Efficient Few-Shot Neural Architecture Search by Counting the Number of Nonlinear Functions

Youngmin Oh[1], **Hyunju Lee**[1], **and Bumsub Ham**[1,2*]

[1]Yonsei University
[2]Korea Institute of Science and Technology (KIST)
{youngmin.oh, hyunjulee, bumsub.ham}@yonsei.ac.kr

Abstract

Neural architecture search (NAS) enables finding the best-performing architecture from a search space automatically. Most NAS methods exploit an over-parameterized network (*i.e.*, a supernet) containing all possible architectures (*i.e.*, subnets) in the search space. However, the subnets that share the same set of parameters are likely to have different characteristics, interfering with each other during training. To address this, few-shot NAS methods have been proposed that divide the space into a few subspaces and employ a separate supernet for each subspace to limit the extent of weight sharing. They achieve state-of-the-art performance, but the computational cost increases accordingly. We introduce in this paper a novel few-shot NAS method that exploits the number of nonlinear functions to split the search space. To be specific, our method divides the space such that each subspace consists of subnets with the same number of nonlinear functions. Our splitting criterion is efficient, since it does not require comparing gradients of a supernet to split the space. In addition, we have found that dividing the space allows us to reduce the channel dimensions required for each supernet, which enables training multiple supernets in an efficient manner. We also introduce a supernet-balanced sampling (SBS) technique, sampling several subnets at each training step, to train different supernets evenly within a limited number of training steps. Extensive experiments on standard NAS benchmarks demonstrate the effectiveness of our approach.

Code — https://cvlab.yonsei.ac.kr/projects/EFS-NAS

Introduction

Manually designing network architectures is a labor-intensive and time-consuming process. Neural architecture search (NAS) helps to automate the designing process, and provides optimal network architectures for various hardware configurations (*e.g.*, FLOPs). Early NAS methods (Zoph et al. 2018; Baker et al. 2017; Zoph and Le 2017) adopt reinforcement learning (Williams 1992) with policy networks (*i.e.*, controllers), which requires training deep neural networks from scratch, taking lots of computational costs (*e.g.*, typically thousands of GPU hours). To overcome this problem, one-shot NAS approaches (Liu, Simonyan, and Yang 2019; Guo et al. 2020; Xu et al. 2020; Chu, Zhang, and Xu 2021; Cai, Zhu, and Han 2019) adopt a weight-sharing technique (Pham et al. 2018), where they train a single supernet that consists of all possible network architectures (*i.e.*, subnets) in a given search space. The trained supernet can act as a performance estimator for various subnets, sampled from the supernet, indicating that each subnet does not need to be trained from scratch to predict its performance. Although one-shot NAS methods are efficient, they are limited in that the performance of the subnet estimated from the supernet is less correlated with the one obtained from training the subnet from scratch. The major reason is that the parameters of the supernet suffer from conflicts between subnets during training. To address this problem, few-shot NAS methods (Zhao et al. 2021; Hu et al. 2022; Su et al. 2021) propose to exploit multiple supernets. They typically

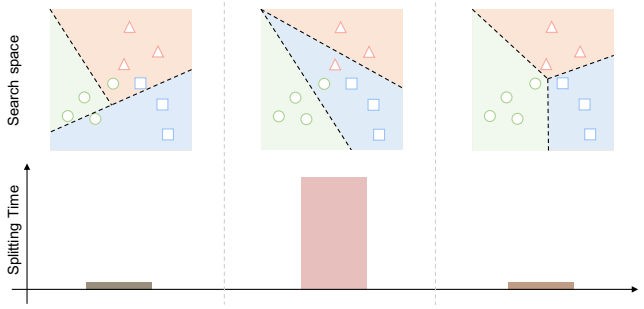


Figure 1: Illustration of search space splitting strategies. Individual supernets are highlighted in different colors. Subnets with similar characteristics are marked by the same shape. **Left:** FS-NAS (Zhao et al. 2021) splits the space randomly. Although the random splitting strategy is efficient, each supernet could contain subnets that are likely to conflict with each other. **Middle:** GM-NAS (Hu et al. 2022) compares gradients of a supernet to split the space, better grouping subnets. This however incurs a lot of computational cost. **Right:** We propose to count the number of nonlinear functions within a subnet such that each subspace contains subnets with the same number of nonlinear functions only. Our splitting criterion incurs negligible overheads, while separating the space effectively. Best viewed in color.

*Corresponding author.

divide a search space into multiple subspaces, and then train an individual supernet for each subspace. Since each supernet has its own parameters, subnets from different supernets do not interfere with each other during training. This allows the estimated performance of subnets to become more correlated with the actual one, which however requires at least a few times more computational cost than one-shot methods.

