Automatic Subspace Evoking for Efficient Neural Architecture Search

Yaofo Chen, Yong Guo, Daihai Liao, Fanbing Lv, Hengjie Song, and Mingkui Tan

***Abstract*—Neural Architecture Search (NAS) aims to automatically find effective architectures from a predefined search space. However, the search space is often extremely large. As a result, directly searching in such a large search space is non-trivial and also very time-consuming. To address the above issues, in each search step, we seek to limit the search space to a small but effective subspace to boost both the search performance and search efficiency. To this end, we propose a novel Neural Architecture Search method via Automatic Subspace Evoking (ASE-NAS) that finds promising architectures in automatically evoked subspaces. Specifically, we first perform a global search, *i.e.*, automatic subspace evoking, to evoke/find a good subspace from a set of candidates. Then, we perform a local search within the evoked subspace to find effective architectures. More critically, we further boost search performance by taking well-designed/searched architectures as the initial candidate subspaces. Experimental results demonstrate that our method not only greatly reduces the search cost but also finds better architectures than state-of-the-art methods in various benchmark search spaces.**

***Index Terms*—Neural Architecture Search, Search Space Evoking, Subspace Graph, Global Search and Local Search, Search Efficiency, Convolutional Neural Networks**

I. Introduction

DEEP neural networks (DNNs) have been the workhorse of many challenging tasks, including image classification [1]–[4], action recognition [5]–[8] and neural language processing [9], [10]. The success of DNNs is largely attributed to the innovation of effective neural architectures. However, designing effective architectures often greatly depends on expert knowledge and human efforts. Thus, it is non-trivial to design architectures to satisfy the requirements manually. To address this, neural architecture search (NAS) [11] is developed to automate the process of the architecture design.

Existing NAS methods search for effective architectures in a predefined search space [11]–[13]. To cover as many good architectures as possible, the search space is often designed to be extremely large (*e.g.*, $\sim 10^{12}$ in ENAS [14] and $\sim 10^{11}$ in DARTS [12]). Directly searching in such a large space is very difficult and time-consuming in practice. Specifically, to explore the large search space, we have to sample and evaluate plenty of architectures, which is very computationally expensive and time-consuming. Moreover, we can only access a small proportion of architectures in the search space due to the limitation of the computational resources in practice. In other words, regarding a very large search space, we can only obtain limited information to guide the architecture search.

To overcome the above difficulties brought by the large search space, PNAS [15] and CNAS [16] propose to start from a very small search space to perform an architecture search and then gradually enlarge the search space by adding nodes or operations. Recently, AlphaX [17] partitions the search space into existing good subspaces and unexplored subspaces and adopts the Monte Carlo Tree Search (MCTS) method to encourage exploring the good ones. However, these methods suffer from two limitations. **First**, these methods still find architectures from a very large space at each search step, which may not only result in unnecessary explorations but also affect the search results. **Second**, the small search spaces partitioned by these methods are fixed during the search phase, which may not be optimal to find good architectures. Thus, how to find/design a small but effective search space that covers as many good architectures as possible is an important problem.

To achieve this goal, an underlying hypothesis is that the neighborhood around an effective architecture is usually a good subspace for further exploration. In this case, once we find the small but effective search space around a promising architecture, it is more likely to find a better architecture within the subspace rather than the entire search space. We empirically verify this hypothesis that similar architectures tend to have close performance (See the results of Figures 6, as well as the observations in [16], [18]). Inspired by this, instead of searching in the whole search space, we seek to limit the search space within a reduced one in each search step by recognizing and evoking a small but effective subspace. In this sense, we may boost the search performance and search efficiency as well since we only focus on an evoked effective subspace, in which it is easier to find good architectures than directly exploring the whole search space.

In this paper, we propose an efficient Neural Architecture Search method via Automatic Subspace Evoking (called ASE-NAS). The key idea is to find/evoke a small but effective subspace from the whole search space in each step of the architecture search. To this end, we first construct a set of candidate subspaces and then build a subspace graph above them,

Y. Chen, Y. Guo, H. Song and M. Tan are with the School of Software Engineering, South China University of Technology, Guangzhou 510641, China (e-mail: chenyaofo@gmail.com; guoyongcs@gmail.com; sehjsong@scut.edu.cn; mingkuitan@scut.edu.cn)

D. Liao and F. Lv are with Changsha Hisense Intelligent System Research Institute Co., Ltd, Changsha 410006, China (e-mail: liaodaihai@hisense.com; lvfanbing@hisense.com)

This work was partially supported by Key-Area Research and Development Program of Guangdong Province (2019B010155002, 2019B010155001), National Natural Science Foundation of China (NSFC) 61836003 (key project), National Natural Science Foundation of China (NSFC) 62072190, National Key R&D Program of China (No.2020AAA0106900), Key Realm R&D Program of Guangzhou 202007030007, Program for Guangdong Introducing Innovative and Enterpreneurial Teams 2017ZT07X183.

Y. Chen and Y. Guo contributed equally to this paper.

H. Song and M. Tan are the corresponding authors.

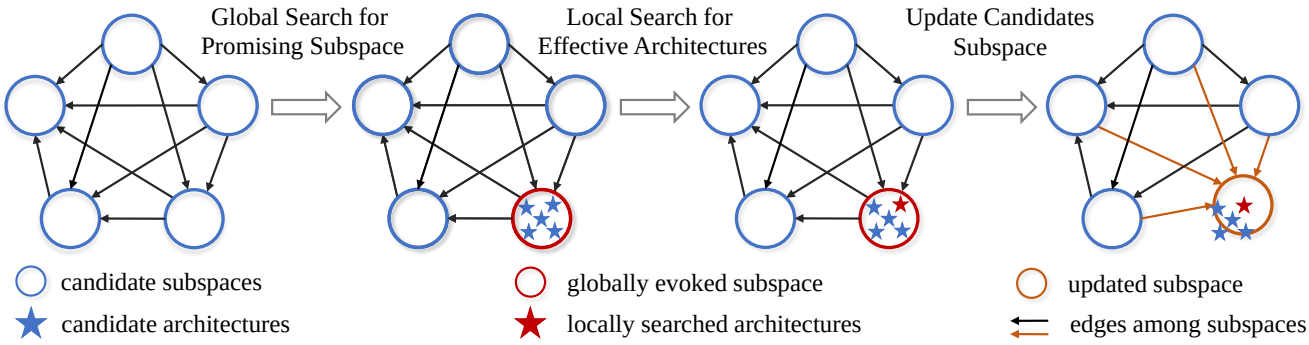


Fig. 1: An illustration of the search process. We find promising architectures in a two-step search manner: 1) we perform a global search to evoke/find a promising subspace from a set of candidates; 2) we move the focus on the subspace and conduct a local search for effective architectures within it. Then, we update the candidate subspace with the better searched architecture.

in which edges denote the relationships/information among different subspaces. As shown in Figure 1, we first perform a *global search* to evoke/find a promising subspace from the subspace graph. Then, we focus on the searched subspace and conduct a *local search* to obtain the resultant architectures. It is worth noting that, once we find a better architecture, we would also update the subspace graph according to it. In this way, it becomes possible to gradually find better architectures during the search process. Moreover, we are able to further boost our ASE-NAS by taking existing well-designed architectures (*e.g.*, OFA architecture [19]) to construct candidate subspaces. Extensive experiments in two benchmark search spaces show the effectiveness of the proposed method.

Our contributions are summarized as follows:

- Instead of searching in the entire search space, we seek to find/evoke a small but effective subspace for each step of the architecture search. With the help of the automatically evoked subspaces, we are able to improve search performance and efficiency simultaneously.
- We propose a novel Automatic Subspace Evoking algorithm to enhance the performance of neural architecture search. Specifically, we first perform a global search to find promising subspaces and then perform a local search to obtain the resultant architectures.
- Extensive experiments demonstrate the superiority of the proposed method over existing NAS methods. More critically, the searched subspaces also exhibit promising transferability to a new dataset.

II. Related Work

A. Neural Architecture Search

In recent years, NAS has drawn great attention for the effective architectures designing. As the pioneering work, Zoph and Le [11] use reinforcement learning (RL) to discover the optimal configuration of each neural network layer. After that, many RL-based methods [20]–[23] enhance the performance of searched architectures of the above method. Besides, evolutionary based methods [24]–[28] search for promising architectures by gradually evolving a population. Moreover, gradient based methods [12], [29]–[32] find effective architectures by relaxing the search space to be continuous.

