Automated Dominative Subspace Mining for Efficient Neural Architecture Search

Yaofo Chen, Yong Guo, Daihai Liao, Fanbing Lv, Hengjie Song,
James Tin-Yau Kwok, *IEEE Fellow*, and Mingkui Tan, *IEEE Member*

***Abstract*—Neural Architecture Search (NAS) aims to automatically find effective architectures within 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 Dominative Subspace Mining (DSM-NAS) that finds promising architectures in automatically mined subspaces. Specifically, we first perform a global search, *i.e.*, dominative subspace mining, to find a good subspace from a set of candidates. Then, we perform a local search within the mined subspace to find effective architectures. More critically, we further boost search performance by taking well-designed/searched architectures to initialize candidate subspaces. Experimental results demonstrate that DSM-NAS not only reduces the search cost but also discovers better architectures than state-of-the-art methods in various benchmark search spaces.**

***Index Terms*—Neural Architecture Search, Search Space Mining, 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]–[3], action recognition [4], [5] and natural language processing [6]–[9]. 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) [10] is developed to automate the process of the architecture design.

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

Yong Guo is with the Max Planck Institute for Informatics, Max Planck Institute, Saarbrücken 66123, Germany. (e-mail: guoyongcs@gmail.com).

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

James Tin-Yau Kwok is with the Department of Computer Science and Engineering, the Hong Kong University of Science and Technology, Hong Kong 999077, China (e-mail: jamesk@cse.ust.hk)

This work was partially supported by National Natural Science Foundation of China (NSFC) 62072190, the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. 16202523 and HKU C7004-22G), the Major Key Project of Peng Cheng Laboratory (PCL) PCL2023A08, the Young Scholar Project of Pazhou Lab (No.PZL2021KF0021), TCL Science and Technology Innovation Fund.

Yaofo Chen and Yong Guo contributed equally to this paper.

Mingkui Tan is the corresponding author.

Existing NAS methods search for effective architectures in a predefined search space [10]–[12]. 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 [13] and $\sim 10^{19}$ in OFA [14]). Directly searching in such a large space is very difficult and time-consuming in practice [15]. 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 [16] and CNAS [15] 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 Figure 6, as well as the observations in [15], [18]). Inspired by this, instead of searching in the whole search space, we seek to limit the search space to a reduced one in each search step by recognizing and mining a small but effective subspace. In this sense, we may boost the search performance and search efficiency as well since we only focus on a mined 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 Dominative Subspace Mining (called

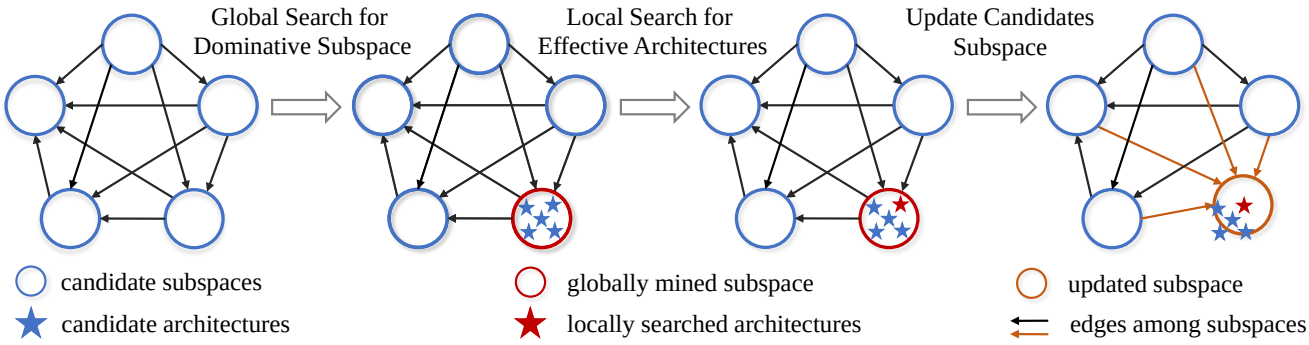


Fig. 1: An illustration of the search process. We find promising architectures in a two-step search manner: 1) we perform global search to mine/find a dominative subspace from a set of candidates; 2) we move the focus to the subspace and conduct a local search for effective architectures within it. Then,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

DSM-NAS). The key idea is to find/mine a small but effective/dominative 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 from them, with edges denoting the relationships/information among different subspaces. As shown in Figure 1, we first perform a *global search* to automatically mine a dominative subspace from the subspace graph. Then, we focus on the mined subspace and conduct a *local search* to obtain the resultant architectures. It is worth noting that once we find a better architecture, we also update the subspace graph accordingly. In this way, it becomes possible to gradually find better architectures during the search process. Moreover, we are able to further boost search performance of the proposed DSM-NAS by taking existing well-designed architectures (*e.g.*, OFA architecture [14]) to construct candidate subspaces. Extensive experiments in two benchmark search spaces demonstrate the effectiveness of the proposed method.

Our contributions are summarized as follows:

- Instead of searching in the entire search space, we seek to find/mine a small but effective/dominant subspace for each step in the architecture search. With the help of the mined subspaces, we are able to improve search performance and efficiency.
- We propose a novel Dominative Subspace Mining algorithm to enhance the performance of neural architecture search. Specifically, we first perform a global search to find dominative 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 importantly, the searched subspaces also exhibit promising transferability to new datasets.

II. Related Work

A. Neural Architecture Search

In recent years, NAS [19]–[21] has drawn great attention for effective architecture design. Traditional reinforcement learning-based NAS methods [22]–[26] directly maximizes the expectation of the performance of searched architectures sampled from the whole search space. In contrast, our DSM-NAS searches in a small and effective subspace instead of the whole space by employing both global search and local search policies. In this sense, DSM-NAS alleviates the difficulties due to a large search space. Gradient-based NAS methods [11], [27]–[31] find effective architectures by relaxing the search space to continuous-valued and optimizing by gradient descent. They introduce differentiable architecture parameters, in which the network weights and architecture parameters are alternately optimized. Our proposed method focuses on searching in small and effective subspaces instead of the entire large space. This is a significant difference from gradient-based methods, which typically operate in a continuous relaxation of the whole search space.

Besides, Evolutionary-based NAS methods [32]–[36] find promising architectures through crossover and mutation in the neighborhood/subspace of the population. Our DSM-NAS differs from these methods in three significant ways. First, they usually perform crossover and mutation randomly. Our DSM-NAS searches architectures in the mined subspace with a learned policy, which results in higher search efficiency. Second, they usually greedily select the best architecture from the population for reproduction. In contrast, our DSM-NAS finds a promising subspace that has the potential to achieve the largest performance improvement, which helps to encourage the exploration ability (See Figure 8a). Third, during mutation, they treat elements in the population independently to find a subspace. Our method builds a subspace graph to exploit the relationship among different subspaces to enhance the search performance (See Figure 8b).

However, traditional NAS methods [22], [37] often require great computational resources and thus result in unaffordable time costs. A lot of efforts have been made to improve the search efficiency of the search process. They mainly focus on improving the efficiency of architecture performance estimation. Specifically, weight-sharing-based NAS methods [13], [14] estimate the performance of candidate architectures without the need to train each one from scratch. Instead, they train a supernet that encompasses all possible architectures under

consideration. While estimating architectures, their weights are inherited from the trained supernet, which greatly lowers the computational cost of architecture evaluation. EcoNAS [38] observes that most existing proxies exhibit different behaviors in maintaining rank consistency among architectures. Based on this, it designs a proxy training strategy that is potentially more accurate in evaluating architectures on small proxy datasets.

Besides, Zero-cost proxy methods (such as Grad-norm [39], SNIP [40], GraSP [41] and Synflow [42]) reduce the computational cost by summing up the saliency value of the model weights. ZiCO [43] devises a new architecture estimation strategy by calculating *Zero-shot inverse Coefficient of Variation* scores with a single forward/backward propagation. These methods are very efficient since they just require a single forward/backward propagation while evaluating an architecture. This is in stark contrast to traditional methods that may require extensive training and multiple iterations to obtain the performance of an architecture. Unlike these methods which focus on rapidly evaluating architectures, our method achieves high efficiency from a different perspective. To be specific, we reduce the number of architecture evaluations required to identify promising architectures. To achieve this, our DSM-NAS focuses on small and effective subspaces that are more likely to contain high-performing architectures. By narrowing down the search to these subspaces, the number of required architecture evaluations decreases. This not only accelerates the search process but also ensures that the computational resources are expended on evaluating architectures that have a higher likelihood of success.

