order-preserving ability. In this work, we analyze the order-preserving ability on the whole search space (global) and a sub-space of top architectures (local), and empirically show that the local order-preserving for current two-stage NAS methods still need to be improved. To rectify this, we propose a novel concept of ***Supernet Shifting****, a refined search strategy combining architecture searching with supernet fine-tuning. Specifically, apart from evaluating, the training loss is also accumulated in searching and the supernet is updated every iteration. Since superior architectures are sampled more frequently in evolutionary searching, the supernet is encouraged to focus on top architectures, thus improving local order-preserving. Besides, a pre-trained supernet is often un-reusable for one-shot methods. We show that Supernet Shifting can fulfill transferring supernet to a new dataset. Specifically, the last classifier layer will be unset and trained through evolutionary searching. Comprehensive experiments show that our method has better order-preserving ability and can find a dominating architecture. Moreover, the pre-trained supernet can be easily transferred into a new dataset with no loss of performance.*

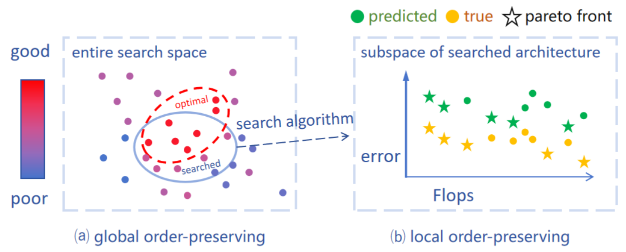

Figure 1. An illustration of global and local order-preserving ability. For global one, we care about coarse-grained comparison to wipe out poor architectures in entire search space. For local one, we care about fine-grained comparison to rank the architectures in a subspace of top architectures.

1. Introduction

Apart from traditional manually designed neural networks e.g. VGG [27] and RESNET [13], Neural Architecture Search (NAS), as an important part of Automated Machine Learning (AutoML), aims to automatically search an optimal architecture in a certain search space.

Early NAS approaches [2, 19, 39, 40] adopt a time-consuming pipeline by sampling architectures and training their weights separately. To speed up the search procedure, many recent approaches introduce performance estimators for each architecture. Though there exists a performance estimation gap, the search stage focuses more on the relative ranking of architectures rather than true performance.

Some works [18, 23, 30] propose zero-cost proxies, which only require one forward step or one backward step. Nevertheless, it cannot perform consistently well on diverse tasks [1]. Other works [3, 6, 24] build up a supernet that contains all candidate operations and connections in the search space. Once a supernet is trained, all architectures can be quickly evaluated by loading the supernet weight. Since such a supernet-based performance estimator is widely used, the order-preserving ability, i.e. whether the

[*]Correspondence author. The work was partly supported by NSFC (92370201, 62222607).

estimated architecture ranking is consistent with true performance ranking, is an important index. In this work, we delve into two aspects of order-preserving ability. One is **global order-preserving ability** to distinguish good architectures from poor ones in the whole search space. The other is **local order-preserving ability** to rank top architectures that have good performance. We argue that both the above abilities are important. Poor global order-preserving abilities will lead to unsatisfactory results since it cannot screen out poor architectures. Poor local order-preserving ability will reduce search efficiency.

Some NAS approaches, like DARTS [20] and GDAS [9], introduce an architecture parameter optimized during supernet training. It lets supernet focus on part of the architectures, improving local resolution. However, it can't ensure global order-preserving ability. Some previous works [11, 15, 35] have pointed out that an early bias is easily introduced. The search direction may be led wrongly in the early stage.

Other works like SPOS [11] and FairNAS [6] ensure global order-preserving ability. They treat NAS as a two-stage process. First, a single-path supernet is trained by uniform sampling and all architectures' weights are optimized equally. Then, a searching algorithm like evolutionary algorithm (**EA**) is applied. This method ensures fairness in the training stage so global order-preserving ability is better. However, the local order-preserving ability still needs to be improved. We select the top-10 architectures during searching and retrain them separately. The *Kendall's tau* of the accuracy in supernet and after retrain is only 0.17, which is far from satisfactory. While although the choice blocks are similar, there do exist a non-negligible accuracy gap of 0.8% after retrain. The problem is that weights of different architectures are not fully decoupled due to weight-sharing strategy. Some works [12, 16, 21, 29, 33] point out that the phenomenon of multi-model forgetting exists in weight sharing. Uniform sampling strategy avoids early bias and ensures fairness, but may not be precise in local comparison. In fact, the supernet is encouraged to focus on some local superior architectures. Inferior architectures may produce noise and should be neglected.

However, ensuring both isn't straightforward. Uniform sampling isn't precise enough. Biased sampling is needed to focus on some superior architectures, but it can lead to improper bias and hurt global order-preserving ability.

To this end, we propose a refined search strategy combining architecture searching with supernet fine-tuning to both achieve high global and local order-preserving ability. To ensure global order-preserving, we first train a supernet by uniform sampling to avoid early bias and ensure fairness. To improve local order-preserving, we add a **Supernet Shifting** stage during searching. Specifically, we calculate and accumulate the training loss together with evaluation when an architecture is sampled. After each iteration of evolutionary searching, the supernet is updated. Since superior architectures are sampled more frequently in evolutionary searching, our shifted supernet is encouraged to focus on top architectures, thus having better local discernment and better local order-preserving ability. While for inferior architectures, the supernet gradually forgets them. Unlike superior architectures, we don't care about the fine-grained estimation since they should all be eliminated in searching.