However, traditional NAS methods [20], [33] often require great computational resources and thus result in unaffordable time costs. To improve search efficiency, plenty of efforts have been made to reduce the computational cost during search. Weight-sharing based NAS methods [14], [19], [22] estimate the performance of candidate architectures by inheriting weights from a trained supernet, which greatly lowers the computational cost of architecture evaluation. In addition, EcoNAS [34] carefully designs a proxy training strategy to evaluate candidate architectures on small proxy datasets. Unlike these methods, our method achieves high efficiency from a different perspective. To be specific, we reduce the number of architecture evaluations by searching in small and effective subspaces instead of the whole large space.

B. Search Space Design of NAS Methods

NAS methods often find promising architectures in a predefined large search space, such as NASNet [35], DARTS [12] and MobileNet-like [13] search spaces. Most existing methods directly perform searching in these large search spaces, which may not only result in inefficient sampling but also hamper the search performance. To address this issue, local search methods [36] search in an iterative manner: travel all architectures in the neighborhood of an architecture and update it with the best-found architecture, which is computationally expensive since they require plenty of architectures evaluations. Recent works [15], [16], [37] seek to search from a small search space and enlarge the space by progressively adding the nodes or operations. In addition, AlphaX [17] and LaNAS [38] build a Monte Carlo Search Tree to partition the search space into different subspaces according to their performance and encourage to explore the promising subspaces. However, these methods still need to generate architecture from the overall search space and subspaces designed by them are fixed and may not be optimal. Unlike these methods, our global and local search strategy is able to recognize and evoke the small but effective subspaces from candidates dynamically during the search process and find effective architectures in it.

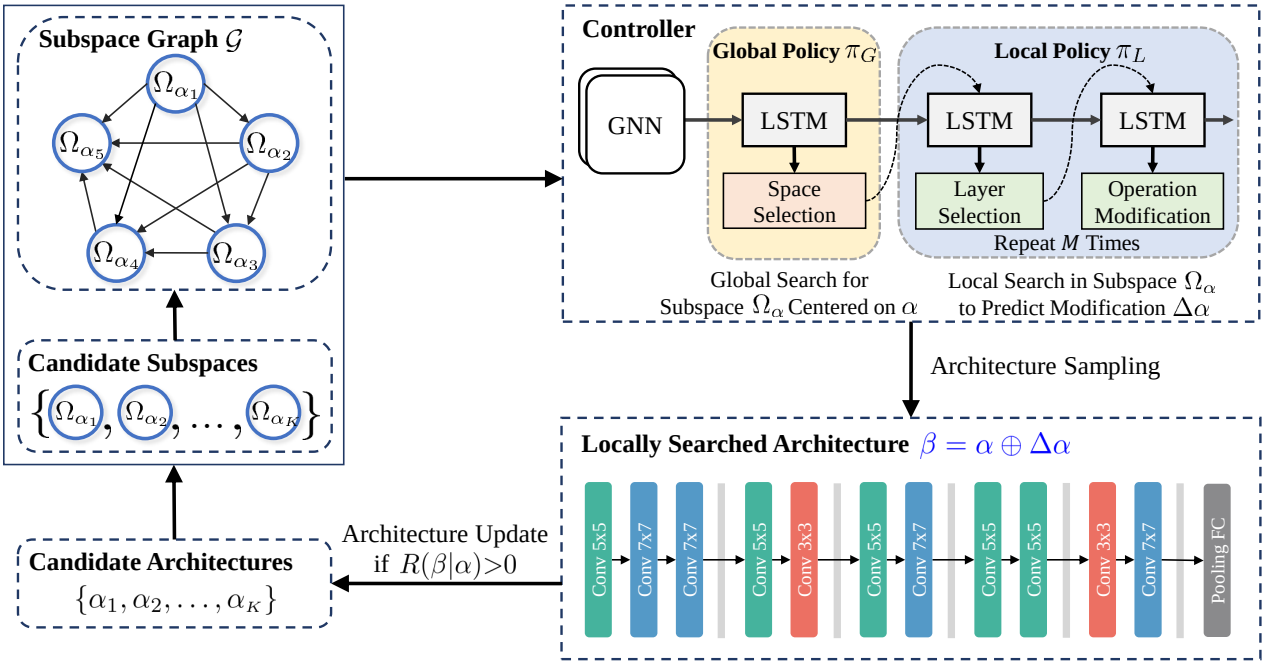


Fig. 2: An overview of our ASE-NAS. We build a set of subspaces $\{\Omega_i\}_{i=1}^K$ centered on randomly sampled candidate architectures $\{\alpha_i\}_{i=1}^K$ and construct a subspace graph $\mathcal{G}$ to model the relationships among these subspaces. By taking $\mathcal{G}$ as the input, the controller first evokes a promising subspace $\Omega \sim \pi_G(\cdot|\mathcal{G}; \theta_G)$ via global search policy and then predicts an architecture modification $\Delta \sim \pi_L(\cdot|\Omega; \theta_L)$ via local search policy. Next, we update the candidate architecture α with the resultant architecture $\beta = \alpha \oplus \Delta$ if β has better performance than α (*i.e.*, $R(\beta|\alpha) > 0$).

III. Architecture Search via Subspace Evoking

In this paper, we propose the Automatic Subspace Evoking (ASE-NAS) method to boost both the search performance and efficiency of NAS. In Section III-A, we first discuss the motivation and the overview of ASE-NAS. Then, we depict the details of two key steps of our method, *i.e.*, the global search and local search in Sections III-B and III-C, respectively.

A. Motivation and Method Overview

Existing NAS methods often consider an extremely large search space Ω to find good architectures [12], [14], [19]. However, directly performing architecture search in such a large search space is non-trivial and often very expensive. Regarding these issues, instead of using the whole search space, it should be possible to limit each search step within a reduced search space to boost the search performance and search efficiency as well. To achieve this goal, an underlying hypothesis is that the neighborhood around an effective architecture is usually a good subspace for further exploration to find better ones. In this case, we are more likely to find effective architectures in the neighborhood/subspace than the whole search space under the same budget of search cost.

Inspired by this, we propose a new search algorithm by evoking/identifying effective subspaces, in which it is easier to find good architectures than directly exploring the whole space. To this end, we define a subspace Ω based on a center architecture α within it (See more details in Section III-B). As shown in Figure 2 and Algorithm 1, we first learn a global search policy π_G to search for a promising subspace Ω based on a center architecture α. Then, we further learn a local search policy π_L to produce the resultant architectures within the subspace. Specifically, the global search policy π_G takes a set of candidate subspaces $\{\Omega_i\}_{i=1}^K$ (as well as their relationship) as inputs and evokes/finds a promising subspace Ω. Based on the evoked/searched subspace, the local policy π_L further generates a modification Δ (*i.e.*, modifying some operations of some layers in α) to explore the subspace. Finally, we combine α and Δ to obtain the resultant architecture by

$$\beta = \alpha \oplus \Delta \; . \tag{1}$$

Here, $\oplus$ denotes the combination operation, as will be depicted in Section III-C. Note that Δ is devised to constrain the architecture after modifications still in the subspace.

It is worth noting that, if we directly maximize the performance of the resultant architecture β, the search algorithm may always select the same subspace with the best center architecture at the current step and thus easily get stuck in a local optimum (See results in supplementary). To avoid this, we encourage the exploration ability by maximizing the performance improvement between β and the center architecture α in Ω, *i.e.*, $R(\beta|\alpha) = R(\beta; w_\beta) - R(\alpha; w_\alpha)$, where $R(\alpha; w_\alpha)$ denotes some performance metric (*e.g.*, validation accuracy) and w_α denotes the optimal parameters of α trained on some dataset. This can be thought of as finding the subspace with the largest potential to find better architectures, instead of the one containing the best architecture found previously. Formally, we seek to solve the following optimization problem:

$$\max_{\theta_G, \theta_L} \mathbb{E}_{\pi_G} [\mathbb{E}_{\Delta \sim \pi_L} R(\beta|\alpha)]; \quad \beta = \alpha \oplus \Delta \; ; \tag{2}$$

Algorithm 1 Training method for ASE-NAS.