B. Search Space Design of NAS Methods

NAS methods often find promising architectures in a predefined large search space, such as NASNet [44], DARTS [11] and MobileNet-like [12] search spaces. Most existing methods directly perform search 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 [45] search in an iterative manner. They visit architectures in the neighborhood of an architecture and update it with the best-found architecture. This is computationally expensive since it requires plenty of architecture evaluations. Recent works, such as PNAS [16] and CNAS [15], implement a progressive space-growing search strategy. They start from a compact search space to mitigate the challenges posed by a large search space, and subsequently expand the search space by incorporating additional nodes or operations. However, these methods suffer from two limitations. First, in the later stages of the search, these methods still try to identify architectures within an exceedingly large space, potentially leading to insufficient explorations. Secondly, the small search spaces delineated by these methods are arbitrarily created through the addition of nodes or operations and remain static throughout the search process. This may not be optimal for discovering high-performing architectures. In contrast, our DSM-NAS consistently searches in small yet potent subspaces throughout the entire search process and dynamically identifies improving and dominant subspaces.

AlphaX [17] and LaNAS [46] build a Monte Carlo Search Tree to partition the search space into different subspaces according to their performance and encourage exploration of the promising subspaces. Nonetheless, note that the partitioning process is bound by a predetermined tree structure, which imposes certain limitations on the method. During each step of the search, the algorithm is restricted to partitioning based on the available candidate operations. In contrast, our DSM-NAS adopts a more flexible and unconstrained approach to navigating the search space. It focuses on identifying and mining the dominant subspace, which is determined solely based on the central architecture of that subspace. It is important to note that this central architecture is not fixed, but instead can be any configuration in the entire search space, thus providing a broader exploration.

Besides, Few-shot NAS [47] adopts multiple sub-supernets to encompass different regions (*i.e.*, subspaces) of the search space, aiming for a precise evaluation of architectural performance. These subspaces are different from the entire search space at the beginning and remain constant in subsequent iterations. In contrast, our DSM-NAS differs from these methods in two significant ways: 1) DSM-NAS tries to mitigate the difficulties incurred by a large search space rather than enhancing architectural performance estimation; 2) DSM-NAS adopts a dynamic approach, actively searching for the dominant subspaces, instead of relying on static, predetermined subspaces. RegNet [48] designs an initial search space that is very large, and then subsequently narrows it down to a more refined and compact search space through empirical analysis of the architectural behaviors. It is imperative to highlight that the entire pipeline operates manually. In contrast, our DSM-NAS automatically explores dominative search spaces and identifies promising architectures without the need for human intervention, thanks to our innovative global and local search strategies.

III. Architecture Search via Subspace Mining

In this paper, we propose a neural architecture search method with Dominative Subspace Mining (DSM-NAS) to boost both the search performance and efficiency of NAS. In Section III-A, we first discuss the motivation and provide an overview of DSM-NAS. Then, we describe the details of the two key steps in our method, *i.e.*, the global search and local search in Sections III-B and III-C, respectively. In Table I, we present some frequently used notations.

A. Motivation and Method Overview

Existing NAS methods often consider an extremely large search space Ω to find good architectures [11], [13], [14]. However, directly performing architecture search in such a large search space is non-trivial and often very expensive. Instead of using the whole search space, it should be possible to limit each search step within a reduced search space to boost search performance and search efficiency. To achieve this goal, an underlying hypothesis is that the neighborhood around an effective architecture is usually a promising/dominative subspace for further exploration to find better ones [15], [18].

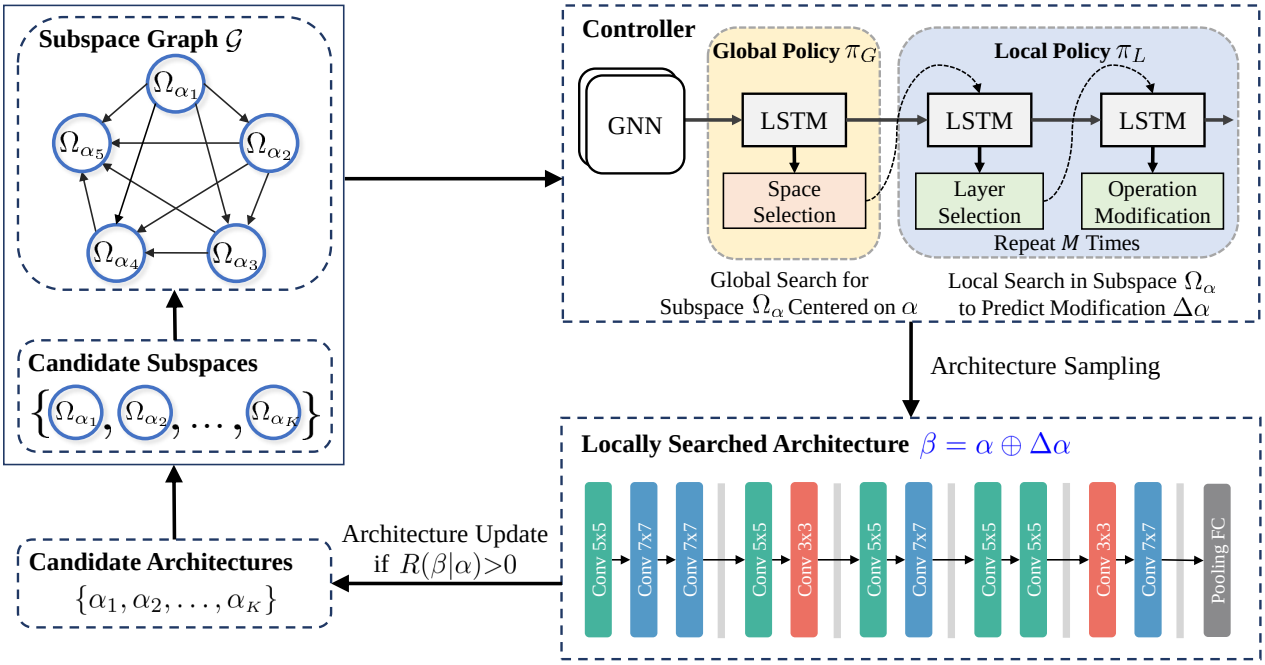


Fig. 2: An overview of the proposed DSM-NAS. We build a set of subspaces $\{\Omega_{\alpha_i}\}_{i=1}^K$ centered on randomly sampled candidate architectures $\{\alpha_i\}_{i=1}^K$ and construct a subspace graph $\mathcal{G}$ to model the relationships among these subspaces. By taking $\mathcal{G}$ as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

TABLE I: Summary of frequently used notations.

Notation	Description
Ω	search space
α	architecture
Ω_α	search subspace centered on an architecture α
$R(\alpha, w_\alpha)$	performance metric of an architecture α
$R(\beta\|\alpha)$	performance improvement between β and α
π_G	global search policy
π_L	local search policy
$\mathcal{G}$	subspace graph
K	number of candidate subspaces in $\mathcal{G}$
M	local search distance
$\Delta\alpha$	architecture modification
$\oplus$	combination operation

In this case, we are more likely to find effective architectures in the dominative subspace than the whole space under the same search budget.

Inspired by this, we propose a new search algorithm by mining/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_α 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 automatically search for a dominative subspace Ω_α based on a center architecture α. Then, we further learn a local search policy π_L to produce the resultant architectures in the subspace. Specifically, the global search policy π_G takes a set of candidate subspaces $\{\Omega_{\alpha_i}\}_{i=1}^K$ (as well as their relationship) as inputs and mines/finds a dominative subspace Ω_α. Based on the mined/searched subspace, the local policy π_L further generates a modification $\Delta\alpha$ (*i.e.*, modifying some operations of some layers in α) to explore the subspace. Finally, we combine α and $\Delta\alpha$ to obtain the resultant architecture by

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

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

Note 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 Figure 8a). To avoid this, we encourage the exploration ability by maximizing the performance improvement between β and the center architecture α in Ω_α, *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:

$$\max_{\pi_G, \pi_L} \mathbb{E}_{\alpha\sim\pi_G}\left[\mathbb{E}_{\Delta\alpha\sim\pi_L} R(\beta|\alpha)\right], \quad \text{s.t. } \beta = \alpha \oplus \Delta\alpha, \tag{2}$$

Algorithm 1 Training method for DSM-NAS.