Moreover, our method has better transferability compared to other one-shot NAS methods that have to train a new supernet for a new dataset. In contrast, our method can adopt the original supernet and fine-tune the weight during the search procedure. Plenty of previous work on transfer learning and pre-trained models [8, 14] proves that weights can be inherited and only need some small changes for different tasks. This provides a theoretical basis to transfer supernet to different datasets.

Overall, our contributions can be summarized as follows:

1) Comprehensive analysis about order-preserving ability for NAS methods. Many NAS methods adopt proxies to estimate performance to speed up the search process, making order-preserving ability a general metric. In this work, we further define global and local order-preserving ability and verify the dilemma of current NAS method.

2) A stable strategy improving both global and local order-preserving ability. This work introduces supernet shifting, a simple but effective method to improve global and local order-preserving ability. Supernet is self-adaptively revised during searching. Compared with previous method, our method is by design more in line with the essence of NAS in the sense of paying adaptive attention to different architectures, and meanwhile introduces less bias.

3) A flexible and efficient strategy realizing transferring during searching. Our method neatly realizes the transfer of supernet by reusing the feature encoder part of pre-trained supernet. This allows flexible design of decoders to better fit different tasks. Transferring with searching further ensures efficiency. To the best of our knowledge, this is the first time an entire supernet can be transferred.

4) Strong Performance. Experiments in Sec. 4 show the general effectiveness of our method. It improves both global and local order-preserving ability and can obtain dominating architectures. Flops can be reduced by $5M$ and the accuracy increases by 0.3% on ImageNet-1K. For transferability, it can accelerate searching process for ten times compared with SPOS without performance loss.

2. Related Works

One-shot NAS. Early NAS approaches [2, 19, 39, 40] train different architectures separately. The cost of time is often unaffordable. *ENAS* [24] introduces weight-sharing strategy so that different architectures can be jointly optimized. This greatly speeds up the NAS process. Based on weight-sharing strategy, *One-Shot NAS* [3] further adopts path dropout technique to train a supernet. Once a supernet is trained, the architectures can simply inherit weight from it. Most follow-up works adopt this strategy. They can be roughly divided into two classes based on their search space. Some works [4, 5, 9, 10, 20] adopt contiguous search space. They introduce an architecture parameter and use gradient decent to optimize the supernet weight and the architecture parameter. Another line of works [6, 11, 28] adopt discrete search space. They treat NAS as a two-stage problem. In supernet training stage, only one path is sampled and optimized from search space. Then searching algorithms are applied to find the optimal architecture. This type of method is usually more stable and is easier to optimize. A widely used NAS method *SPOS* [11] falls into the second category. *SPOS* trains a supernet by uniform sampling and applies evolutionary searching algorithm to search for optimal architecture. This is our main baseline.

Evolutionary-based NAS. Evolutionary algorithm is widely used in discrete optimization problem. It simulates the process of biological evolution process. New candidates are created by crossover and mutation. *Large-Scale Evolution of Image Classifiers* [25] first introduces evolutionary algorithm into NAS problem. New candidate architectures are produced and trained separately. After the proposal of weight sharing strategy and supernet, evolutionary algorithm is widely used in the works [6, 11] using discrete search space as a sample strategy during searching stage. Specifically, *SPOS* [11] applies evolutionary searching algorithm first on a single-path supernet while *FairNAS* [6] uses NSGA-II. It's widely accepted that evolutionary searching algorithm is more efficient than random searching since it takes use of the current evaluation results.

Transferability in NAS. Transfer learning aims to find an effective way to transfer the knowledge learned from a source domain to a target domain. Usually, the weights of the pre-trained model are carefully fine-tuned and the structure is often adjusted to adapt it to the target domain. Transferability is an important feature for NAS method, since it can significantly reduce the time cost and hardware restriction for NAS application. However, few NAS methods find an effective way to realize tranferability. Some NAS methods [20, 32] first search on a small-scale dataset like Cifar10 [17], then extend the searched architecture to large-scale datasets like ImageNet-1K [7]. This transfer mode could be unreliable because of the diverse data distribution. Some works [26, 34] introduce meta learning to achieve transferability. However, this is often complicated. The model and hyperparameters have to be carefully designed and they usually lack explainability. To the best of our knowledge, no previous work has transferred an entire pre-trained supernet to a new dataset.

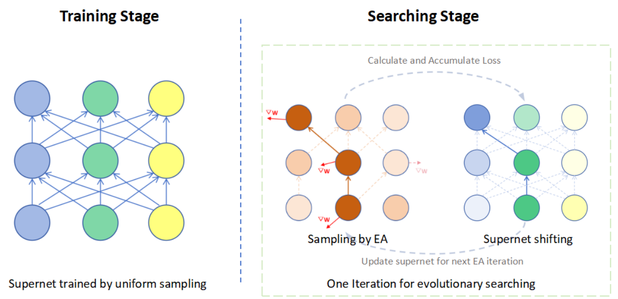
Training Stage
Searching Stage
Calculate and Accumulate Loss
∇w
Sampling by EA
Supernet shifting
Update supernet for next EA iteration
Supernet trained by uniform sampling
One Iteration for evolutionary searching

Figure 2. Pipeline of our method with two stages. In the training stage, a single-path supernet is trained by uniform sampling. Each architecture is equally treated. In the searching stage, evolutionary searching is applied. When an architecture is sampled, the training loss is calculated and accumulated apart from evaluating. At the end of each iteration, the supernet is updated. Since superior architectures are sampled more frequently in evolutionary searching, the supernet is expected to shift to focus on top architectures.