Require: The search space Ω, the global policy $\pi_G(\cdot;\theta_G)$ and the local policy $\pi_L(\cdot;\theta_L)$.

```
Train the parameters of the supernet.
Randomly sample architectures {α_i}_{i=1}^K from Ω to build the subspaces {Ω_{α_i}}_{i=1}^K using Eqn. (3).
Construct the subspace graph G based on {Ω_{α_i}}_{i=1}^K.
while not convergent do
    // Perform a global search to evoke a promising subspace Ω_α
    Sample a subspace Ω_α ~ π_G(·|G;θ_G) centered on α.
    // Perform a local search in the evoked subspace Ω_α
    Sample modifications Δα ~ π_L(·|Ωα;θ_L).
    Build a resultant architecture β = α ⊕ Δα.
    Compute reward R(β|α)=R(β,w_β)−R(α,w_α) using the weights inherited from the supernet.
    // Update subspaces with the locally searched architecture β
    if R(β|α) > 0 then
        Replace the candidate subspace Ω_α with Ω_β.
        Update the edges connected to Ω_β in G.
    end if
    Update the parameters θ_G and θ_L by optimizing Eqn. (2) using policy gradient [39].
end while
```

Since we would update the searched subspace with the resultant architecture (See Figure 2), the performance improvement of the previously searched subspace may not always be the largest one and our method is able to explore other subspaces in the subsequent iterations.

B. Global Search with Automatic Subspace Evoking

As the first step of ASE-NAS, we seek to find/evoke effective subspaces via a global search process. Specifically, we first discuss the construction of candidate subspaces. Then, we depict the details of our global search algorithm.

Subspace Construction. At the beginning of our search method, we seek to construct candidate subspaces centered on a set of architectures for search. To this end, we randomly collect a set of discrete architectures $\{\ _i\}_{i=1}^K$ from the search space and build the candidate subspaces $\{\Omega\ _i\}_{i=1}^K$ around them. Let $D(\ ;\ )$ be a function to measure the *Architecture Distance* between two architectures and . Specifically, we take an architecture as the center of the subspace Ω and constrain that all the architectures in Ω have distances less than a specific threshold M from the center architecture , *i.e.*, $D(\ ;\ ) \le M$. Formally, we construct the subspace Ω w.r.t. a center architecture as follows:

$$\Omega\ = \{\ \mid D(\ ;\ ) \le M;\quad \in \Omega\}: \tag{3}$$

Note that architectures in different subspaces may have different operations/topologies and different performances as well. To exploit the relationship/information among different subspaces, as shown in Figure 2, we build a subspace graph $\mathcal{G} = (\mathcal{V};\mathcal{E})$ to guide the search. Here, $\mathcal{V}$ is a set of nodes and each node denotes a specific subspace $\Omega\ _i$. $\mathcal{E}$ is a set of directed edges from a weak subspace (with a worse center architecture) to a better subspace (whose center architecture has higher accuracy). Note that, instead of randomly initializing the center architectures, we may further improve our ASE-NAS by taking existing well-designed/searched architectures to construct subspace (See results in Tables I and II).

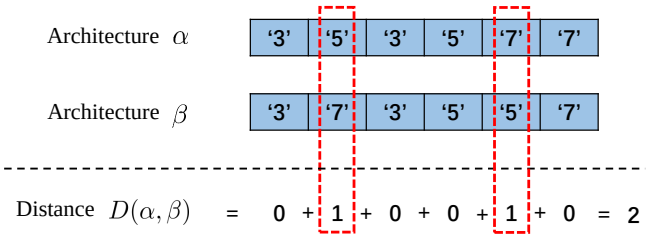


Fig. 3: An illustration of the architecture representation method and calculation of the architecture distance. We represent an architecture as a string, in which each item denotes an operation (*e.g.*, convolution). For example, '3', '5' and '7' denote 3×3, 5×5 and 7×7 convolution, respectively.

Searching for Effective Subspaces. In each search step, we conduct a global search to find/evoke a promising subspace Ω from all candidate subspaces $\{\Omega\ _i\}_{i=1}^K$. As mentioned before, we seek to find Ω that has the potential to achieve a large performance improvement $R(\ \mid\ )$. (as aforementioned in Eqn. (2)). Here, is the center architecture of Ω and is another architecture in this subspace. To this end, we devise a controller model which contains a two-layer graph neural network (GNN) [40] and an LSTM network. [1] Specifically, to exploit the information of the subspace graph $\mathcal{G} = (\mathcal{V};\mathcal{E})$, we first employ the GNN model to extract features from $\mathcal{G}$. Then, we feed the extracted features into the LSTM network that samples a candidate subspace Ω via a softmax classifier.

Note that the search performance greatly depends on the initial subspaces, if we fix all candidate subspaces in $\mathcal{G}$ during search. In this sense, the controller may get stuck in a local optimum due to the very limited search space (See results in Figure 7). To address this issue, we propose a simple strategy to gradually update/improve the candidate subspaces $\{\Omega\ _i\}_{i=1}^K$ using the newly searched architectures. Specifically, for a selected subspace Ω (See Figure 2), we replace its center architecture with the locally searched architecture if yields better performance than . This follows that the subspace around better architecture may be more likely to contain promising architectures. Once the center architecture updates, the corresponding subspace also gets updated. Then, we will also update the subspace graph $\mathcal{G}$ by updating all the edges that are connected to Ω . In this way, the candidate subspaces constantly improve, which helps to explore more and more promising spaces. After search, we select the best center architecture in the subspace graph as the inferred architecture according to the validation performance.

C. Local Search within the Evoked Subspace

Given a searched subspace Ω , we then perform a local search to find effective architectures. To guarantee that the search process is limited within the subspace, we first discuss how to measure the distance between architectures. Then, we depict the details of our local search algorithm.

[1] More details about GNN are put in Appendix.

Before defining the *Architecture Distance* between two architectures, we first revisit the representation method of architectures, making it easier to be understood. Following [14], [19], one can represent an architecture as a L-dimensional string $=[\ ^{(1)};\ ^{(2)};\cdots;\ ^{(L)}]$, where L denotes the number of components in the architecture and each item $^{(i)}$ denotes some operation (*e.g.*, convolution). We propose to compute the *Architecture Distance* $D(\cdot;\cdot)$ by counting the number of different components between two architectures (See Figure 3). Let $\mathbb{1}\{\cdot\}$ be an indicator function. Given two architectures and , the distance between them is computed by

$$D(\ ;\ ) := \sum_{i=1}^{} \mathbb{1}\{\ ^{(i)} \neq\ ^{(i)}\}; \tag{4}$$

Here, we justify the proposed architecture distance by empirically investigating the relationship between accuracy and architecture distance (See results in Figure 6). We observe that the architectures with smaller distances tend to have similar performance, which is also observed in popular NAS methods [16], [18]. In this sense, once we find a promising subspace centered on a good architecture, it would be much easier to find better architectures via a local search.

To ensure that the locally searched architecture belongs to Ω , we devise a local policy $_L$ that modifies the center architecture by M times. Specifically, for each time modification, the local policy $_L$ determines which layer to be modified and which kind of operation to be applied to this layer. Such decision process will be repeated M times to obtain the complete modification Δ . Then, by applying Δ to , we obtain the modified/searched architecture $= \oplus\Delta$ in the subspace. Note that if $_L$ selects the same operation as the original one in some layer, it will produce no modification to the architecture. Thus, after M times single-layer modification, the resultant architecture still belongs to Ω , *i.e.*, satisfying the constraint $D(\ ;\ ) \leq M$.

Analysis on the size of local search space. Let L be the number of components in an architecture and C be the number of candidate operations for each component. The size of the whole search space is $|\Omega| = C^L$. Given a local search distance M, the size of the subspace becomes $|\Omega\ | = \ ^L_M\ C^M$, where denotes the combination function. When we consider a very small $M \ll L$, the subspace would be much smaller than the whole space. In this case, the union of these subspaces constructed with a small M may not cover the entire search space. Nevertheless, it is exactly our key idea that we seek to focus on some promising subspaces instead of the entire search space to enhance both the search performance and search efficiency. In the extreme case, when we consider $M{=}L$, the subspace is exactly the whole space and our method will be reduced to the standard NAS. Thus, we investigate the effect of M on the performance of ASE-NAS in Section V-F.