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

1: Train the parameters of the supernet.
2: Randomly sample architectures $\{\alpha_i\}_{i=1}^K$ from Ω to build the subspaces $\{\Omega_{\alpha_i}\}_{i=1}^K$ using Eqn. (3).
3: Construct the subspace graph $\mathcal{G}$ based on $\{\Omega_{\alpha_i}\}_{i=1}^K$.
4: **while** not convergent **do**
5: *// Perform global search to mine dominative subspace Ω_α*
6: Sample a subspace $\Omega_\alpha \sim \pi_G(\cdot|\mathcal{G};\theta_G)$ centered on α.
7: *// Perform local search in the mined subspace Ω_α*
8: Sample modifications $\Delta\alpha \sim \pi_L(\cdot|\Omega_\alpha;\theta_L)$.
9: Build a resultant architecture $\beta = \alpha \oplus \Delta\alpha$.
10: Compute reward $R(\beta|\alpha)$=$R(\beta, w_\beta)$−$R(\alpha, w_\alpha)$ using the weights inherited from the supernet.
11: *// Update subspaces with the locally searched architecture β*
12: **if** $R(\beta|\alpha) > 0$ **then**
13: Replace the candidate subspace Ω_α with Ω_β.
14: Update the edges connected to Ω_β in $\mathcal{G}$.
15: **end if**
16: Update the parameters θ_G and θ_L by optimizing Eqn. (2) using policy gradient [49].
17: **end while**

Since we would update the searched subspace with the resultant architecture β (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 Dominative Subspace Mining

As the first step of DSM-NAS, we seek to mine dominative subspaces via a global search process. Specifically, we first discuss the construction of candidate subspaces. Then, we describe 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 $\{\alpha_i\}_{i=1}^K$ from the search space and build the candidate subspaces $\{\Omega_{\alpha_i}\}_{i=1}^K$ around them. Let $D(\alpha, \beta)$ be a function to measure the *Architecture Distance* between two architectures α and β. Specifically, we take an architecture α as the center of the subspace Ω_α and constrain all architectures β in Ω_α to have distances less than a specific threshold M from the center architecture α, *i.e.*, $D(\alpha, \beta) \leq M$. Formally, we construct the subspace Ω_α w.r.t. a center architecture α as:

$$\Omega_\alpha = \{\beta \mid D(\alpha, \beta) \leq M, \beta \in \Omega\}. \tag{3}$$

Note that architectures in different subspaces may have different operations/topologies and different performances. 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 Ω_{α_i}. $\mathcal{E}$ is a set of directed edges from a weak subspace (with a poor center architecture) to a better subspace (whose center architecture has higher accuracy). Note that even when initializing the center architectures $\{\Omega_{\alpha_i}\}_{i=1}^K$ randomly, our proposed DSM-NAS is able to achieve good search performance. We may further improve DSM-NAS by taking existing well-designed/searched architectures to construct the subspace (See experimental results in Tables II and IV).

In the subspace graph $\mathcal{G}$, since the node/subspace Ω_α is uniquely determined by its centered architecture α, we use the embedding of the architecture α to represent the subspace Ω_α. Specifically, for architecture α, we use the concatenation of the learnable vector of each component in it to represent the corresponding embedding $\mathbf{h}_\alpha$. For each edge, we represent its embedding $\mathbf{e}_{\alpha\alpha'}$ by $\mathbf{h}_{\alpha'} - \mathbf{h}_\alpha$. 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 in the edge features will be zero. In this case, the edges explicitly capture the information on how to modify one architecture to another, serving as a rich source of information for the local search and providing exemplary guidance on how to enhance an architecture via modification. To verify the significance of the subspace graph, we conduct ablation studies in Section V-D. Empirical evidence clearly demonstrates that DSM-NAS with the subspace graph significantly outperforms its counterpart without this feature.

Searching for Dominative Subspaces. In each search step, we conduct a global search to automatically mine/select a dominative subspace Ω_α from all candidate subspaces $\{\Omega_{\alpha_i}\}_{i=1}^K$. As mentioned before, we seek to find Ω_α that has the potential to achieve a large performance improvement $R(\beta|\alpha)$. (Eqn. (2)). Here, α is the center architecture of Ω_α and β is another architecture in this subspace. To this end, we devise a controller model that contains a two-layer graph neural network (GNN) [50] and an LSTM network. Specifically, to exploit the information of the subspace graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we first employ the GNN to extract features from $\mathcal{G}$. Then, we feed the extracted features into the LSTM that samples a candidate subspace Ω_α via a classifier.

Note that the search performance greatly depends on the 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 $\{\Omega_{\alpha_i}\}_{i=1}^K$ using the newly searched architectures. Specifically, for a selected subspace Ω_α (See Figure 2 and lines 12-15 in Algoritm 1), we replace its center architecture α with the locally searched architecture β if β yields better performance than α. This follows the hypothesis that the subspace around a better architecture may be more likely to contain promising architectures. Once the center architecture is updated, the corresponding subspace Ω_α is also updated to Ω_β. Then, we also update the subspace graph $\mathcal{G}$ by updating all the edges that are originally connected to Ω_α. In this way, the candidate subspaces constantly improve, which helps to explore more and more promising spaces. After the search, we select the best center architecture in the subspace graph as the inferred architecture according to the validation performance.

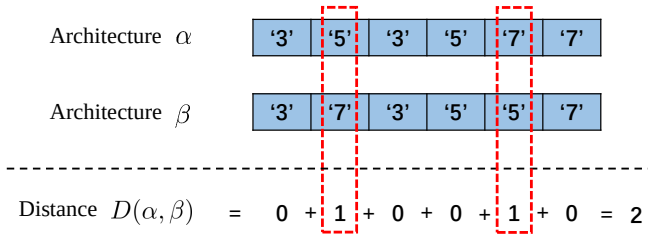


Fig. 3: An illustration of the architecture representation method and calculation of the architecture distance. We represent architecture as a string, in which each item denotes an operation (*e.g.*, convolution). For example, '3', '5' and '7' denote 3×3, 5×5 and 7×7 convolution, respectively.

C. Local Search in the Mined Subspace

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

Before defining the *Architecture Distance* between two architectures, we first revisit the representation of architectures, making it easier to understand. Following [13], [14], we represent an architecture as a L-dimensional string $\alpha{=}[\alpha^{(1)}, \alpha^{(2)}, \cdots, \alpha^{(L)}]$, where L is the number of components in the architecture and each $\alpha^{(i)}$ 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). Note that this distance metric is able to calculate the distance between two architecture with different depths. For non-existent layers in the architecture, we use a placeholder value (like "0") to ensure that the strings are of the same length. Let $\mathbb{1}\{\cdot\}$ be the indicator function. Given two architectures α and β, the distance between them is

$$D(\alpha, \beta) := \sum_{i=1}^{L} \mathbb{1}\{\alpha^{(i)} \neq \beta^{(i)}\}. \tag{4}$$

We empirically demonstrate that the proposed distance metric is reasonable and effective when considering all components in the architecture as equal. This empirical relationship between accuracy and architecture distance is shown in Figure 6. We observe that architectures with smaller distances tend to have similar performance, thereby confirming the reliability of the proposed distance metric. Interestingly, a similar phenomenon has also been observed in well-known NAS methods [15], [18]. This consistency strengthens our justification for the effectiveness of the proposed distance metric. Consequently, when we identify a dominant subspace centered around a high-performing architecture, it becomes considerably easier to discover better architectures through local search techniques.

To ensure that the locally searched architecture β belongs to Ω_α, following common practices [13], [15], we adopt an LSTM model with a local policy π_L that modifies the center architecture α M times. Specifically, for each time modification, the local policy π_L determines which layer has to be modified and which kind of operation is to be applied to this layer. Then, by applying $\Delta\alpha$ to α, we obtain the modified/searched architecture $\beta = \alpha \oplus \Delta\alpha$ in the subspace. Such a decision process is as repeated M times to obtain the complete modification $\Delta\alpha$. The modification process can be considered as a sequential decision-making process. This sequential generation process aligns perfectly with the capabilities of LSTM models, which make predictions based on previously determined results. Note that if π_L selects the same operation as the original one in some layers, it will produce no modification to the architecture. Thus, after M times single-layer modification, the resultant architecture β still belongs to Ω_α, *i.e.*, satisfying the constraint $D(\alpha, \beta) \leq M$.

Analysis of 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. Traditional NAS methods directly search in the whole search space, whose size is $|\Omega| = C^L$. In our DSM-NAS, given a local search distance M, the size of the subspace becomes $|\Omega_\alpha| = \binom{L}{M} C^M$, where $\binom{\cdot}{\cdot}$ denotes the combination function. When $M \ll L$, 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 practice, taking MobileNet-like search space as an example, we have $C = 9$, $L = 20$ and $M = 3$. The size of the whole search space is 9^{20}. In contrast, the size of our search subspace is $20!/(3! \times 17!) \times 9^3 = 1140 \times 9^3$, which is much smaller the the size of the whole space. In the extreme case, when we consider $M{=}L$, the subspace is exactly the whole space and our method reduces to the standard NAS. We investigate the effect of M on the performance of DSM-NAS in Section V-F.

IV. Experiments

The experiments are organized as follows. We first perform experiments in NAS-Bench-201 [51] search space to demonstrate the effectiveness of our DSM-NAS. Then, we evaluate our DSM-NAS in MobileNet-like [59] search space and compare the performance of our method with SOTA methods on a large-scale benchmark dataset ImageNet [60]. Our code and the pretrained models are publicly available at https://github.com/chenyaofo/DSM-NAS.