Table 1. Overview of different one-shot NAS methods. By adding supernet shifting, our method introduces extra attention on partial architectures based on their validation performance. Therefore, the local resolution can be improved and less bias is introduced. Besides, ours has the best transferability.

Method	Supernet training	Search	Attention to different architectures	global resol.	local resol.	transferable supernet
DARTS [20]	gradient descent, path dropout	/	biased by architecture parameter at start	✗	✓	✗
OneShot [3]	path dropout	random search	equal attention	✓	✗	✗
SPOS [11]	uniform sampling	EA	equal attention	✓	✗	✗
FairNAS [6]	strict fairness sampling	EA	strictly equal attention	✓	✗	✗
Ours	uniform sampling	EA with shifting	biased by validation after uniform training	✓	✓	✓

3. Method

3.1. NAS Retrospection from Order-preserving

To have a deeper understanding of different NAS methods and their advantages as well as shortcomings, we use some mathematical expressions to show their training pipelines and training goals. Thus, we need to make some definitions for the following mathematical expressions.

We denote $\mathcal{N}$ as the supernet, θ as the learned architecture variable, α represents different architectures, $\tau(\alpha)$ represents a sampling distribution of architecture α. $\mathcal{A}$ represents the entire search space and $u(\mathcal{A})$ represents a uniform sampling over the search space. $\mathcal{W}_{\mathcal{A}}$ represents the weight of the entire supernet and $\mathcal{W}_{\mathcal{A}}^{\alpha}$ represents the weight of architecture α inherited from supernet $\mathcal{W}_{\mathcal{A}}$.

As discussed above, we divide the previous one-shot NAS into two main categories. Methods in the first category [4, 5, 9, 10, 20] use continuous search space and introduce an architecture variable θ which is jointly optimized with supernet weights. Their optimization steps are described as follows:

$$(\theta, \mathcal{W}_\theta) = \underset{\theta, \mathcal{W}}{\arg\min}\, \mathcal{L}_{train}(\mathcal{N}(\mathcal{A}(\theta), \mathcal{W})) \tag{1}$$

The existence of θ leads the supernet to focus on a local set of architectures, and this improves the local resolution near θ. This kind of method theoretically works well if θ is optimal. However, as some works [11, 15, 35] points out, θ is jointly optimized and this may introduce early bias. So the supernet may be misled and trapped into local optimal architecture, which hurts global order-preserving ability.

Approaches in the other category [6, 11, 28] use discrete search space and get rid of the learnable architecture variable θ. They first train a supernet by uniform sampling and apply searching algorithms to find the optimal architecture based on their performance on the supernet. Their optimization steps are sequential, which can be illustrated as:

$$\mathcal{W}_\mathcal{A} = \underset{\mathcal{W}}{\arg\min}\, \mathbb{E}_{\alpha \sim u(\mathcal{A})}[\mathcal{L}_{train}(\mathcal{N}(\alpha, \mathcal{W}))] \tag{2}$$

$$\alpha^* = \underset{\alpha}{\arg\max}\, ACC_{val}(\mathcal{N}(\alpha, \mathcal{W}_\mathcal{A}^\alpha)) \tag{3}$$

By removing θ and applying uniform sampling, it better achieves the fairness of comparison and the global order-preserving ability. However, it isn't precise enough for local comparison because it pays equal attention to every single architecture. Our work aims to achieve both global and local order-preserving ability. Sec. 3.2 aims to ensure the former and Sec. 3.3 aims to improve the latter.

3.2. Single-Path Supernet Training

First, we construct a single-path supernet. The construction and training of supernet follow SPOS [11]. Specifically, the supernet has a series of choice blocks, and each choice block further could be specified by certain configurations. Only one choice is invoked at the same time. The supernet is trained by uniform sampling. Every single architecture is treated equally. The training process can be defined as:

$$\mathcal{W}_\mathcal{A} = \underset{\mathcal{W}}{\arg\min}\, \mathbb{E}_{\alpha \sim u(\mathcal{A})}[\mathcal{L}_{train}(\mathcal{N}(\alpha, \mathcal{W}))] \tag{4}$$

As will be shown in the experiments in Sec. 4.3, global order-preserving ability for the supernet trained by uniform sampling is quite ideal, so that most poor architectures can be effectively eliminated. However, the local order-preserving ability is still far from satisfactory.

3.3. Supernet Shifting

After training a supernet by uniform sampling, the variance of different architectures' accuracy is relatively small. This is enough for global comparison. The supernet can distinguish between good architecture and poor architecture. However, it's not capable enough to distinguish the best architecture from several similar architectures, and this seriously affects its performance.