IV. Experiments

The experiments are organized as follows. We first perform experiments in NAS-Bench-201 [41] search space to demonstrate the effectiveness of our ASE-NAS. Then, we evaluate our ASE-NAS in MobileNet-like [46] search space and compare the performance of our method with state-of-the-art methods on ImageNet [47].

A. Performance Comparisons on NAS-Bench-201

Search Space. We apply our ASE-NAS to a cell-based NAS-Bench-201 search space [41]. Each cell is a directed acyclic graph with 4 nodes and 6 edges. Each edge is associated with an operation, which has 5 different candidates, including *zeroize*, *skip connection*, *1×1 convolution*, *3×3 convolution* and *3×3 average pooling*. Since we search for the candidate operation for each edge, there are $5^6 = 15625$ candidate architectures in total. For each architecture, NAS-Bench-201 provides precomputed training, validation, and test accuracies on three different datasets, namely CIFAR-10, CIFAR-100 and ImageNet-16-120. Note that ImageNet-16-120 is a subset of ImageNet [47] dataset with 120 classes and 16×16 image resolution.

Implementation Details. Following the settings in NAS-Bench-201 [41], we use the validation accuracy in epoch 12 as the reward and report the test accuracy in epoch 200 to compare with other baseline methods. For a fair comparison, we consider the evaluation time of candidate architectures when computing the search cost (limited to 25k seconds). Following the setting in [14], we train our ASE-NAS with a batch size of 1 and set the strength of the entropy regularizer to $7{:}5 \times 10^{-4}$. We use an Adam optimizer with a learning rate of 1×10^{-2}. We set the number of candidate subspaces K to 4 and the search distance M to 4. In practice, we combine the global policy $_G$ and the local policy $_L$ into a single policy to make decisions to predict the promising subspace Ω and the modification Δ in succession. In other words, we seek to learn a joint policy that first selects a candidate subspace and then finds a promising architecture modification within the selected subspace. Note that combining the global and local policies together is equivalent to treating them individually.

Comparisons with State-of-the-art Methods. We compare our method with two baselines, namely *Random Search* and *REINFORCE* [39]. Random Search baseline randomly samples architectures and selects one with the highest accuracy among them as the final derived architecture. REINFORCE baseline performs searching by directly maximizing the expectation of the performance of sampling architectures with reinforcement learning. We report the average test accuracy on three datasets over 500 runs with different seeds. In this experiment, we initialize the subspace graph with randomly sampled centered architectures. We also conduct experiments using the subspace graph initialized with searched well-designed architectures, which achieves better search performance.

From Table I, compared with the baselines, our ASE-NAS achieves the highest average accuracy on three datasets, *i.e.*, CIFAR-10, CIFAR-100 and ImageNet-16-120. Compared with REINFORCE [39] baseline, the proposed ASE-NAS yields better search accuracy and lower variance (*e.g.*, 72.76±0.80% *vs.*71.71±1.09% on CIFAR-100). The reason is our ASE-NAS searches by focusing on the small but effective subspace, which reduces the search difficulty resulting from the large

TABLE I: Comparisons with existing methods in NAS-Bench-201 [41]. "ImageNet-16-120'" denotes a subset of ImageNet dataset with 120 classes and 16×16 resolution. "Search Cost" denotes the time cost in the search phase (measured by second). $^{\gamma}$ denotes we implement the baselines with the official code under our same settings. Our ASE-NAS achieves higher accuracy than state-of-the-art methods with less search cost, which verify the search performance and efficiency of our method.

Method	Search Cost (s)	CIFAR-10		CIFAR-100		ImageNet-16-120	
Random Search†	25k	93.88	0.27	71.54	1.04	45.19	1.06
REINFORCE [39]†	25k	93.85	0.37	71.71	1.09	45.24	1.18
PNAS [15]†	25k	93.71	0.29	70.89	0.99	44.75	0.80
CNAS [16]†	25k	93.95	0.28	71.73	1.05	45.46	0.97
ENAS [14]	–	53.89	0.58	13.96	2.33	14.84	2.10
DARTS [12]	30k	54.30	0.00	15.61	0.00	16.32	0.00
SETN [42]	34k	87.64	0.00	59.05	0.24	32.52	0.21
GDAS [43]	32k	93.61	0.09	70.70	0.30	41.71	0.98
DSNAS [44]	–	93.08	0.13	31.01	16.38	41.07	0.09
PC-DARTS [45]	–	93.41	0.30	67.48	0.89	41.31	0.22
ASE-NAS (Ours)	25k	**94.23**	**0.22**	72.76	0.80	46.13	0.67
ASE-NAS+ (Ours)	25k	–		**73.12**	**0.61**	**46.66**	**0.52**

search space. During the search, our ASE-NAS updates the candidate subspaces with the locally searched architectures and finds better architectures in the constantly improved subspaces. Besides the superior performance, we also highlight that the search time cost of our ASE-NAS is 25k seconds, which is much more efficient than most state-of-the-art NAS methods, such as DARTS [12], SETN [42] and GDAS [43]. These results demonstrate the superiority of our proposed ASE-NAS over the considered methods.

Transferability of Subspace Graph to New Datasets. When we search on a new target dataset, we often have to search from scratch (*i.e.*, initializing candidate subspaces with randomly sampled architectures). However, we may have found some promising architectures on existing datasets. In this case, we can introduce a new variant of our method, called ASE-NAS+, which may enhance the search performance on the target dataset by transferring the subspaces of the subspace graph found in the existing datasets. To verify this, we conduct experiments on NAS-Bench-201 by considering CIFAR-10 as the existing dataset and CIFAR-100 as well as ImageNet-16-120 as two new target datasets.

From the results in Table I, the ASE-NAS+ with the subspaces transferred from CIFAR-10 achieves higher accuracy and lower variance than the ASE-NAS without that (73.12±0.61 *vs.* 72.76±0.80 on CIFAR-100, 46.66±0.52 *vs.* 46.13±0.67 on ImageNet-16-120). The reason is that the well-designed architectures on CIFAR-10 may also have high performance on other datasets (*e.g.*, CIFAR-100 and ImageNet-16-120). In this sense, subspaces searched in CIFAR-10 provide a good initialization when searching in a new dataset. These results show the transferability of the candidate subspaces between two different datasets and the potential of our method when applied to new datasets.

B. Performance Comparisons on ImageNet

Search Space. In this part, we further evaluate our method on another benchmark search space, *i.e.*, MobileNet-like search space [46]. The candidate architecture consists of 5 different units and each of them has consecutive layers. We search for MBConv in each layer with kernel sizes selected from $\{3; 5; 7\}$, expansion rates selected from $\{3; 4; 6\}$ and the number of layers in each unit selected from $\{2; 3; 4\}$. Following [53], we randomly choose 10% classes from the original dataset as the training set to train the supernet. We measure the validation accuracy of sub-networks on 1000 validation images sampled from the training set. We train the supernet with a progressive shrinking strategy [19] for 90 epochs. To compute the performance improvement $R(\ |\ )$ with the validation accuracy, we train a predictor to predict the validation accuracy following [19].

Implementation Details. Following [14], we train the ASE-NAS for 10k iterations with a batch size of 1. We use an Adam optimizer with a learning rate of 3×10^{-4}. To encourage the exploration of the ASE-NAS, we add an entropy regularizer to the reward weighted by 1×10^{-3}. We set the number of candidates subspaces K to 10 and the local search distance M to 3. We report the search cost based on NVIDIA Tesla V100 GPU. Following the *mobile setting* [12], we constraint the number of multiply-adds (#MAdds) of the searched architecture to be less than 600M. To achieve this, we update the candidate subspace (See line 12 in Algorithm 1) with the architecture which has #MAdds less than 600M. To accelerate model evaluation, following [19], [24], we first obtain the parameters from the full network of OFA and then finetune them for 75 epochs. We perform data augmentations including horizontally flipping, random crops, color jittering, and AutoAugment. We use an SGD optimizer with a learning rate of 0.012. The learning rate decay follows the cosine annealing strategy with a minimum of 0.001.