A. Performance Comparisons on NAS-Bench-201

Search Space. We apply our DSM-NAS to a cell-based NAS-Bench-201 search space [51]. 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

TABLE II: Comparisons with existing methods in NAS-Bench-201 [51]. "ImageNet-16-120'" denotes a subset of the ImageNet dataset with 120 classes and 16×16 resolution. "Search Cost" denotes the time cost in the search phase (measured by second). † denotes we implement the baselines with the official code under our same settings. Our DSM-NAS achieves higher accuracy than state-of-the-art methods with less search cost, which verifies 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 [49]†	25k	93.85±0.37	71.71±1.09	45.24±1.18
PNAS [16]†	25k	93.71±0.29	70.89±0.99	44.75±0.80
CNAS [15]†	25k	93.95±0.28	71.73±1.05	45.46±0.97
ENAS [13]	–	53.89±0.58	13.96±2.33	14.84±2.10
DARTS [11]	30k	54.30±0.00	15.61±0.00	16.32±0.00
SETN [52]	34k	87.64±0.00	59.05±0.24	32.52±0.21
GDAS [53]	32k	93.61±0.09	70.70±0.30	41.71±0.98
DSNAS [54]	–	93.08±0.13	31.01±16.38	41.07±0.09
PC-DARTS [55]	–	93.41±0.30	67.48±0.89	41.31±0.22
DARTS- [56]	30k	93.80±0.07	73.02±0.16	46.41±0.14
EZNAS [57]	–	93.63±0.12	69.82±0.16	43.47±0.20
GradSign [58]	–	93.31±0.47	70.33±1.28	42.42±2.81
ZiCO [43]	–	93.50±0.18	70.62±0.26	42.04±0.82
DSM-NAS (Ours)	25k	**94.23±0.22**	72.76±0.80	46.13±0.67
DSM-NAS+ (Ours)	25k	–	**73.12±0.61**	**46.66±0.52**

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 [60] dataset with 120 classes and 16×16 image resolution.

Implementation Details. Following the settings in NAS-Bench-201 [51], 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 [13], we train our DSM-NAS with a batch size of 1 and set the strength of the entropy regularizer to 7.5×10^{-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.

To simplify implementation, we combine global policy π_G and the local policy π_L into a single policy to make decisions to predict the dominative subspace Ω_α and the modification $\Delta\alpha$. In other words, we seek to learn a joint policy that first selects a candidate subspace and then finds a promising architecture modification in the selected subspace. Note that combining the global and local policies together is equivalent to treating them individually. The publicly available source code demonstrates that the implementation effort required for our DSM-NAS is comparable to that of traditional reinforcement learning-based NAS methods.

Comparisons with State-of-the-art Methods. We compare our method with two baselines, namely *Random Search* and *REINFORCE* [49]. Random Search randomly samples architectures and selects the one with the highest accuracy among them as the final architecture. REINFORCE performs search 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 II, compared with the baselines, our DSM-NAS achieves the highest average accuracy on three datasets, *i.e.*, CIFAR-10, CIFAR-100 and ImageNet-16-120. Compared with REINFORCE [49] baseline, the proposed DSM-NAS yields better search accuracy and lower variance (*e.g.*, 72.76±0.80% *vs.*71.71±1.09% on CIFAR-100). In addition, our DSM-NAS consistently outperforms the best competitor DARTS- across all three datasets. This is because our DSM-NAS searches by focusing on the small but effective subspace, which reduces the search difficulty resulting from the large search space. During the search, DSM-NAS updates the candidate subspaces with the locally searched architectures and finds better architectures in the constantly improved subspaces. Besides, our DSM-NAS not only delivers superior performance but also achieves them with greater efficiency. It requires only 25k seconds for the search process, compared to DARTS [11] (30k) and DARTS- [56] (30k), and GDAS [53] (32k). This significant reduction in search time underlines the methodological efficiency of DSM-NAS, which concentrates on mining potent subspaces automatically, rather than combing through the entire search space exhaustively.

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). Instead of initiating the search from scratch with randomly sampled architectures, we can leverage promising architectures from existing datasets to initialize candidate subspaces. This variant of our method, called DSM-NAS+, not only potentially reduces computational

TABLE III: Comparisons with existing methods in NAS-Bench-360 [61]. We report the classification error (%) and mean square error in the NinaPro and Darcy Flow, respectively. In both tasks, smaller metrics represent better performance. For a fair comparison, the search cost of all the methods is limited to 25k seconds. Thus we do not report the search cost.

Method	NinaPro ↓	Darcy Flow ↓
Random Search	8.09±0.71	0.0252±0.006
REINFORCE [49]	8.07±0.73	0.0247±0.006
Evolution [37]	8.15±0.85	0.0244±0.006
BOHB [62]	8.17±0.57	0.0194±0.002
Hyperband [63]	8.16±0.57	0.0191±0.002
ENAS [13]	11.56±1.12	0.253±0.000
DARTS [11]	22.06±2.00	0.150±0.093
RSWS [64]	9.82±1.49	0.221±0.045
GDAS [53]	17.61±6.39	0.180±0.103
SETN [52]	14.56±7.30	0.253±±0.000
EZNAS [57]	7.34±0.32	0.0242±0.022
GradSign [58]	7.65±0.86	0.0228±0.085
ZiCO [43]	7.49±0.25	0.0223±0.043
DSM-NAS (Ours)	**6.76±0.12**	**0.0211±0.030**

costs but also enhances performance, ensuring a more efficient and effective search process. 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. Since CIFAR-10 is the original/existing dataset, conducting a direct evaluation on CIFAR-10 is meaningless. The "-" symbol underscores CIFAR-10's role as a source dataset used to inform our search strategy, rather than as a target for performance evaluation.

From the results in Table II, DSM-NAS+ with the subspaces transferred from CIFAR-10 achieves higher accuracy and lower variance than DSM-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). This is because that the well-designed architectures on CIFAR-10 may also have high performance on the other datasets (*e.g.*, CIFAR-100 and ImageNet-16-120). Thus, subspaces searched in CIFAR-10 provide a good initialization when searching in a new dataset. Employing additional data from CIFAR-10 gives DSM-NAS a strategic advantage. This approach is particularly useful when adapting to a new target dataset. This strategy not only potentially lowers computational costs but also boosts performance, ensuring a more efficient and productive search.

B. Performance Comparisons on NAS-Bench-360

We further apply DSM-NAS to NAS-Bench-360 [61] benchmark. In addition to the image classification task, we further consider the tasks of electromyography signals classification and partial differential equations (PDEs) solving. We first provide the details of tasks and implementation and then compare our DSM-NAS with state-of-the-art methods.

Considered Tasks. Following NAS-Bench-360, we consider two more tasks, *i.e.*, *NinaPro* and *Darcy Flow*. NinaPro seeks to classify hand gestures indicated by electromyography signals. We use a subset of the NinaPro DB5 dataset [65] that contains EMG signals from 10 test individuals with 18 different hand gestures. Darcy Flow is a regression task that focuses on mapping from the initial conditions of a PDE to the solution at a later timestep. Its input is a 2d grid specifying the initial conditions of a fluid, and the output is a 2d grid specifying the fluid state at a later time, with the ground truth being the result computed by a traditional solver. We use the same Darcy Flow dataset that was used in [66]. We report the mean square error. Note that NAS-Bench-360 provides precomputed accuracies on these datasets.

Implementation Details. Following the settings in NAS-Bench-360, the search space is also the same as NAS-Bench-201. The hyperparameters for policy learning are the same as those in NAS-Bench-201. Specifically, we train our DSM-NAS with a batch size of 1 and set the strength of the entropy regularizer to 7.5×10^{-4}. We use the 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.

Comparisons with State-of-the-art Methods. In Table III, we compare our DSM-NAS on the *NinaPro* and *Darcy Flow* tasks of the NAS-Bench-360 benchmark. We report the average test accuracy over 500 runs with different seeds. Our DSM-NAS achieves high accuracy than REINFORCE [49] (0.0211 *vs.* 0.0252 on Darcy Flow) and Evolution [37] (0.0211 *vs.* 0.0247 on Darcy Flow) baselines. Compared with the weight-sharing based NAS methods (*e.g.*, ENAS [13] and DARTS [11]), DSM-NAS still outperforms them. This superior performance can be attributed to DSM-NAS's focus on a smaller, more effective subspace of the search space, as opposed to exploring the entire vast search space. This approach reduces the complexity of the search, making it more likely to yield promising architectural designs. The results verify the effectiveness of DSM-NAS across various tasks, extending beyond the realm of image classification.