To solve the problem, the supernet weights should be shifted based on the performance of different architectures evaluated on the current supernet. Specifically, the architectures which perform better should be more emphasized and sampled more frequently. It can be depicted as:

$$\mathcal{W}_{\mathcal{A}^*} = \underset{\mathcal{W}}{\arg\min}\, \mathbb{E}_{\alpha \sim \tau(\alpha)}[\mathcal{L}_{train}(\mathcal{N}(\alpha, \mathcal{W}_\mathcal{A}^\alpha)] \tag{5}$$

The prior distribution $\tau(\alpha)$ is important. If $\tau(\alpha)$ is a uniform, the equation is the same as Eq. 4. As discussed above, for architecture α, we want the sampling probability $\tau(\alpha)$ and the performance of α to be positively correlated:

$$r[\tau(\alpha), ACC_{val}(\mathcal{N}(\alpha, \mathcal{W}_\mathcal{A}^\alpha))] \ > \ 0 \tag{6}$$

Thus, after the shifting stage, the supernet is led to focus on better architectures. We then search for the optimal architecture based on the shifted supernet.

$$\alpha^* = \underset{\alpha}{\arg\max}\, ACC_{val}(\mathcal{N}(\alpha, \mathcal{W}_{\mathcal{A}^*}^\alpha)) \tag{7}$$

Compared with those biased architecture sampling strategies using architecture parameter θ [5, 9, 20], this sampling strategy could be more precise and more reliable as fairness can be ensured in the supernet training stage and the sampling distribution is determined by their evaluation performance directly.

Till now, there is problem remaining unsolved. A proper sampling strategy satisfying Eq. 6 isn't straightforward.

We take use of the property of evolutionary searching algorithm to solve the problem. The main difference between evolutionary searching algorithm and random searching is that, for different architectures, the probability of being sampled by evolutionary searching algorithm is different. The evolutionary searcher tends to sample the architectures which are similar to the current top architectures, so better architectures are sampled more frequently. Therefore, we can claim that the sampling probability of evolutionary searching algorithm satisfies Eq. 6

Thus, we implement Eq. 5 and Eq. 7 homogeneously and realize supernet shifting together with evolutionary searching. Specifically, when an architecture is sampled in evolutionary searching stage, apart from evaluating the validation performance, the loss is calculated according to tens of training iterations and is accumulated. After one iteration of evolutionary searching, the supernet is updated. Thus, it will be shifted to focus on a local set of good architectures due to the biased sampling distribution. This improves the precision for local comparison.

Fig. 3 shows the supernet shifting process during the evolutionary searching stage. The accuracy increases for superior architectures while remains or even decreases for inferior ones. This verifies our assumption.

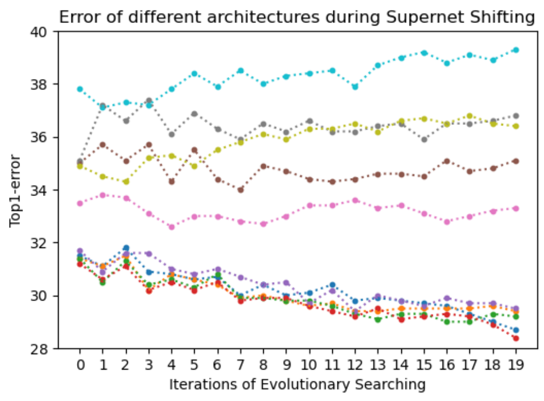

Figure 3. Trajectory of Supernet Shifting process. We sample 5 searched superior architectures (bottom) and 5 random architectures (top). We monitor their error rate over iterations of evolutionary searching. Iteration 0 denotes the original supernet trained by uniform sampling. The shifting supernet gradually focuses on superior architectures and dismisses inferior ones.

Since supernet keeps shifting in searching, we have to eliminate the bias produced by the changing supernet. Therefore, repeatedly sampling should be allowed. Specifically, for a single particular architecture, we allow a new sample in each iteration of evolutionary algorithm and we will update its latest accuracy. After each iteration, we keep the top-50 different architectures for later mutation and crossover process.

As will be empirically shown in the experiment in Sec. 4.2 and Sec. 4.3, adding the additional shifting stage significantly improves the local order-preserving ability on the superior architectures searched by evolutionary searching algorithm and the global order-preserving ability doesn't decay. One can obtain dominating architectures on different datasets by applying supernet shifting. This verifies the general effectiveness of our method.

Since we implement supernet shifting and architecture searching homogeneously, and the shifting process need few iterations for training comparing with evaluation, the additional time cost is quite little. Take ImageNet-1K as an example, the total time for evolutionary searching stage only rises from 17 GPU hours to 19 GPU hours.

Alg. 1 shows the entire searching stage of our method. It can be applied in any two-stage NAS method, after a supernet is trained to improve the local resolution of the supernet.

3.4. Supernet Transferring

The lack of transferring ability is a common problem for previous NAS methods. For a new dataset, most of the previous methods need to train a new supernet before searching stage. This is quite time-consuming, which may need several GPU days.

The success of transfer learning and pre-trained models has shown the inner-correlation of different downstream tasks and different datasets. Our method is inspired by fine-tuning, which is one of the most straight-forward and effective implementations of transfer learning.

Algorithm 1 Supernet shifting during EA searching

```
Input: Pre-trained Supernet $\mathcal{N}$ with weights $\mathcal{W}$, Size of a population $T$.
Output: Pareto optimal architectures.
1: Initialize a population of architectures $Top\_T$ and get the validation accuracy
2: repeat
3:     Generate a new population of architectures by EA $\{\mathcal{A}_t\}_{t=1}^{T}$ = $Generate\_new\_candidates(Top\_T, T)$
4:     Initialize total gradient $\nabla\mathcal{W} = 0$
5:     for $t = 0$ to $T$ do
6:         Get validation accuracy for architecture $\mathcal{A}_t$ $ACC_{\mathcal{A}_t} = validate(\mathcal{N}, \mathcal{W}, \mathcal{A}_t, val\_data)$
7:         compute and accumulate the gradient of supernet $\nabla\mathcal{W}_{\mathcal{A}_t} = get\_gradient(\mathcal{N}, \mathcal{W}, \mathcal{A}_t, train\_data)$, $\nabla W += \nabla\mathcal{W}_{\mathcal{A}_t}$
8:     end for
9:     Update the supernet weights $\mathcal{W} \leftarrow \nabla W$
10:    Update current $Top\_T$ for next iteration $Top\_T$ = $Update\_Top\_T(Top\_T, \{\mathcal{A}_t\}_{t=1}^{T}, \{ACC_{\mathcal{A}_t}\}_{t=1}^{T})$
11: until The end of search stage
```