Comparisons with State-of-the-art Methods. To investigate the effectiveness of the proposed method, we apply our method to MobileNet-like search space as two variants: 1) **ASE-NAS** searches based on the subspace graph initialized with a set of randomly sampled architectures, which is suitable in the scenario without any available well-designed

TABLE II: Comparisons of the architectures searched/designed by different methods on ImageNet. "–" means unavailable results. "#Queries", a widely used metric [48], [49], denotes the number of architecture-accuracy pairs queried from supernet or performance predictor during the search. A smaller "#Queries" means the search algorithm is more efficient. Our ASE-NAS outperforms than most human designed and automatically searched architectures with less search cost and fewer search queries.

Search Space	Architecture	Test Accuracy (%)		#MAdds (M)	#Queries (k)	Search Time (GPU days)
		Top-1	Top-5			
–	ResNet-18 [1]	69.8	89.1	1,814	–	–
	MobileNetV2 (1.4×) [50]	74.7	–	585	–	–
	ShuffleNetV2 (2×) [51]	73.7	–	524	–	–
NASNet	NASNet-A [35]	74.0	91.6	564	20	1800
	AmoebaNet-A [33]	74.5	92.0	555	20	3150
DARTS	DARTS [12]	73.1	91.0	595	19.5	4
	P-DARTS [52]	75.6	92.6	577	11.7	0.3
	CNAS [16]	75.4	92.6	576	100	0.3
	AlphaX [17]	75.5	92.2	579	–	12
MobileNet-like	MobileNetV3-Large [46]	75.2	–	219	–	–
	FBNet-C [53]	74.9	–	375	11.5	9
	AtomNAS [54]	75.9	92.0	367	78	–
	ProxylessNAS [55]	75.1	92.3	465	–	8.3
	FairNAS [56]	77.5	93.7	392	11.2	12
	DNA [57]	78.4	94.0	611	–	32.6
	FBNetV2 [58]	77.2	–	325	11.5	25
	EfficientNet-B1 [59]	79.2	94.5	734	–	–
	OFA-Large [19]	80.0	94.9	595	20	51.7
	Cream-L [60]	80.0	94.7	604	–	12
	NEAS-L [25]	80.0	94.8	574	–	$<$13
	ASE-NAS (Ours)	79.9	94.8	597	**10**	**0.8**
	ASE-NAS+ (Ours)	**80.2**	**94.9**	582	**10**	51.7+0.8

architectures. 2) **ASE-NAS+** adopts the subspace graph which is initialized with a set of existing well-designed architectures searched by OFA [19].

As shown in Table II, under the mobile setting, the architecture searched by ASE-NAS reaches 79.9% top-1 accuracy and 94.8% top-5 accuracy, which outperforms not only the manual designed architectures but also most automatic searched ones. Specifically, ASE-NAS outperforms the best manually designed architecture (*i.e.*, MobileNetV2) by 5.2% (*i.e.*, 79.9% *vs.* 74.7%). Compared with the state-of-the-art NAS method (*e.g.*, OFA and Cream-L), ASE-NAS also achieves competitive performance (*i.e.*, 79.9% *vs.* 80.0%) with less number of queries and search cost (only 0.8 GPU days). Here, the lower costs mainly benefit from two aspects: 1) accelerating the search policy learning by reducing the number of queries (ours 10k *vs.* OFA 20k); 2) accelerating the supernet training by using a proxy dataset (ImageNet-100) and early stop (training 90 epochs). These results show the effectiveness and efficiency of the proposed ASE-NAS.

Compared with ASE-NAS, our ASE-NAS+ further improves the top-1 accuracy on ImageNet from 79.9% to 80.2%. Note that our ASE-NAS+ outperform all of the considered manually-designed and automatically searched architectures. The reason is that ASE-NAS searches for the promising subspace from randomly initialized candidates and needs to improve the candidate subspaces gradually. Instead, ASE-NAS+ directly employs the promising subspaces centered on a set of well-designed architectures at the beginning of the search. In this case, the architectures in such initialized subspaces have better performance than that in randomly initialized subspaces, which accelerates the search process by providing good subspace initialization centered on the good architectures. The results demonstrate that we are able to apply our method to existing designed architectures/subspaces to further enhance the search performance.

Visualization of Searched Architectures. We show the visualization results of ASE-NAS (top-1 accuracy 79.9%) and ASE-NAS+ (top-1 accuracy 80.2%) in MobileNet-like search space in Figure 4 and Figure 5, respectively. From the visualization results, we found that the number of conv 5×5 and conv 7×7 is much larger than the number of conv 3×3. The reason may have two aspects. First, conv 5×5 and conv 7×7 have a larger reception field and larger model capacity with more parameters than conv 3×3 [61]. Second, larger conv operations may help the network preserve more information when doing downsampling (mentioned by ProxylessNAS [55]). In this case, architectures with conv 5×5 and conv 7×7 may often be beneficial to achieve better performance than that with conv 3×3. Similar results of "large convs are dominated" can also be found in [19], [55].

Fig. 4: The architecture searched by ASE-NAS in MobileNet-like search space.

Fig. 5: The architecture searched by ASE-NAS+ in MobileNet-like search space.

V. More Discussions and Ablations

In this section, we perform more experiments to demonstrate the effectiveness of the proposed architecture distance (Section V-A), the proposed subspace updating scheme (Section V-B), the proposed performance improvement reward (Section V-C), and the subspace graph (Section V-D). In addition, we conduct ablations to investigate the effect of the number of candidate subspaces K and the local search spaces in Sections V-E and V-F, respectively.

A. More Discussions on Architecture Distance

As mentioned before, our method is built upon an underlying hypothesis that the neighborhood around a good architecture is usually a promising subspace. In other words, the architectures in the neighborhood/subspace are more likely to have good performance. In this sense, once we find a promising subspace centered on a good architecture, it would be much easier to find better architectures via a local search. To build such subspace around a given architecture, we devise an *Architecture Distance* $D(\cdot;\cdot)$ to measure the distance between two architectures. In the following, We provide empirical results in NAS-Bench-201 and MobileNet-like search spaces to demonstrate that the devised distance function is able to support the hypothesis. We conduct experiments in NAS-Bench-201 and MobileNet-like search space by computing the performance difference (measured in accuracy) between two architectures with different distances.

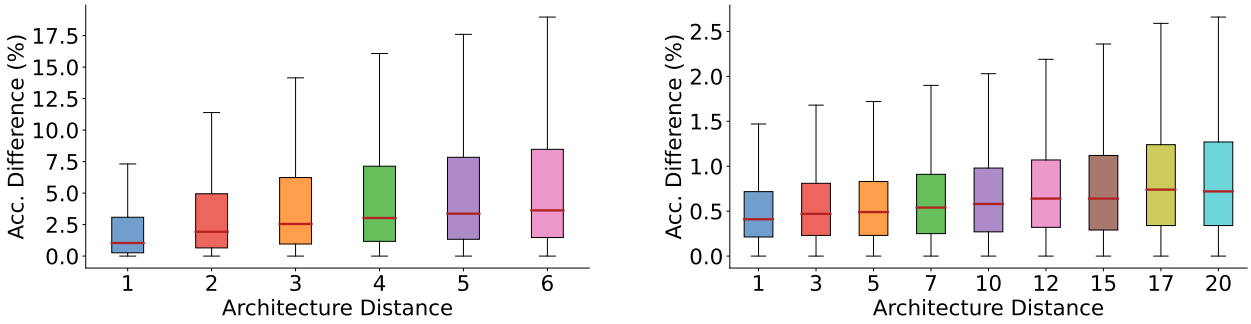
We show the results over 100 different trials in Figure 6. From the results, the average accuracy difference between two architectures becomes larger when their distance increases. For example, in NAS-Bench-201 space, the accuracy difference increases from 4.49 to 6.69 when the distance grows from 1 to 2. Meanwhile, the variance of the performance difference also increases as the corresponding distance grows (*e.g.*, increasing from 6.1 to 9.5 when the distance grows from 1 to 2). The results demonstrate that the rationality and effectiveness of the designed architecture distance, *i.e.*, the architectures with smaller distances (in the same subspace) tend to have similar performance. Thus, we are able to find promising architectures in a subspace around existing good architectures more easily than the overall search space. Similar observations are also found in NAS-Bench-101 [18] search space.