In addition, we find that weight-sharing NAS methods (*i.e.*, ENAS [13], DARTS [11], RSWS [64], GDAS [53] and SETN [52]) achieve worse performance than the Random Search baseline on the NinaPro dataset. The reasons may be twofold. First, the primary issue with weight sharing in methods like ENAS [13] is multi-model forgetting [67], [68], where the performance of previously trained models deteriorates due to the overwriting of shared parameters during sequential training of multiple networks. The degree of performance degradation is directly proportional to the amount of shared weights, with more sharing leading to greater impacts. Additionally, the complexity of optimization increases, making it more challenging to improve subsequent models without negatively affecting the performance of earlier ones. These challenges underscore the need for careful parameter management to minimize performance interference between models. Therefore, weight sharing cannot provide accurate model performance evaluation, leading to limited search performance. Second, prior studies [56], [61] demonstrate that the search process in methods like DARTS [11], unintentionally prefers architectures with a higher number of skip connections. While a balanced number of skip connections can boost

TABLE IV: Comparisons of the architectures searched/designed by different methods on ImageNet. "–" means unavailable results. "#Queries", a widely used metric [69], [70], 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 DSM-NAS outperforms 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×) [71]	74.7	–	585	–	–
	ShuffleNetV2 (2×) [72]	73.7	–	524	–	–
NASNet	NASNet-A [44]	74.0	91.6	564	20	1800
	AmoebaNet-A [37]	74.5	92.0	555	20	3150
DARTS	DARTS [11]	73.1	91.0	595	19.5	4
	P-DARTS [73]	75.6	92.6	577	11.7	0.3
	CNAS [15]	75.4	92.6	576	100	0.3
	AlphaX [17]	75.5	92.2	579	–	12
	DARTS- [56]	74.6	92.1	547	93.6	4.5
	Shapley-NAS [29]	76.1	–	582	–	4.2
	RF-DARTS [30]	76.0	92.4	593	93.6	–
MobileNet-like	RLNAS [74]	75.6	92.6	473	–	–
	AtomNAS [75]	75.9	92.0	367	78	–
	Few-shot NAS [47]	75.9	–	521	–	11.6
	FairNAS [76]	77.5	93.7	392	11.2	12
	DNA [77]	78.4	94.0	611	–	32.6
	FBNetV2 [78]	77.2	–	325	11.5	25
	EfficientNet-B1 [79]	79.2	94.5	734	–	–
	OFA-Large [14]	80.0	94.9	595	20	51.7
	Cream-L [80]	80.0	94.7	604	–	12
	NEAS-L [33]	80.0	94.8	574	–	<13
	GM [81]	76.6	93.0	530	–	24.9
	MAGIC-AT [82]	76.8	93.4	598	–	–
	NAS-LID [83]	77.1	93.7	678	–	2.3
	ZiCo [43]	79.4	–	603		0.4
	DSM-NAS (Ours)	79.9	94.8	597	**10**	**0.8**
	DSM-NAS+ (Ours)	**80.2**	**94.9**	582	**10**	51.7+0.8

performance, prompting NAS algorithms to favor the skip connection operation, an excessive focus on this aspect often leads to less optimal search results. Excessive reliance on skip connections would affect search performance, producing architectures overwhelmed by these operations and reducing their overall effectiveness. This narrows the diversity of the explored architecture space and overfits to a particular pattern, which may not necessarily result in superior performance.

C. Performance Comparisons on ImageNet

Search Space. In this experiment, we further evaluate our method on another benchmark search space, *i.e.*, MobileNet-like search space [59]. 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 R selected from $\{3, 5, 7\}$, expansion ratios E selected from $\{3, 4, 6\}$ and the number of layers in each unit selected from $\{2, 3, 4\}$. To encode the architecture, we use a string of length 20, in which the element in the string represents the combination setting of the expansion ratio and the kernel size. For instance, the element "0" denotes non-existent layer, "1" denotes the layer with $E = 3$ and $R = 3$, "2" denotes the layer with $E = 3$ and $R = 4$, and so on. In this case, we ensure that our proposed method can effectively calculate the distance between architectures with varying depths.

Following [84], 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 [14] for 90 epochs. To compute the performance improvement $R(\beta|\alpha)$ with the validation accuracy, we train a predictor to predict the validation accuracy following [14].

Implementation Details. Following [13], we train DSM-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 exploration of DSM-NAS, we add an entropy regularizer to the reward weighted by 1×10^{-3}. We set the number of candidate subspaces K to 10 and the local search distance M to 3. We

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

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

report the search cost based on the NVIDIA Tesla V100 GPU. Following the *mobile setting* [11], 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 (line 12 in Algorithm 1) with the architecture that has #MAdds less than 600M. To accelerate model evaluation, following [14], [32], we first obtain the parameters from the full network of OFA and then finetune them for 75 epochs on the ImageNet training set (containing 128k images). We use a batch size of 512 and 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. We perform data augmentations including horizontally flipping, random crops, color jittering, and AutoAugment.

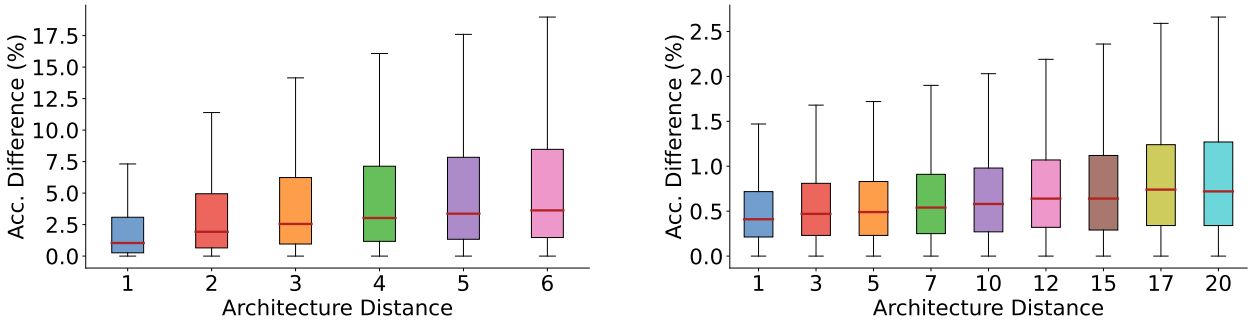
Comparisons with State-of-the-art Methods. To investigate the effectiveness of the proposed method, we apply our method to MobileNet-like search space as two variants: 1) **DSM-NAS** searches based on the subspace graph initialized with randomly sampled architectures, which is suitable in the scenario without any available well-designed architectures. 2) **DSM-NAS+** adopts the subspace graph which is initialized with a set of existing well-designed architectures searched by OFA [14]. Note that our DSM-NAS outperforms SOTA methods and DSM-NAS+ further boosts the search performance.

As shown in Table IV, under the mobile setting, the architecture searched by DSM-NAS reaches 79.9% top-1 accuracy and 94.8% top-5 accuracy, which outperforms not only the manually designed architectures but also most automatically searched ones. Specifically, DSM-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), DSM-NAS also achieves competitive performance (*i.e.*, 79.9% *vs.* 80.0%) with fewer 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 DSM-NAS.

Compared with zero-shot NAS method ZiCo [43], DSM-NAS achieves higher accuracy than ZiCO (79.9% *vs.* 79.4%) under the *mobile setting* (#MAdds $<$ 600M). The potential reasons are that our DSM-NAS searches by focusing on the small but effective subspace, which reduces the search difficulty resulting from the large search space, while ZiCo adopts a vanilla evolutionary algorithm that still searches in the whole search space. As for the search cost, DSM-NAS consumes more GPU resources than ZiCO (0.8 *vs.* 0.4 GPU days). The reason is that DSM-NAS needs to train a supernet (0.7 GPU days) on a proxy ImageNet subset, which is costly compared with the zero-shot proxy used in ZiCO.

Compared with DSM-NAS, our DSM-NAS+ further improves the top-1 accuracy on ImageNet from 79.9% to 80.2%. Note that DSM-NAS+ outperforms all of the considered manually-designed and automatically searched architectures. The reason is that DSM-NAS searches for the dominative subspace from randomly initialized candidates and needs to improve the candidate subspaces gradually. Instead, DSM-NAS+ directly employs the dominative 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 DSM-NAS (top-1 accuracy 79.9%) and DSM-NAS+ (top-1 accuracy 80.2%) in MobileNet-like search space in Figure 4 and Figure 5, respectively. From the visualization results, we find 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 [85]. Second, larger conv operations may help the network preserve more information when doing downsampling (mentioned by ProxylessNAS [86]). In this case, architectures with conv 5×5



(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).

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 [14], [86].

V. More Ablation Studies

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 dominative subspace. In other words, the architectures in the neighborhood/subspace are more likely to have good performance. In this sense, once we find a dominative 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 the rationality and effectiveness of the designed architecture distance, *i.e.*, architectures with smaller distances (in the same subspace) tend to have similar performance. Thus, we are able to find promising architectures in a dominative subspace around existing good architectures more easily than the overall search space. Similar observations are also found in NAS-Bench-101 [18] search space.

We find that the performance difference in the NAS-Bench-201 search space is more obvious than the MobileNet-like search space. The is because that the candidate operations are different in these two search spaces In the NAS-Bench-201 search space, the available operations are: (1) zeroize, (2) skip connection, (3) 1-by-1 convolution, (4) 3-by-3 convolution, and (5) 3-by-3 average pooling. The "zeroize" operation essentially drops the features, leading to a substantial impact on model performance. When an architecture undergoes a transition that involves the "zeroize" operation, the model performance changes sharply. Instead, in the MobileNet-like search space, we search for the expansion ratio and kernel size from $\{3, 4, 6\}$ and $\{3, 5, 7\}$, respectively. Unlike the NAS-Bench-201 space, the MobileNet-like search space does not include the "zeroize" operation. As a result, changes in architecture distance tend to produce more subtle variations.

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 α in the mined subspace with locally searched architecture β if β has better performance than α. 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 process.