With a little modification, our supernet shifting stage can be used to transfer a pre-trained supernet to a new dataset. Specifically, we keep the feature-extraction part of the supernet and only set a new fully-connected layer for prediction. The supernet is fine-tuned for tens of iterations when an architecture is sampled in the evolutionary searching algorithm. The loss is updated immediately instead of accumulating for faster transfer. Since the feature extraction part has a strong internal correlation on different datasets, only prediction layer needs to train from scratch. So the supernet is quickly shifted to a new dataset after several architectures are sampled. The searching algorithm can work properly.

We also try a different mode, freezing the feature extraction part and only fine-tuning the final prediction layer. In experiment, fine-tuning the whole supernet works better, especially on CIFAR-100 dataset. This is probably because ImageNet-1K dataset isn't large-scale enough. Thus, the general knowledge learned can't overcome the gap of data distribution. Moreover, since we adopt a lightweight search space, fine-tuning the whole supernet isn't time-consuming. If a more complicated supernet is pre-trained on a larger-scale dataset, the second mode may be better.

Note that the size of images may vary in different datasets, we try two different solutions. The first solution is to simply resize the image into a fixed size like 224×224, while the second solution is to set an up-sampling or down-

Table 2. Overview of different methods on improving relative performance estimation in weight-sharing strategy. Some works [38] reduce the shared-weight. Some works [12, 37] directly reduce gradient conflict. Some other works [31, 36] including ours adopt a nonuniform sampling strategy to focus on superior architectures. Comparing with other methods, ours consumes little extra time, storage and introduce less early bias.

Method	Strategy	Extra storage	Extra time cost
Few-shot NAS[38]	**Reduce shared weight:** split supernet into sub-supernets	sub-supernet	None
SUMNAS[12]	**Reduce gradient conflict:** Reducing gradient direction conflict in training	None	Compute reptile gradient in training
AttentiveNAS[31]	**Nonuniform sample:** Focus on Pareto-best and worst in training	None	Pre-train evaluator + evaluation in training
GreedyNAS[36]	**Nonuniform sample:** Multi-path sampling with rejection in training	None	Extra evaluation in training
Ours	**Nonuniform sample:** Shift supernet to focus on superior architectures in searching	None	Loss accumulate in searching(in total 2 GPU hours)

sampling layer in the network. This layer is also initialized and trained during searching together with the prediction layer. By experiment, the former method works better.

In the experiment, we find the transferring supernet converges quickly after sampling several architecture. And after 4 iterations of evolutionary searching, the transferring supernet achieves nearly the same accuracy as a new supernet which is trained from scratch for 80,000 iterations on the new dataset. This strongly confirms that the supernet is theoretically transferable.

Note that our transferring method is time-saving with no loss of performance. Since no new supernet is trained, we make the searching process 10 times faster and even get a dominating architecture comparing with training a new supernet on ImageNet-100 dataset. The detailed experiments are shown in Sec. 4.4

3.5. Approach Summary and Remarks

We propose a refined searching method for NAS. By introducing supernet shifting, the supernet gradually focus on superior architectures and both global and local order-preserving ability can be ensured. Moreover, a pre-trained supernet can be transferred into other tasks effectively.

Table 2 shows the comparison of different method trying to overcome multi-model forgetting and improve performance estimation in weight-sharing. Ours consumes little extra time and storage. Encouraging supernet to focus on top architectures in searching instead of training further introduces less early bias, and supports transferability.

Our method also inherits the flexibility and simplicity of two-stage NAS. Multiple constraints can be added in the evolutionary searching stage to restrict the maximum flops, parameters and latency.

4. Experiment

4.1. Experiment Setting

Dataset. To show the general effect of our method, three different datasets are used. The biggest dataset is *ImageNet-1K* [7], which contains 1000 different classes, over one million images for training and 50,000 images for validation. *ImageNet-1K* is the most important dataset in our experiment. Since getting retrain performance of a large number of architectures on *ImageNet-1K* is time-consuming, we also use *ImageNet-100* dataset. *ImageNet-100* is a dataset sampled from *ImageNet-1K*. It contains 100 classes and each class contains 1300 images for training and 50 images for validation. *ImageNet-100* dataset is mainly used to evaluate the order-preserving ability and the performance of supernet transferring. Besides, we also use *Cifar-100* [17] in supplementary experiments.

Search Space. Our search space also follows SPOS. It is based on ShuffleNet-V2 [22], which is a powerful lightweight network. There are total 20 searching blocks and 4 choices for each block. The total search space is 4^{20}. We use $FLOPS \leq 330M$ as complexity constraint in evolutionary searching as well.