B. Effect of Subspace Updating Scheme

In the global search, if we keep all candidate subspaces fixed, the controller may get stuck in a local optimum due to the very limited search spaces, which would lead to poor search performance. To address this issue, we propose a simple strategy to gradually update the candidate subspaces with the locally searched architectures. To be specific, we replace the center architecture in the evoked subspace with locally searched architecture if has better performance than . This simple updating strategy ensures the candidate subspaces would become more and more promising, which helps to find good architecture within the gradually improved subspaces during the local search.

To verify the effectiveness of the proposed subspace updating scheme, we conduct more experiments on NAS-Bench-201 and compare our ASE-NAS with three baselines and variants, namely *Random Search in Overall Search Space*, *Random Search in Evoked Search Space* and *ASE-NAS without Subspace Update*. The first two baselines conduct a random search in the entire search space and the evoked subspace in the final search step of our method, respectively. The variant *ASE-NAS without Subspace Update* uses the same settings as our method but performs a search without updating the candidate subspaces.

From the results in Figure 7, random searching in evoked subspace (orange bar) has higher average accuracy and lower variances than that in overall search space (purple bar), *i.e.*, 91.89 ± 1.53 *vs.* 87.37 ± 9.71, which demonstrate the effectiveness of the proposed subspace updating scheme. Moreover, our ASE-NAS with subspace (red bar) updating outperforms ASE-NAS without that (blue bar) on three considered datasets. The results indicate that our ASE-NAS constantly finds promising subspaces during the search, which also shows the effectiveness of the subspace updating scheme. In addition, compared with the baseline that searches randomly in the



(a) Comparisons in NAS-Bench-201 search space. (b) Comparisons in MobileNet-like search space.

Fig. 6: The performance difference that is measured by accuracy (%) *vs.* the architecture distance in NAS-Bench-201 search space (a) and MobileNet-like search space (b).

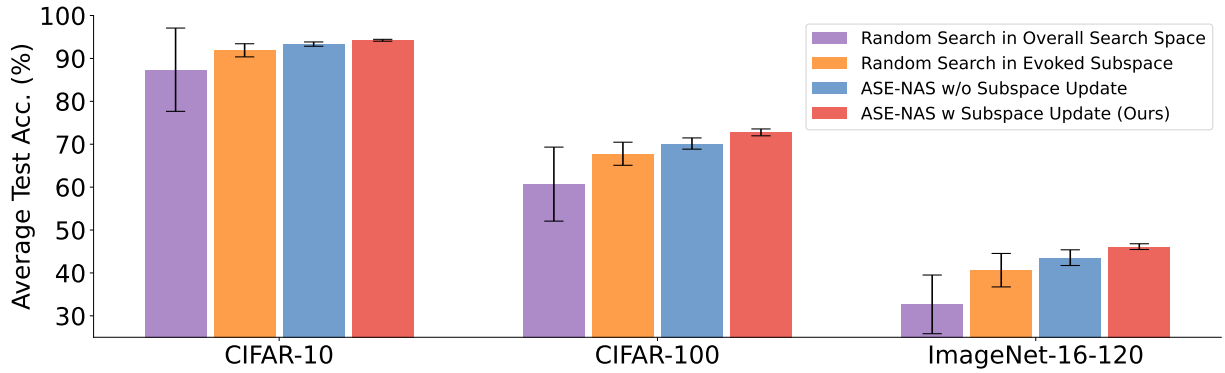


Fig. 7: Comparisons of the search performance with/without subspace updating on NAS-Bench-201.

evoked subspace, our ASE-NAS method consistently achieves better searched accuracy. The results show that the local search scheme is able to further enhance the search performance by finding promising architectures within the evoked subspace.

C. Effect of the Performance Improvement Reward

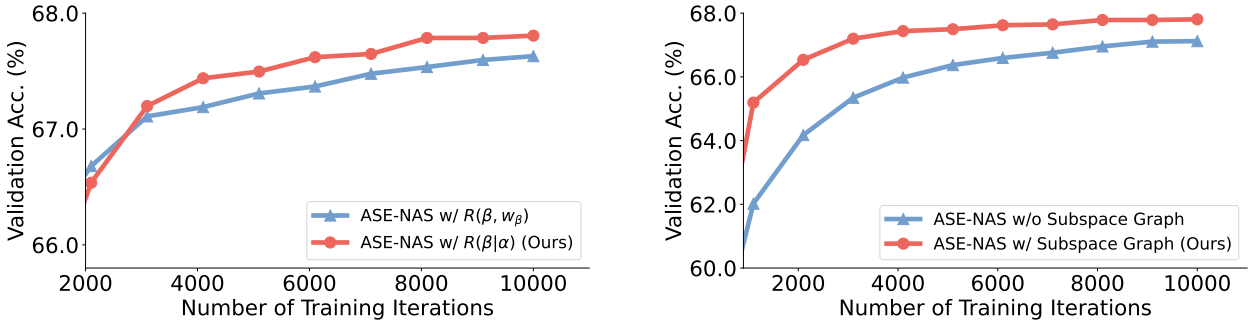
In the policy learning, we use the performance improvement between the resultant and the center architecture in Ω as the reward. Besides this, one can learn the policy by considering the performance of as the reward. However, the policy may easily get stuck in a local optimum and always select the same subspace with the highest performing center architecture. To verify this, we conduct more experiments in the MobileNet-like search space on ImageNet to compare the search performance with these two kinds of reward functions.

We show the averaged validation accuracy (obtained by the supernet) over 10 different runs in Figure 8(a). The variant using the performance improvement (red line) achieves higher search performance than that using the performance of resultant architecture (blue line) (67.81$\pm$0.18 *vs.* 67.63$\pm$0.31). In addition, maximizing the performance improvement finds more new subspaces than using the absolute performance during the search process (319 *vs.* 284). The reason is that if we directly maximize the performance of the resultant architecture , the search algorithm may always select the same subspace with the best architecture. In this case, the remaining subspaces are ignored, which results in poor explorations during search. In contrast, using the performance improvement encourages explorations among different subspaces.

D. Effect of Subspace Graph

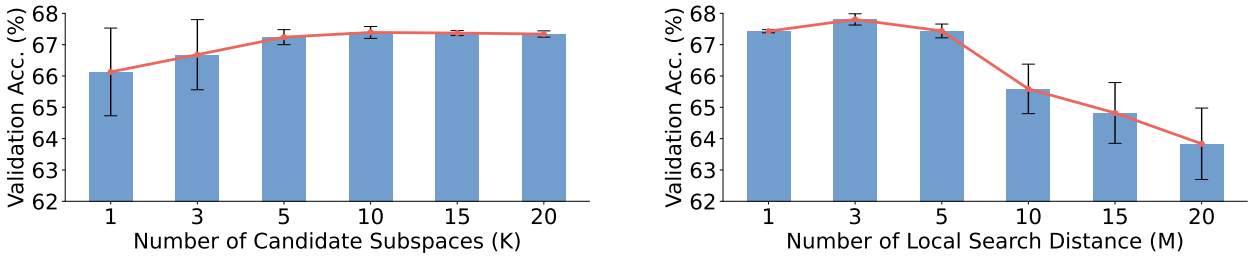
We build the subspace graph with a set of candidate subspaces. In the graph, nodes denote candidate subspaces and direct edges denote the relationships among them. In practice, we represent direct edges as a modification vector that how to modify the center architecture in the weak subspace to that in the better subspace. These edges take helpful information for the local search since they are good examples to represent how to modify an architecture to a better one for the local policy. In addition, the architectures in different subspaces may have different computational operations/topologies and different performances. In this case, the graph structure in the subspace graph may convey beneficial information to select a promising subspace in the global search.

We investigate the effect subspace graph by performing more experiments in MobileNet-like search space with/without the subspace graph structure. Specifically, for the variant without the subspace graph structure, we treat the candidate subspaces as separate points and extract the features from them using two fully-connected layers instead of the two-layer graph neural network. We show the results in Figure 8(b). We report the averaged validation accuracy (obtained by the



(a) Comparisons with different reward functions. (b) Comparisons with/without subspace graph.