To verify the effectiveness of the proposed subspace updating scheme, we conduct more experiments on NAS-Bench-201 and compare our DSM-NAS with three baselines and variants, namely *Random Search in Overall Search Space*,

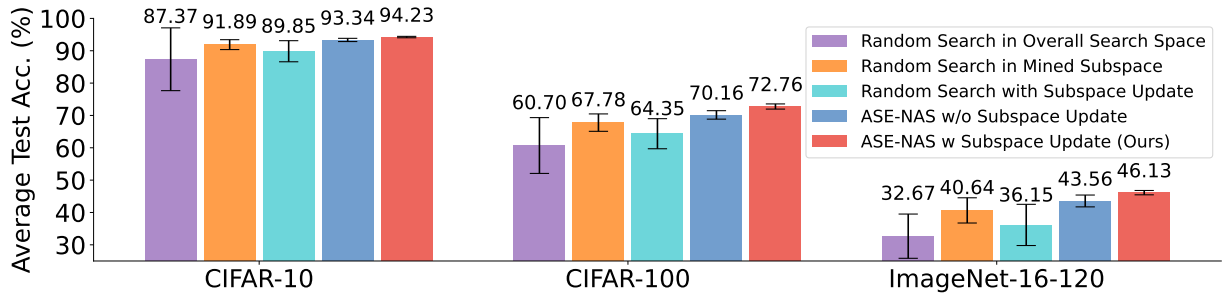


Fig. 7: Comparisons of the search performance with/without subspace updating on NAS-Bench-201.

Random Search in Mined Search Space, *Random Search with Subspace Update* and *DSM-NAS without Subspace Update*. The first two baselines conduct a random search in the entire search space and the mined subspace in the final search step of our method, respectively. The third baseline performs a random search with our proposed subspace update strategy. The variant *DSM-NAS without Subspace Update* uses the same settings as our method but performs a search without updating the candidate subspaces.

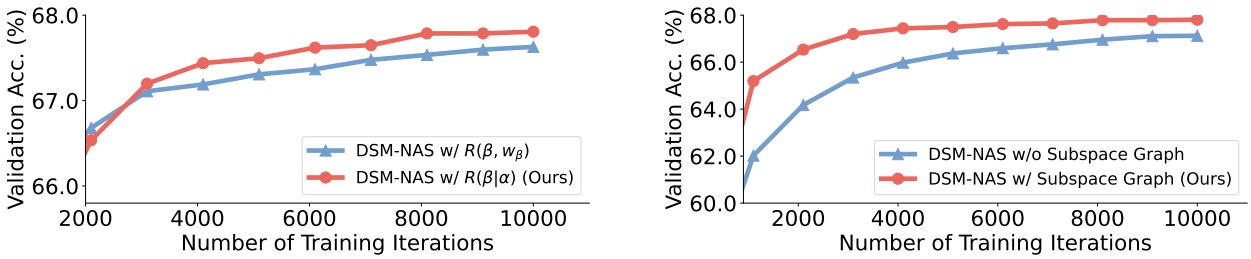
From the results in Figure 7, random search in the mined subspace (orange) has higher average accuracy and lower variances than that in the overall search space (purple), *i.e.*, 91.89±1.53% *vs.* 87.37±9.71%, which demonstrate the effectiveness of the proposed subspace updating scheme. Our DSM-NAS with subspace (red) updating outperforms DSM-NAS without that (blue) on three considered datasets. The results indicate that DSM-NAS constantly can find dominative 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 mined subspace, DSM-NAS consistently achieves better search accuracy. The experimental results demonstrate that the local search scheme is able to further enhance the search performance by finding promising architectures in the mined subspace. In addition, the random search with a subspace update baseline (cyan) surpasses the performance of the standard random search across the entire search space (purple). This improvement suggests that updating the search subspace leads to the identification of more promising regions within the search space. However, random search with subspace update (cyan) does not perform as well as random search in a mined search space (orange). This implies that relying solely on random search is insufficient for effectively finding the most effective subspaces.

As for the search speed, random search baselines require less than 1 microsecond to sample architectures from a uniform distribution since it does not depend on any heavy deep models. We sample a total of 100 architectures and then select the one with the highest predicted accuracy. The total time for this process is less than 1 second. For DSM-NAS, the controller requires 0.1 seconds to generate an architecture by performing global and local searches and 0.22 seconds to update the policy parameters according to the estimated performance by the performance predictor. In MobileNet-like search space, we train the controller for 10k iterations with a batch size of 1 following [13]. Thus, the policy search cost is 1 GPU hour (*i.e.*, 0.04 GPU days). Though random search requires a very low search cost, it struggles to uncover promising architectures because of its unplanned attempts. In contrast, DSM-NAS significantly outperforms random search baselines. This is attributed to our innovative global and local search strategy, which iteratively identifies promising architectures by leveraging knowledge from previous explorations. This strategic method ensures a more directed and efficient search process, leading to superior performance in finding optimal architectures.

C. Effect of the Performance Improvement Reward

In policy learning, we use the performance improvement between the resultant β and the center architecture α in Ω_α as the reward. Besides this, one can learn the policy by considering the performance of β 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. All the hyperparameters are the same as in Section IV-C. We report the validation accuracy over 10 different runs estimated by the accuracy predictor and compare our DSM-NAS with the variant. For the tested variant, we maintain consistency in all experimental conditions except for the component under examination.

In Figure 8(a), we employ a variant as the baseline to verify the effectiveness of the proposed reward function. This variant directly uses the performance of the resultant architecture β as reward in policy learning. From the results, DSM-NAS using the performance improvement (red) achieves higher search performance than that using the performance of resultant architecture (blue) (67.81±0.18% *vs.* 67.63±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 β, 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 the search. In contrast, using performance improvement encourages explorations among different subspaces.



(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.

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 implies 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 dominative subspace in the global search.

We investigate the effect subspace graph by performing more experiments in the MobileNet-like search space with/without the subspace graph structure. The experimental setup is the same as that in Section V-C. Specifically, we perform an ablation study by comparing our DSM-NAS with a variant without the subspace graph structure. For this variant, 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 supernet) over 10 different runs. From the results, DSM-NAS with subspace graph (red) has not only higher validation accuracy but also lower variance than DSM-NAS without that (blue) (67.81±0.18% *vs.* 67.12±0.46%). The results demonstrate the significant role of the subspace graph in guiding the search process. Our DSM-NAS with the subspace graph outperformed the variant without it, highlighting the graph's effectiveness in narrowing the search scope and concentrating efforts on promising architectural regions.

E. Effect of the Number of Candidate Subspaces

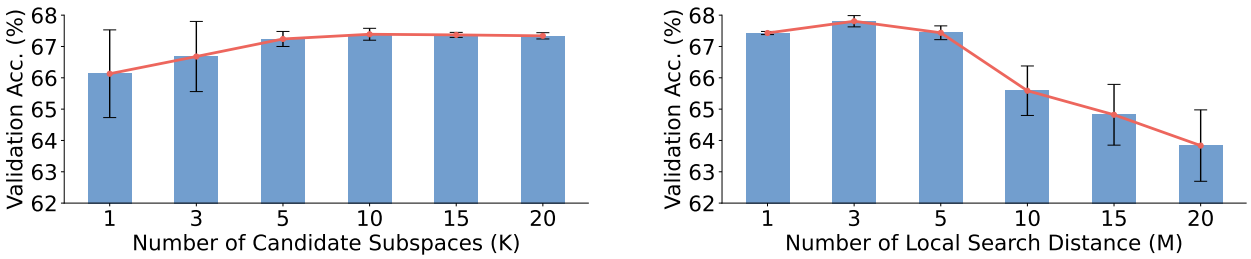
We build the subspace graph with K architectures and thus have K candidate subspaces $\{\Omega_{\alpha_i}\}_{i=1}^{K}$. When we consider a small K, the information carried by the subspaces would be limited, resulting in poor search performance. In contrast, a larger K means more exploration in the candidate subspaces. Nevertheless, too many candidate subspaces would introduce a heavy computational burden since the computational cost of the GNN increases quadratically as 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 be the same.

In Figure 9(a), DSM-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, DSM-NAS achieves better validation accuracy. The reason is that more architectures benefit the search by exploring more diverse dominative subspaces. In this case, DSM-NAS has a larger probability of finding promising architectures. Besides, when K is larger than 10, DSM-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

When performing the local search in Ω_α, 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 DSM-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 a 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), DSM-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 architectures. In this case, the search performance of larger search space drops greatly compared with that of small search space (*e.g.*, only 63.84% when M=20). Thus, we set the search distance M to 3 in the MobileNet-like search space.



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

Fig. 9: Comparisons of search performance with different candidate subspaces (a) and different local search distances (b).