Training. For the training of supernet and the retraining of the searched architectures, we use the same setting (including hyper-parameters, data-augmentation strategy, learning-rate decay, etc.) as SPOS. The batchsize is 1024, the supernet is trained for 150,000 iterations and the searched architecture is trained for 300,000 iterations on ImageNet-1K. For ImageNet-100, the batchsize is 256, the supernet is trained for 80,000 iterations and the searched architecture is trained for 120,000 iterations. To further ensure fairness, we use the same supernet weight for different searching algorithms if the search space is the same. Training uses 4 *NVIDIA GeForce RTX 3090* GPUs and searching uses 1 *NVIDIA GeForce RTX 3090* GPU.

4.2. Searching Result

For comparison, we implement multiple searching algorithms and retrain the searched architecture. It involves:

1) Randomly select five architectures and choose the best architecture according to retrain performance

2) Oneshot(Train a supernet by uniform sampling and use random search to select the best architecture from 100 candidates according to supernet evaluation)

3) SPOS (train a supernet by uniform sampling and apply evolutionary searching algorithm)

4) FairNAS (train a supernet with strict-fairness sampling and apply evolutionary searching algorithm)

Table 3 shows the retrain result on different datasets of different NAS methods. We get a dominating architecture which has the lowest flops and the highest accuracy.

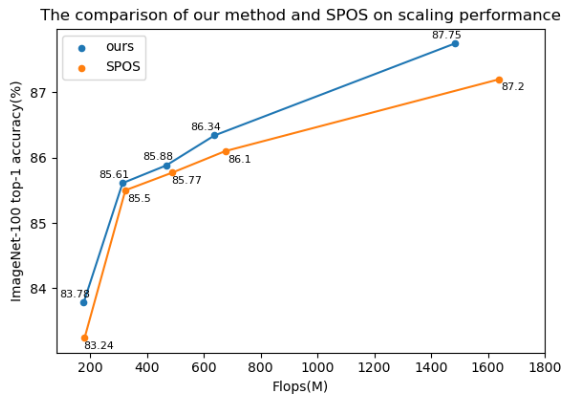

Figure 4. We choose 5 depth multipliers: 0.5, 1.0, 1.5, 2.0 and 4.0. For each we train a new supernet to which we apply our method and the SPOS method. Then, we retrain the searched architecture separately and compare the results on ImageNet-100.

Table 3. The retrain result on different datasets. For fairness, we implement different strategies ourselves in the same search space based on ShuffleNet-V2.

	ImageNet-1K			ImageNet-100	
Method	**Flops**	**Top-1 acc**	**Top-5 acc**	**Flops**	**Top-1 acc**
Random select	324M	73.29	91.01	312M	85.42
Oneshot [3]	326M	73.52	91.44	304M	85.41
SPOS (block) [11]	323M	74.01	92.25	304M	85.50
FairNAS [6]	326M	74.03	92.31	300M	85.44
Ours	**318M**	**74.28**	**92.92**	**299M**	**85.61**

To further verify the scaling performance, we use different depth multipliers to scale up the search space. By comparing with SPOS in Fig. 4, our method outperforms SPOS under every depth multiplier.

Moreover, since most previous methods [6, 31, 36] all make improvements in supernet training stage, our method can improve the searching quality by adding supernet shifting in evolutionary searching stage in a plug-and-play manner. The result is shown in Tab. 4

4.3. Order-preserving Ability

As discussed before, for one-shot NAS problem using supernet as relative performance predictor, the most important feature of supernet is the order-preserving ability between supernet performance and retrain performance. Since it's time-consuming to get the retrain accuracy of a large number of architectures, we use ImageNet-100 in this part.

We analyze the global order-preserving ability and the local order-preserving ability separately. Specifically, We choose 10 good architectures and 20 random architectures as poor architectures. Each architecture is retrained from scratch for 50,000 iterations. We have verified that all 10 good architectures are better than the 20 poor architectures after retraining. Global order-preserving ability means whether the model is able to distinguish good architectures and poor architectures, so we counted how many of the top-10 architectures evaluated by supernet in the overall 30 architectures are the ten good architectures. And local order-preserving ability refers to whether the supernet can predict the relative performance of similar good architectures. So we introduce Kendall's tau coefficient, an index indicating the positive correlation of two ranks, as our metric. We calculate Kendall's tau coefficient of the 10 good architectures to evaluate the prediction of local relative performance.

Table 4. Retrain result on ImageNet-1K in different search spaces w/ or w/o supernet shifting for different baseline methods(implemented by ourselves).

Method	**w/o shifting**		**w/ shifting**	
	Flops (M)	**Top-1 acc**	**Flops** (M)	**Top-1 acc**
ShuffleNet-V2 **(main search space)**				
SPOS	323	74.01	318 (↓ 1.5%)	74.28 (↑ 0.35%)
FairNAS	326	74.03	321 (↓ 1.5%)	74.36 (↑ 0.45%)
GreedyNAS	329	74.17	325 (↓ 1.2%)	74.17 (→)
AttentiveNAS	319	74.22	324 (↑ 1.6%)	74.38 (↑ 0.22%)
MobileNet-V2 **(supplementary search space)**				
SPOS	333	73.42	329 (↓ 1.2%)	74.01 (↑ 0.80%)
FairNAS	329	73.39	331 (↑ 0.6%)	74.13 (↑ 1.01%)
GreedyNAS	336	73.59	332 (↓ 1.2%)	73.82 (↑ 0.31%)
AttentiveNAS	335	74.12	332(↓ 0.9%)	74.23(↑ 0.15%)