Fig. 8: Comparisons of the search performance with different reward functions(a) and with/without subspace graph(b) in MobileNet-like search space on ImageNet.



(a) Ablation on #candidate subspaces K. (b) Ablation on the local search distance M.

Fig. 9: Comparisons of the search performance with the different numbers of candidate subspaces K (a) and the different local search distance M (b) on ImageNet.

supernet) over 10 different runs. From the results, ASE-NAS with subspace graph (red line) has not only higher validation accuracy but also lower variance than ASE-NAS without that (blue line) (67.81±0.18 *vs.* 67.12±0.46). The results demonstrate the effectiveness of the subspace graph.

E. Effect of the Number of Candidate Subspaces K

We build the subspace graph with K architectures and thus have K candidate subspaces $\{\Omega_i\}_{i=1}^{K}$. When we consider a small K, the information carried by the subspaces would be limited, resulting in poor search performance. In contrast, a larger K means more explorations in candidate subspaces. Nevertheless, too many candidate subspaces would introduce a heavy computational burden since the computational cost of the GNN increase squarely as the K becomes larger. To investigate the effect of K, we conduct an ablation study with different K on ImageNet. For fair comparisons, we set the number of each subspace updates in the graph to the same.

In Figure 9(a), our ASE-NAS achieves the worst validation accuracy when $K=1$ since it only explores a single subspace during the search process, which greatly depends on the initialized architecture. As K becomes larger, our ASE-NAS achieves better validation accuracy. The reason is that more architectures benefit the search by exploring more diverse promising subspaces. In this case, our ASE-NAS has a larger probability to find promising architectures. Besides, when K is larger than 10, our ASE-NAS yields very similar search performance. These results demonstrate that using 10 different subspaces is sufficient to achieve competitive performance. Thus, we set K to 10 on ImageNet.

F. Effect of the Local Search Distance M

When performing the local search in Ω , we use a hyper-parameter M to restrict the size of the local search subspace. A smaller search distance M means that we perform the local search in a smaller subspace. Note that searching in small but effective subspaces is exactly our core idea to enhance both search performance and efficiency. In contrast, a larger M enables ASE-NAS to explore more architectures in the search space but makes it harder to explore the whole search space. Note that the maximum of M equals to the number of components L in the architecture, *i.e.*, $M=L$. To investigate the effect of M, we conduct experiments with more different search distance $M \in \{1; 3; 5; 10; 20\}$ in MobileNet-like search space ($L = 20$ in this space).

In Figure 9(b), our ASE-NAS achieves the best validation accuracy when $M=3$ and the worst validation accuracy when $M=20$. When M is too small (*e.g.*, $M=1$), it is easy to fall into the local optimum and hard to find better architectures in the subspace, resulting in poor search results. When M becomes larger (*e.g.*, $M>3$), the large search space lowers the search efficiency and makes it difficult to find good architec-

tures. In this case, the search performance drops greatly (*e.g.*, only 63.84% when M=20). Thus, we set the search distance M to 3 in the MobileNet-like search space.

VI. Conclusion

In this paper, we have proposed a Neural Architecture Search method via Automatic Subspace Evoking (ASE-NAS), which focuses on automatically searched small and effective subspaces and conduct search in them. Specifically, we first perform global search for a promising subspace from the candidate subspaces. Then, we perform local search for effective architectures in the globally searched subspace instead of the original large one. Finally, we update the candidate subspace with the locally searched architecture. Moreover, our ASE-NAS is able to further enhance search performance by taking well designed/searched architectures as the prior knowledge. Extensive experiments demonstrate the superiority of our method over the considered methods.

Appendix
More Details on Graph Neural Network

In the global search, we employ a two-layer graph neural network (GNN) [40] to extract the features of the subspace graph. Specifically, we first provide more details about how to represent the subspace graph as a set of node embeddings and edge embeddings. Since the node/subspace Ω is uniquely determined by its centered architecture , we use the embedding of the architecture to represent the subspace Ω . Specifically, for the architecture , we adopt the concatenation of the learnable vector of each item $^{(i)}$ in it to represent the embedding $\mathbf{h}$. For each edge, we represent its embedding $\mathbf{e}$ $'$ by $\mathbf{h}$ $' - \mathbf{h}$. Since the edge in the subspace graph implies how to modify an architecture to obtain another, *e.g.*, replacing convolution with max pooling in some layer. Given two center architectures, if the components are the same in some positions, the corresponding part of edge features will be zero. In this case, the edges explicitly capture the information on how to modify one to another.

To extract the features from the subspace graph, we extend GraphSAGE [62] by introducing *edge embeddings* following [40]. Specifically, at each GNN layer l, the message passing function takes the concatenation of the node embedding and the edge embeddings in the previous layer as the input:

$$\mathbf{n}^{(l)} = \textsc{Agg} \quad (\mathbf{P}^{(l)} \cdot \textsc{Concat}(\mathbf{h}^{(l\ \ 1)}; \mathbf{e}^{(l\ \ ,1)}; \mathbf{e}^{(l,\ 1)}) \quad | \forall\ ^{0} \in \mathcal{N}(\ \ )) \tag{5}$$

where Agg is the mean pooling function for aggregation, $(\cdot)$ is the non-linearity function (*e.g.*, the Rectified Linear Unit (ReLU) [63]), $\mathbf{P}^{(l)}$ is the trainable weight, $\mathcal{N}(\cdot)$ is the node neighborhood function. Note that the edge embedding would be a zero vector if the corresponding edge does not exist. Then, we update the node embedding $\mathbf{h}^{(l\ \ 1)}$ by:

$$\mathbf{h}^{(l)} = \quad (\mathbf{Q}^{(l)} \cdot \textsc{Concat}(\mathbf{h}^{(l\ \ 1)}; \mathbf{n}^{(l)})) \tag{6}$$

where $\mathbf{Q}^{(l)}$ is the trainable weight. In addition, we update the edge embedding $\mathbf{e}^{(l\ \ ,1)}$ by:

$$\mathbf{e}^{(l)}{}_{,} = \quad (\mathbf{W}^{(l)} \cdot \textsc{Concat}(\mathbf{e}^{(l\ \ ,1)}; \mathbf{h}^{(l)}; \mathbf{h}^{(l)}_{,})) \tag{7}$$

where $\mathbf{W}^{(l)}$ is the trainable weight. With the extracted node features $\{\mathbf{h}^{(l)}_{i}\}^{K}_{i=1}$, we feed them into the LSTM model that samples decisions via softmax classifiers to perform a global search for the promising subspace Ω .

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[2] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021.

[3] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *IEEE International Conference on Computer Vision*, 2021.

[4] Z. Liu, H. Hu, Y. Lin, Z. Yao, Z. Xie, Y. Wei, J. Ning, Y. Cao, Z. Zhang, L. Dong, F. Wei, and B. Guo, "Swin transformer v2: Scaling up capacity and resolution," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2022.

[5] H. Duan, Y. Zhao, Y. Xiong, W. Liu, and D. Lin, "Omni-sourced webly-supervised learning for video recognition," in *European Conference on Computer Vision*, 2020, pp. 670–688.

[6] C. Feichtenhofer, H. Fan, J. Malik, and K. He, "Slowfast networks for video recognition," in *IEEE International Conference on Computer Vision*, 2019, pp. 6201–6210.

[7] Z. Tu, H. Li, D. Zhang, J. Dauwels, B. Li, and J. Yuan, "Action-stage emphasized spatiotemporal vlad for video action recognition," *IEEE Transactions on Image Processing*, vol. 28, no. 6, pp. 2799–2812, 2019.

[8] J. Liu, J. Guo, and D. Xu, "Apsnet: Toward adaptive point sampling for efficient 3d action recognition," *IEEE Transactions on Image Processing*, vol. 31, pp. 5287–5302, 2022.

[9] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," in *International Conference on Learning Representations*, 2018.

[10] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, 2020, pp. 1877–1901.

[11] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *International Conference on Learning Representations*, 2017.

[12] H. Liu, K. Simonyan, and Y. Yang, "DARTS: differentiable architecture search," in *International Conference on Learning Representations*, 2019.

[13] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[14] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *International Conference on Machine Learning*, 2018, pp. 4092–4101.