VI. More Discussions of DSM-NAS

A. More Discussions on Computational Complexity

The computational/time cost of DSM-NAS consists of two parts, *i.e.*, the performance estimation cost and policy learning cost. 1) Cost of Architecture Performance Estimation. To assess the candidate architectures, we train a supernet on 10% of the original dataset's classes, as suggested by [84]. We then evaluate the validation accuracy of sub-networks on a subset of 1000 validation images. The supernet is trained with a progressive shrinking strategy [14] over 90 epochs. Additionally, to compute the performance improvement $R(\beta|\alpha)$ with the validation accuracy, we train a predictor to estimate validation accuracy. Together, training the supernet and the predictor requires approximately 0.75 GPU days on an NVIDIA V100 GPU. 2) Cost of Policy Learning. In Mobilenet-like search space, the controller, composed of a two-layer Graph Neural Network (GNN) and an LSTM, is trained over 10k iterations. Each iteration includes forward and backward propagations, consuming 2.73M FLOPs per iteration, totaling 27.3G FLOPs. The overall training time for the controller is about 1 GPU hour (0.04 GPU days).

B. More Discussions on Advanced Subspace Search Strategies

DSM-NAS facilitates extensive exploration beyond the initially identified dominant subspace through two key strategies: 1) Policy Learning for Diversity: In Eqn. (2), we have introduced a unique policy update strategy that prioritizes performance improvement over absolute performance. This method helps avoid local optima by encouraging the exploration of new subspaces whenever the incremental performance improvement is negligible. This is because if the policy is in a local optimum, the performance improvement would be around zero. In this case, DSM-NAS would be encouraged to explore other subspaces in the subsequent iterations, which boosts the diversity of the explored search subspaces. 2) Dynamic Subspace Graph Update: We dynamically update the subspace graph based on the performance of newly discovered architectures. When an architecture is identified that performs better than the current ones, the subspace graph is updated to include a new subspace centered around this superior architecture. By only updating the subspace with a clear performance improvement, we maintain the integrity of our search results and ensure that the subspaces we identify are always promising and accurate regions of the search space.

C. Mutual Influence of Global and Local Search

In this section, we depict the mutual influence, relationship and contributions of our global and local search strategies.

1) Mutual Influence and Relationship: The global search seeks to identify a dominative subspace from a set of candidate subspaces, mining a small and effective search space for further exploration. Once a dominative subspace is determined, the local search operates within this subspace to find effective architectures. The mutual influence is evident as the results of the local search feed back into the global search. In this sense, the relationship between local and global search is complementary. When a better architecture is found through local search, we update the subspace graph with this architecture, which in turn influences subsequent global searches. This creates a feedback loop where the global search helps to guide the local search to promising regions, and the local search helps to improve the subspaces for the next global searches.

2) Contributions of Global and Local Search: By identifying small and effective subspaces, the global search significantly reduces the complexity of the search problem. This contribution is crucial in the early stages of the search process, where the need for a broad overview and quick narrowing down of the search space is imperative. On the other hand, the local search shines in its capacity for detailed exploration within the confined areas identified by the global search. Its contribution is most noticeable in meticulously exploring the narrowed-down space to find effective architectures.

D. More Discussions on Initial Subspaces

The search performance of our DSM-NAS depends on the initial conditions, particularly the number of initial subspaces K. However, our DSM-NAS does not need a great number of initial subspaces from three aspects:

Our proposed strategy significantly increases the diversity of the explored subspace. Our DSM-NAS incorporates advanced exploration strategies specifically designed to navigate through large search spaces effectively. Specifically, as detailed in Eqn. (2), we employ a policy optimization scheme that prioritizes performance improvement over the absolute performance of the resultant architecture. This strategy targets subspaces

with the highest potential to yield superior architectures, rather than those that merely present the best-found architecture at the moment. Our empirical analysis (refer to Figure 8(a)) demonstrates that our DSM-NAS is capable of exploring a more diverse range of subspaces in subsequent search phases. This significantly diminishes the risk of overfitting to initially sampled subspaces.

Transferring from previously searched architectures improves the initialization of candidate subspaces. In addition to employing a random initialization strategy for candidate subspaces, our DSM-NAS enhances search performance by utilizing well-designed or previously searched architectures. This is evident from the results shown in Tables II and IV, where initializing with promising architectures leads to improved outcomes. By integrating prior knowledge of effective architectural patterns, we establish a robust mechanism for initializing candidate subspaces. This not only reduces our dependence on a large number of initial subspaces to comprehensively explore the search space but also allows us to focus our efforts on the neighborhood of promising subspaces. Such a targeted search approach effectively addresses the challenges of large search spaces and variability due to initial conditions, thereby enhancing the effectiveness of our algorithm.

Empirical studies have confirmed that $K{=}10$ is sufficient for exploring a large search space effectively. We have conducted ablations to investigate the effect of the number of candidate subspaces in MobileNet-like search space. It is crucial to highlight that this search space is exceptionally vast, with its size reaching up to 10^{19}, significantly surpassing the dimensions of other adopted search spaces such as NAS-Bench-201, which has a size of 10^5. Our empirical findings reveal that setting $K{=}10$ is sufficient to achieve promising search performance, demonstrating that even in extensive search spaces, a modest value of K can yield satisfactory results. Interestingly, when $K{>}10$, the performance stabilizes and exhibits negligible variations, indicating a plateau in performance gains. This suggests that the necessity for a substantial value of K to ensure optimal performance as highlighted in the initial query, may not be as critical as presumed in large search spaces.

E. More Discussions on Convergence Property

DSM-NAS employs a dynamic subspace update strategy, a pivotal feature designed to iteratively refine the search subspace, thereby improving the search's efficacy and efficiency. This strategy is governed by a key rule: only those subspaces that yield superior performance architectures are considered for replacing and updating existing ones. This targeted updating mechanism ensures that the search is consistently steered towards more promising areas within the architecture space. Given a sufficiently large number of iterations, we anticipate that DSM-NAS will converge as it increasingly focuses the search on subspaces that demonstrate consistent improvements over previous iterations. This iterative refinement process, driven by a performance optimization goal, is indicative of a convergence towards an optimal or near-optimal solution in the architectural space.

While our manuscript does not provide traditional mathematical proofs of convergence and optimality, the design and operational logic of the algorithm, particularly the dynamic subspace update strategy, offer a conceptual basis for expecting properties of convergence and optimality. Additionally, the experimental results presented in Tables II and IV validate the effectiveness of DSM-NAS. Together, the analytical insights and empirical evidence presented address the concerns about the theoretical underpinnings of our algorithm.

VII. Conclusion

In this paper, we proposed a Neural Architecture Search method via Dominative Subspace Mining (DSM-NAS), which focuses on automatically mining small and effective subspaces and conducts a search in them. Specifically, we first perform a global search for a dominative subspace from the candidate subspaces. Then, we perform a 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, DSM-NAS further enhance search performance by taking well-designed/searched architectures as the prior knowledge. Extensive experiments on several benchmark search spaces demonstrate the superiority of our method over the considered methods.

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, pp. 1–21.

[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] 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.

[5] 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.

[6] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," in *International Conference on Learning Representations*, 2018, pp. 1–12.

[7] 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.

[8] Y.-F. Song, Z. Zhang, C. Shan, and L. Wang, "Richly activated graph convolutional network for robust skeleton-based action recognition," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 5, pp. 1915–1925, 2021.

[9] X. Shu, J. Yang, R. Yan, and Y. Song, "Expansion-squeeze-excitation fusion network for elderly activity recognition," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 8, pp. 5281–5292, 2022.

[10] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *International Conference on Learning Representations*, 2017, pp. 1–16.

[11] H. Liu, K. Simonyan, and Y. Yang, "DARTS: differentiable architecture search," in *International Conference on Learning Representations*, 2019, pp. 1–13.

[12] 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.

[13] 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.

[14] 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, pp. 1–15.

[15] 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.

[16] 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.

[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. Wu and J. Zhou, "Iid-net: Image inpainting detection network via neural architecture search and attention," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 3, pp. 1172–1185, 2022.

[20] Z. Mei, P. Ye, H. Ye, B. Li, J. Guo, T. Chen, and W. Ouyang, "Automatic loss function search for adversarial unsupervised domain adaptation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 1, no. 1, pp. 1–14, 2023.

[21] B. Guo, L. Xu, T. Chen, P. Ye, S. He, H. Liu, and J. Chen, "Latency-aware neural architecture performance predictor with query-to-tier technique," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 1, no. 1, pp. 1–1, 2023.

[22] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *International Conference on Learning Representations*, 2017, pp. 1–18.

[23] 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.

[24] 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.

[25] Z. Liu, Z. Shen, Y. Long, E. Xing, K.-T. Cheng, and C. Leichner, "Data-free neural architecture search via recursive label calibration," in *European Conference on Computer Vision*, 2022, pp. 391–406.

[26] A. C. Yüzügüler, N. Dimitriadis, and P. Frossard, "U-boost nas: Utilization-boosted differentiable neural architecture search," in *European Conference on Computer Vision*, 2022, pp. 173–190.

[27] X. Chen, R. Wang, M. Cheng, X. Tang, and C. Hsieh, "Drnas: Dirichlet neural architecture search," in *International Conference on Learning Representations*, 2021, pp. 1–17.