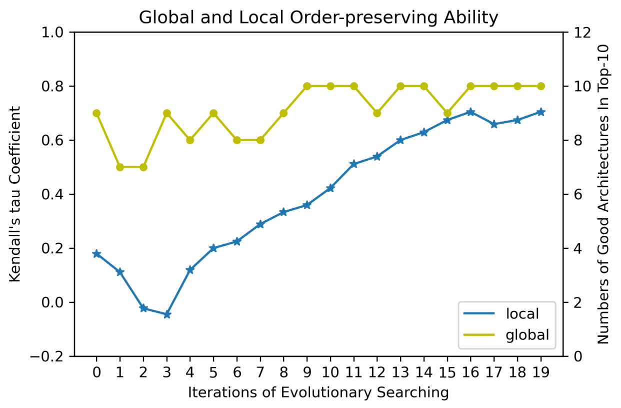

Figure 5. Experiments on order-preserving ability. The number of good architectures predicted correctly as the top-10 architectures indicates the global ranking. The Kendall's tau coefficient of the 10 good architectures indicates the local consistency.

Fig. 5 shows the result of our experiment. We can see that for the supernet trained by uniform sampling strategy, the global ranking is quite ideal but the local consistency still needs to be improved. At the beginning of the searching stage, both the global ranking and the local consistency decreases. This is probably because the searching experience is not enough for evolutionary searching algorithm and thus

Table 5. The retrain result on different transferring methods on ImageNet-100 and Cifar-100. Time cost consists of supernet training and searching and is measured in GPU hours.

	ImageNet-100			CIFAR-100		
Method	**Flops**	**Top-1 acc**	**Time**	**Flops**	**Top-1 acc**	**Time**
same architecture	307M	85.50	/	233M	74.73	/
supernet	304M	85.50	48	228M	75.40	20
supernet + shifting	**299M**	85.61	50	230M	**75.62**	21
Ours	299M	**85.83**	6	**226M**	75.38	3

Table 6. Time cost on ImageNet-1K in GPU hours and the top-1 accuracy is estimated under the same Flops. The total search time is divided into supernet training time and evolutionary search time.

Method	**Training**	**Search**	**Top-1 acc**
SPOS [11]	150	17	73.91
Ours	150	20	74.11
SPOS(short training) [11]	100	17	73.52
Ours(short training)	100	20	73.96

the sampling doesn't satisfy Eq. 6. After several iterations of evolutionary algorithm, we can see a contiguous improving of local consistency and global ranking. This verifies that by using a reliable biased sample strategy like ours, the local order-preserving ability can be improved. After 15 iterations, the local Kendall's tau rate achieves a high score and remains nearly unchanged, which indicates the convergence, and ensures the fairness of final comparison.

4.4. Supernet Transfer

To evaluate the transferability of our method, we first pretrain a supernet on ImageNet-1K and use our method to transfer the pre-trained supernet to downstream datasets including ImageNet-100 and Cifar100 during supernet shifting to search for the optimal architecture. We set three different method for comparison.

1. **ImageNet-1K $\rightarrow$ downstream:** Choose the architecture searched on ImageNet-1K directly.

2. **downstream only - SPOS:** Train a new supernet and search by evolutionary algorithm on downstream dataset.

3. **downstream only - Ours:** Train a new supernet and search by evolutionary algorithm with supernet shifting on downstream dataset.

For these three different methods, the input size is all resized to 224×224 to ensure fairness.

Table 5 demonstrates the results. Our transferring method can speed up the search process for about 10 times by reusing the encoder of supernet. Meanwhile, the search quality does not decrease. Instead, we even find dominating architecture in ImageNet-100. This shows the supernet as pre-trained on large-scale datasets like ImageNet-1K contains general knowledge and can improve the performance on downstream datasets.

We find our transferring method usually prefers architectures with lower flops and fewer parameters. This is probably because simpler architectures usually converge faster and thus they can occupy an advantageous position at the start of evolutionary searching.

4.5. Time Cost Analysis

As our method involves the supernet shifting together with evolutionary search, the additional time cost is rather small. Moreover, since supernet is fine-tuned in shifting, the requirements of the quality of pre-trained supernet is lower. We only need a rough global prediction on different architectures, and the local resolution can be improved during supernet shifting. Thus, the supernet training iterations can be reduced without significantly hurting performance.

In Table 6, we shorten supernet the training stage from 150K iterations to 100K iterations. Under the same flops, the top-1 accuracy searched by our method only decreases by 0.1, while the accuracy searched by SPOS drops by 0.4.

Therefore, under some resource-limited situations where the supernet cannot be trained for sufficient iterations and thus can't provide accurate estimation, our method can help to maintain performance.

5. Conclusion and Outlook

In this work, We have proposed supernet shifting, a strong and flexible method to improve order-preserving ability and transferability for Neural Architecture Search. With little additional time cost, the supernet can focus on superior architectures and thus ensure both local and global order-preserving ability. The search quality can be improved. Extensive experiments demonstrate the effectiveness.

Besides, a pre-trained supernet can be transferred to a new dataset in searching. This gives space for developing new NAS pipelines for future study. We believe a supernet pre-trained on a large-scale dataset will be open source and can be transferred to other tasks effectively. Optimal architectures can be searched by only applying searching algorithms with no need of tricky and time-consuming supernet training. This will significantly reduce the time cost, hardware restriction and searching difficulty for NAS. NAS and AutoML can be widely applied to various situations.

Limitations. For efficiency reasons, our supernet shifting is highly dependent on the sampling by EA. Although statistically correct, there are no explicit expressions on the sampling distribution. Mutation makes the sampling even more uncontrollable. Besides, whether it's fair and effective for different architectures to compare in an ever-changing supernet still needs to be verified.