[15] 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 *European Conference on Computer Vision*, 2018, pp. 19–35.

[16] Y. Guo, Y. Chen, Y. Zheng, P. Zhao, J. Chen, J. Huang, and M. Tan, "Breaking the curse of space explosion: Towards efficient nas with curriculum search," in *International Conference on Machine Learning*, 2020, pp. 3822–3831.

[17] L. Wang, Y. Zhao, Y. Jinnai, Y. Tian, and R. Fonseca, "Alphax: exploring neural architectures with deep neural networks and monte carlo tree search," *arXiv preprint arXiv:1903.11059*, 2019.

[18] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-Bench-101: Towards reproducible neural architecture search," in *International Conference on Machine Learning*, 2019, pp. 7105–7114.

[19] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations*, 2020.

[20] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *International Conference on Learning Representations*, 2017.

[21] Y. Guo, Y. Zheng, M. Tan, Q. Chen, Z. Li, J. Chen, P. Zhao, and J. Huang, "Towards accurate and compact architectures via neural architecture transformer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[22] Y. Chen, Y. Guo, Q. Chen, M. Li, Y. Wang, W. Zeng, and M. Tan, "Contrastive neural architecture search with neural architecture comparators," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2021, pp. 9502–9511.

[23] Y. Tang, B. Li, M. Liu, B. Chen, Y. Wang, and W. Ouyang, "Autopedestrian: An automatic data augmentation and loss function search scheme for pedestrian detection," *IEEE Transactions on Image Processing*, vol. 30, pp. 8483–8496, 2021.

[24] Z. Lu, G. Sreekumar, E. D. Goodman, W. Banzhaf, K. Deb, and V. N. Boddeti, "Neural architecture transfer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 9, pp. 2971–2989, 2021.

[25] M. Chen, J. Fu, and H. Ling, "One-shot neural ensemble architecture search by diversity-guided search space shrinking," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2021, pp. 16 530–16 539.

[26] M. Chen, H. Peng, J. Fu, and H. Ling, "Autoformer: Searching transformers for visual recognition," in *IEEE International Conference on Computer Vision*, 2021, pp. 12 250–12 260.

[27] X. Dai, A. Wan, P. Zhang, B. Wu, Z. He, Z. Wei, K. Chen, Y. Tian, M. Yu, P. Vajda, and J. E. Gonzalez, "Fbnetv3: Joint architecture-recipe search using predictor pretraining," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2021, pp. 16 276–16 285.

[28] J. Liu, S. Zhou, Y. Wu, K. Chen, W. Ouyang, and D. Xu, "Block proposal neural architecture search," *IEEE Transactions on Image Processing*, vol. 30, pp. 15–25, 2021.

[29] X. Chen, R. Wang, M. Cheng, X. Tang, and C. Hsieh, "Drnas: Dirichlet neural architecture search," in *International Conference on Learning Representations*, 2021.

[30] Z. Xia, W. Peng, H.-Q. Khor, X. Feng, and G. Zhao, "Revealing the invisible with model and data shrinking for composite-database micro-expression recognition," *IEEE Transactions on Image Processing*, vol. 29, pp. 8590–8605, 2020.

[31] K. Nguyen, C. Fookes, and S. Sridharan, "Constrained design of deep iris networks," *IEEE Transactions on Image Processing*, vol. 29, pp. 7166–7175, 2020.

[32] X. Yang, S. Wang, J. Dong, J. Dong, M. Wang, and T.-S. Chua, "Video moment retrieval with cross-modal neural architecture search," *IEEE Transactions on Image Processing*, vol. 31, pp. 1204–1216, 2022.

[33] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *AAAI Conference on Artificial Intelligence*, 2019, pp. 4780–4789.

[34] D. Zhou, X. Zhou, W. Zhang, C. C. Loy, S. Yi, X. Zhang, and W. Ouyang, "Econas: Finding proxies for economical neural architecture search," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 393–11 401.

[35] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8697–8710.

[36] C. White, S. Nolen, and Y. Savani, "Exploring the loss landscape in neural architecture search," in *Proceedings of Conference on Uncertainty in Artificial Intelligence*, vol. 161, 2021, pp. 654–664.

[37] X. Li, C. Lin, C. Li, M. Sun, W. Wu, J. Yan, and W. Ouyang, "Improving one-shot NAS by suppressing the posterior fading," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 833–13 842.

[38] L. Wang, S. Xie, T. Li, R. Fonseca, and Y. Tian, "Sample-efficient neural architecture search by learning action space," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 5503–5515, 2022.

[39] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

[40] J. You, X. Ma, D. Y. Ding, M. J. Kochenderfer, and J. Leskovec, "Handling missing data with graph representation learning," in *Advances in Neural Information Processing Systems*, 2020, pp. 19 075–19 087.

[41] X. Dong and Y. Yang, "NAS-Bench-201: Extending the scope of reproducible neural architecture search," in *International Conference on Learning Representations*, 2020.

[42] X. Dong and Y. Yang, "One-shot neural architecture search via self-evaluated template network," in *IEEE International Conference on Computer Vision*, 2019, pp. 3680–3689.

[43] X. Dong and Y. Yang, "Searching for a robust neural architecture in four GPU hourshs," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1761–1770.

[44] S. Hu, S. Xie, H. Zheng, C. Liu, J. Shi, X. Liu, and D. Lin, "DSNAS: direct neural architecture search without parameter retraining," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 12 081–12 089.

[45] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, and H. Xiong, "PC-DARTS: Partial channel connections for memory-efficient architecture search," in *International Conference on Learning Representations*, 2020.

[46] A. Howard, R. Pang, H. Adam, Q. V. Le, M. Sandler, B. Chen, W. Wang, L. Chen, M. Tan, G. Chu, V. Vasudevan, and Y. Zhu, "Searching for mobilenetv3," in *IEEE International Conference on Computer Vision*, 2019, pp. 1314–1324.

[47] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li, "Imagenet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[48] R. Luo, X. Tan, R. Wang, T. Qin, E. Chen, and T. Liu, "Semi-supervised neural architecture search," in *Advances in Neural Information Processing Systems*, 2020, pp. 10 547–10 557.

[49] S. Yan, Y. Zheng, W. Ao, X. Zeng, and M. Zhang, "Does unsupervised architecture representation learning help neural architecture search?" in *Advances in Neural Information Processing Systems*, 2020, pp. 12 486–12 498.

[50] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[51] N. Ma, X. Zhang, H. Zheng, and J. Sun, "Shufflenet V2: practical guidelines for efficient CNN architecture design," in *European Conference on Computer Vision*, 2018, pp. 122–138.

[52] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in *IEEE International Conference on Computer Vision*, 2019, pp. 1294–1303.

[53] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 734–10 742.

[54] J. Mei, Y. Li, X. Lian, X. Jin, L. Yang, A. Yuille, and J. Yang, "Atomnas: Fine-grained end-to-end neural architecture search," in *International Conference on Learning Representations*, 2020.

[55] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," in *International Conference on Learning Representations*, 2019.

[56] X. Chu, B. Zhang, and R. Xu, "Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search," in *IEEE International Conference on Computer Vision*, 2021, pp. 12 219–12 228.

[57] C. Li, J. Peng, L. Yuan, G. Wang, X. Liang, L. Lin, and X. Chang, "Block-wisely supervised neural architecture search with knowledge distillation," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1989–1998.

[58] A. Wan, X. Dai, P. Zhang, Z. He, Y. Tian, S. Xie, B. Wu, M. Yu, T. Xu, K. Chen, P. Vajda, and J. E. Gonzalez, "Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 12 962–12 971.

[59] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*, 2019, pp. 6105–6114.

[60] H. Peng, H. Du, H. Yu, Q. Li, J. Liao, and J. Fu, "Cream of the crop: Distilling prioritized paths for one-shot neural architecture search," in *Advances in Neural Information Processing Systems*, 2020, pp. 17 955–17 964.

[61] M. Tan and Q. V. Le, "Mixconv: Mixed depthwise convolutional kernels," in *British Machine Vision Conference*, 2019, p. 74.

[62] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, Eds., 2017, pp. 1024–1034.

[63] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *International Conference on Machine Learning*, 2010, pp. 807–814.