[28] P. Xie and X. Du, "Performance-aware mutual knowledge distillation for improving neural architecture search," in *IEEE Conference on Computer Vision and Pattern Recognition*, June 2022, pp. 11 922–11 932.

[29] H. Xiao, Z. Wang, Z. Zhu, J. Zhou, and J. Lu, "Shapley-nas: Discovering operation contribution for neural architecture search," in *IEEE Conference on Computer Vision and Pattern Recognition*, June 2022, pp. 11 892–11 901.

[30] X. Zhang, Y. Li, X. Zhang, Y. Wang, and J. Sun, "Differentiable architecture search with random features," in *IEEE Conference on Computer Vision and Pattern Recognition*, June 2023, pp. 16 060–16 069.

[31] H. Huang, L. Shen, C. He, W. Dong, and W. Liu, "Differentiable neural architecture search for extremely lightweight image super-resolution," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 33, no. 6, pp. 2672–2682, 2023.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] J. Pan, C. Sun, Y. Zhou, Y. Zhang, and C. Li, "Distribution consistent neural architecture search," in *IEEE Conference on Computer Vision and Pattern Recognition*, June 2022, pp. 10 884–10 893.

[37] 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.

[38] 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.

[39] M. S. Abdelfattah, A. Mehrotra, L. Dudziak, and N. D. Lane, "Zero-cost proxies for lightweight NAS," in *International Conference on Learning Representations*, 2021, pp. 1–14.

[40] N. Lee, T. Ajanthan, and P. H. S. Torr, "Snip: single-shot network pruning based on connection sensitivity," in *International Conference on Learning Representations*, 2019, pp. 1–14.

[41] C. Wang, G. Zhang, and R. B. Grosse, "Picking winning tickets before training by preserving gradient flow," in *International Conference on Learning Representations*, 2020, pp. 1–14.

[42] H. Tanaka, D. Kunin, D. L. K. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020, pp. 6377–6389.

[43] G. Li, Y. Yang, K. Bhardwaj, and R. Marculescu, "Zico: Zero-shot NAS via inverse coefficient of variation on gradients," in *International Conference on Learning Representations*, 2023, pp. 1–14.

[44] 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.

[45] 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.

[46] 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.

[47] Y. Zhao, L. Wang, Y. Tian, R. Fonseca, and T. Guo, "Few-shot neural architecture search," in *International Conference on Machine Learning*, 2021, pp. 12 707–12 718.

[48] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollar, "Designing network design spaces," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 10 425–10 433.

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

[50] 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.

[51] X. Dong and Y. Yang, "NAS-Bench-201: Extending the scope of reproducible neural architecture search," in *International Conference on Learning Representations*, 2020, pp. 1–16.

[52] 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.

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

[54] 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.

[55] 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, pp. 1–13.

[56] X. Chu, X. Wang, B. Zhang, S. Lu, X. Wei, and J. Yan, "DARTS-: robustly stepping out of performance collapse without indicators," in *International Conference on Learning Representations*, 2021, pp. 1–22.

[57] Y. Akhauri, J. P. Muñoz, N. Jain, and R. Iyer, "EZNAS: evolving zero-cost proxies for neural architecture scoring," in *Advances in Neural Information Processing Systems*, 2022.

[58] Z. Zhang and Z. Jia, "Gradsign: Model performance inference with theoretical insights," in *International Conference on Learning Representations*, 2022, pp. 1–21.

[59] 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.

[60] 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.

[61] R. Tu, N. Roberts, M. Khodak, J. Shen, F. Sala, and A. Talwalkar, "Nas-bench-360: Benchmarking neural architecture search on diverse tasks," in *Advances in Neural Information Processing Systems*, 2022.

[62] S. Falkner, A. Klein, and F. Hutter, "BOHB: robust and efficient hyperparameter optimization at scale," in *International Conference on Machine Learning*, J. G. Dy and A. Krause, Eds., vol. 80, 2018, pp. 1436–1445.

[63] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *Journal of Machine Learning Research*, vol. 18, pp. 185:1–185:52, 2017.

[64] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, A. Globerson and R. Silva, Eds., vol. 115. AUAI Press, 2019, pp. 367–377.

[65] U. Côté-Allard, C. L. Fall, A. Drouin, A. Campeau-Lecours, C. Gosselin, K. Glette, F. Laviolette, and B. Gosselin, "Deep learning for electromyographic hand gesture signal classification using transfer learning," *IEEE transactions on neural systems and rehabilitation engineering*, vol. 27, no. 4, pp. 760–771, 2019.

[66] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier neural operator for parametric partial differential equations," in *International Conference on Learning Representations*, 2021, pp. 1–16.

[67] Y. Benyahia, K. Yu, K. Bennani-Smires, M. Jaggi, A. C. Davison, M. Salzmann, and C. Musat, "Overcoming multi-model forgetting," in *International Conference on Machine Learning*, vol. 97, 2019, pp. 594–603.

[68] M. Zhang, H. Li, S. Pan, X. Chang, and S. W. Su, "Overcoming multi-model forgetting in one-shot NAS with diversity maximization," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2020, pp. 7806–7815.

[69] 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.

[70] 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.

[71] 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.

[72] 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.

[73] 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.

[74] X. Zhang, P. Hou, X. Zhang, and J. Sun, "Neural architecture search with random labels," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2021, pp. 10 907–10 916.

[75] 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, pp. 1–14.

[76] 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.

[77] 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.

[78] 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.

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

[80] 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.

[81] S. Hu, R. Wang, L. Hong, Z. Li, C.-J. Hsieh, and J. Feng, "Generalizing few-shot nas with gradient matching," in *International Conference on Learning Representations*, 2022, pp. 1–14.

[82] J. Xu, X. Tan, K. Song, R. Luo, Y. Leng, T. Qin, T. Liu, and J. Li, "Analyzing and mitigating interference in neural architecture search," in *International Conference on Machine Learning*, 2022, pp. 24 646–24 662.

[83] X. He, J. Yao, Y. Wang, Z. Tang, K. C. Cheung, S. See, B. Han, and X. Chu, "Nas-lid: Efficient neural architecture search with local intrinsic dimension," in *AAAI Conference on Artificial Intelligence*, 2023, pp. 7839–7847.

[84] 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.

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

[86] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," in *International Conference on Learning Representations*, 2019, pp. 1–13.

Yaofo Chen received the B.E. degree in Software Engineering from South China University of Technology, China, in 2018. He is currently pursuing the Ph.D. degree in the School of Software Engineering, South China University of Technology, China. His research interests include Neural Architecture Search and Computer Vision.

Yong Guo is currently a postdoctoral researcher in Max Planck Institute for Informatics. He received his bachelor's degree from South China University of Technology in 2016 and his Ph.D. degree from the same university in 2021. His research interests include deep learning and computer vision.

Daihai Liao is currently a Senior Artificial Intelligence Engineer with Changsha Hisense Intelligent System Research Institute Co., Ltd and has applied for 5 invention patents. He received his Master degree in Control Science and Engineering from Central South University, China. His research interests include object detection, object tracking and semantic segmentation.

Fanbing Lv is currently a Deputy Chief Engineer, and Senior Engineer of Big Data with Changsha Hisense Intelligent System Research Institute Co., Ltd and has applied for 20 invention patents. He was selected into Guiyang Big Data Hundred Talents Program and High-level Talent Green Card (Class C) and won the "First Prize" in the Fourth Guizhou Innovative Product Design Competition

Hengjie Song is currently a Professor with the School of Software Engineering, South China University of Technology. He has published several high quality articles on the best journals and conference proceedings, including the *ACM Transactions on the Web*, the IEEE CIMs, the IEEE TRANSACTIONS ON FUZZY SYSTEMS, *Neural Networks* (Elsevier), AAAI, and ICDM. His research interests include artificial intelligence and the applications of AI in commercial search engines.

James Tin-Yau Kwok (Fellow, IEEE) received the Ph.D. degree in computer science from The Hong Kong University of Science and Technology, Hong Kong, in 1996. He is currently a Professor with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology. His current research interests include kernel methods, machine learning, pattern recognition, and artificial neural networks. He received the IEEE Outstanding Paper Award in 2004 and the Second Class Award in Natural Sciences from the Ministry of Education, China, in 2008. He has been a Program Co-Chair for a number of international conferences and served as an Associate Editor for the IEEE TRANS-ACTIONS ON NEURAL NETWORKS AND LEARNINGSYSTEMS from 2006 to 2012. He is currently an Associate Editor of Neurocomputing.

Mingkui Tan is currently a professor with the School of Software Engineering at South China University of Technology. He received his Bachelor Degree in Environmental Science and Engineering in 2006 and Master degree in Control Science and Engineering in 2009, both from Hunan University in Changsha, China. He received his Ph.D. degree in Computer Science from Nanyang Technological University, Singapore, in 2014. From 2014-2016, he worked as a Senior Research Associate on computer vision in the School of Computer Science, University of Adelaide, Australia. His research interests include machine learning, sparse analysis, deep learning, and large-scale optimization.