References

[1] Yash Akhauri, Juan Pablo Muñoz, Nilesh Jain, and Ravi Iyer. EZNAS: evolving zero-cost proxies for neural architecture scoring. In *NeurIPS*, 2022. 1

[2] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *ICLR*, 2017. 1, 2

[3] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Understanding and simplifying one-shot architecture search. In *ICML*, pages 549–558, 2018. 1, 3, 7

[4] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019. 3

[5] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *ICCV*, pages 1294–1303, 2019. 3, 4

[6] Xiangxiang Chu, Bo Zhang, and Ruijun Xu. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. In *ICCV*, pages 12219–12228, 2021. 1, 2, 3, 4, 7

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009. 3, 6

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL*, pages 4171–4186, 2019. 2

[9] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four GPU hours. In *CVPR*, pages 1761–1770, 2019. 2, 3, 4

[10] Jiemin Fang, Yuzhu Sun, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. Densely connected search space for more flexible neural architecture search. In *CVPR*, pages 10625–10634, 2020. 3

[11] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *ECCV*, pages 544–560, 2020. 2, 3, 4, 7, 8

[12] Hyeonmin Ha, Ji-Hoon Kim, Semin Park, and Byung-Gon Chun. SUMNAS: supernet with unbiased meta-features for neural architecture search. In *ICLR*, 2022. 2, 6

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 1

[14] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross B. Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, pages 15979–15988, 2022. 2

[15] Weijun Hong, Guilin Li, Weinan Zhang, Ruiming Tang, Yunhe Wang, Zhenguo Li, and Yong Yu. Dropnas: Grouped operation dropout for differentiable architecture search. *CoRR*, abs/2201.11679, 2022. 2, 4

[16] Shoukang Hu, Ruochen Wang, Lanqing Hong, Zhenguo Li, Cho-Jui Hsieh, and Jiashi Feng. Generalizing few-shot NAS with gradient matching. In *ICLR*, 2022. 2

[17] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 3, 6

[18] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Snip: single-shot network pruning based on connection sensitivity. In *ICLR*. OpenReview.net, 2019. 1

[19] Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan L. Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *CoRR*, abs/1712.00559, 2017. 1, 2

[20] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: differentiable architecture search. In *ICLR*, 2019. 2, 3, 4

[21] Shun Lu, Yu Hu, Longxing Yang, Zihao Sun, Jilin Mei, Jianchao Tan, and Chengru Song. Pa&da: Jointly sampling path and data for consistent NAS. In *CVPR*, pages 11940–11949, 2023. 2

[22] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet V2: practical guidelines for efficient CNN architecture design. In *ECCV*, pages 122–138, 2018. 6

[23] Joe Mellor, Jack Turner, Amos J. Storkey, and Elliot J. Crowley. Neural architecture search without training. In *ICML*, pages 7588–7598, 2021. 1

[24] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *ICML*, pages 4092–4101, 2018. 1, 2

[25] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka I. Leon-Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *ICML*, pages 2902–2911, 2017. 3

[26] Gresa Shala, Thomas Elsken, Frank Hutter, and Josif Grabocka. Transfer NAS with meta-learned bayesian surrogates. In *ICLR*, 2023. 3

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1

[28] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path NAS: designing hardware-efficient convnets in less than 4 hours. In *ECML*, pages 481–497, 2019. 3, 4

[29] Xiu Su, Shan You, Mingkai Zheng, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. K-shot NAS: learnable weight-sharing for NAS with k-shot supernets. In *ICML*, pages 9880–9890, 2021. 2

[30] Chaoqi Wang, Guodong Zhang, and Roger B. Grosse. Picking winning tickets before training by preserving gradient flow. In *ICLR*. OpenReview.net, 2020. 1

[31] Dilin Wang, Meng Li, Chengyue Gong, and Vikas Chandra. Attentivenas: Improving neural architecture search via attentive sampling. In *CVPR*, pages 6418–6427, 2021. 6, 7

[32] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *ICLR*, 2019. 3

[33] Jin Xu, Xu Tan, Kaitao Song, Renqian Luo, Yichong Leng, Tao Qin, Tie-Yan Liu, and Jian Li. Analyzing and mitigating interference in neural architecture search. In *ICML*, pages 24646–24662, 2022. 2

[34] Chao Xue, Junchi Yan, Rong Yan, Stephen M. Chu, Yonggang Hu, and Yonghua Lin. Transferable automl by model sharing over grouped datasets. In *CVPR*, pages 9002–9011, 2019. 3

[35] Yu Xue and Jiafeng Qin. Partial connection based on channel attention for differentiable neural architecture search. *IEEE Trans. Ind. Informatics*, 19(5):6804–6813, 2023. 2, 4

[36] Shan You, Tao Huang, Mingmin Yang, Fei Wang, Chen Qian, and Changshui Zhang. Greedynas: Towards fast one-shot NAS with greedy supernet. In *CVPR*, pages 1996–2005, 2020. 6, 7

[37] Miao Zhang, Huiqi Li, Shirui Pan, Xiaojun Chang, and Steven Su. Overcoming multi-model forgetting in one-shot nas with diversity maximization. In *CVPR*, pages 7806–7815, 2020. 6

[38] Yiyang Zhao, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo. Few-shot neural architecture search. In *ICML*, pages 12707–12718, 2021. 6

[39] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 1, 2

[40] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *CVPR*, pages 8697–8710, 2018. 1, 2