We introduce in this paper a novel few-shot NAS method that splits the search space in an efficient manner, while better alleviating the conflicts between subnets. Specifically, we leverage the number of nonlinear functions (*e.g.*, ReLU (Krizhevsky, Sutskever, and Hinton 2012)) within a particular subnet to split the search space such that each subspace contains subnets with the same number of nonlinear functions only. We then assign a separate supernet to each subspace to prevent subnets from different supernets from interfering with each other. In particular, our method results in subnets from the same supernet having similar characteristics in terms of the number of parameters, FLOPs, and test accuracy, making them less likely to suffer from the conflicts. Different from current few-shot NAS methods (See Fig. 1), our splitting criterion is more efficient, since it does not require comparing gradients of a supernet to split the search space (Hu et al. 2022) and a specific technique for initializing supernets (Zhao et al. 2021; Hu et al. 2022). In addition, we have found that effectively dividing the search space helps to maintain the performance ranking between subnets, even when the number of channels varies. Based on this observation, we propose to adjust the number of channels for each supernet to further improve the efficiency, allowing us to train all supernets on a single machine in contrast to existing few-shot methods. We also introduce a supernet-balanced sampling (SBS) technique for better training supernets. Since each supernet has a different number of subnets, randomly sampling one of the subnets at each training step would be biased towards training the supernet with the largest number of subnets. Instead of selecting a single subnet at each training step, our SBS samples multiple subnets from different supernets. This enables training different supernets evenly within a limited number of training steps. Extensive experiments on NAS201 (Dong and Yang 2020) and ImageNet (Deng et al. 2009) demonstrate that our approach achieves state-of-the-art results with much less computational overheads than other few-shot methods (Zhao et al. 2021; Hu et al. 2022; Su et al. 2021). We summarize our main contributions as follows:

- We introduce a simple yet effective method using the number of nonlinear functions to split the space. Our method enables reducing the number of channels for each supernet, providing much smaller computational cost, compared with current few-shot methods.
- We present a SBS scheme, where multiple subnets are sampled from different supernets at each training step, to train different supernets equally.
- We demonstrate the effectiveness of our approach on NAS201 (Dong and Yang 2020) and ImageNet (Deng et al. 2009), and provide extensive experiments along with ablation studies.

Related Work

NAS aims to find a well-performing network in a search space efficiently. To this end, the search space, typically defined as the total number of layers and a set of candidate operations (*e.g.*, convolutional or pooling layers), should cover various network architectures (Zhou et al. 2021; Ci et al. 2021; Radosavovic et al. 2020; Yu et al. 2020), and the searching process should be effective and efficient (Guo et al. 2020; You et al. 2020; Chu, Zhang, and Xu 2021). In the following, we describe representative methods pertinent to ours.

One-Shot NAS. Many NAS methods adopt a weight-sharing strategy that trains an over-parameterized network (*i.e.*, a supernet) containing all possible operations at each layer in order to reduce the search time. The seminal work of (Liu, Simonyan, and Yang 2019) proposes to compute a weighted average of feature maps at each layer, where each feature map is obtained from a corresponding operation. It thus needs to train all operations at training time, resulting in a large memory footprint. To reduce the memory requirement, several methods (Guo et al. 2020; Pham et al. 2018; You et al. 2020; Lu et al. 2023) propose to sample and train a single operation at each layer only (*i.e.*, a subnet). For example, SPOS (Guo et al. 2020) uniformly samples one of the subnets at each training step and updates corresponding operations only, reducing the computational cost. Instead of equally treating all subnets, GreedyNAS (You et al. 2020) focuses more on sampling subnets that are likely to perform better than others, but this requires evaluating a set of candidate subnets on a validation set repeatedly. On the other hand, FairNAS (Chu, Zhang, and Xu 2021) attempts to sample every operation at each layer more evenly. To this end, it selects several subnets at each training step and updates entire operations for the subnets simultaneously. Our SBS is similar to FairNAS in that it samples multiple subnets at each training step. On the contrary, SBS aims at training different supernets evenly, preventing the supernet with the largest number of subnets from being sampled more during training. Specifically, SBS forces subnets to be sampled from different supernets, while the subnets sampled from FairNAS belong to the same supernet.

Few-Shot NAS. A few methods have been introduced to use multiple supernets in NAS, which can be divided into two groups depending on whether they split a search space or not. First, recent methods (Zhao et al. 2021; Hu et al. 2022) propose to split a search space into a set of subspaces and then train a separate supernet for each subspace. For example, GM-NAS (Hu et al. 2022) formulates splitting the search space as a graph clustering problem. To be specific, it first computes gradients for individual operations at each layer, and measures cosine similarity between all pairs of the gradients. It then applies the graph min-cut algorithm (Stoer and Wagner 1997) with setting the cosine similarity as a cut cost, and splits the search space so that subnets from the same subspace are less likely to interfere with each other. GM-NAS however requires training a single supernet that covers the entire search space to initialize subsequent supernets, incurring extra overheads. Additionally, it trains a set of supernets in a sequential manner, due to the large com-

putational cost. Our approach also divides the search space into a set of subspaces, but differs in that (1) it does not train the single supernet covering the entire search space for initialization and (2) it allows to train all supernets simultaneously. Second, K-shot NAS (Su et al. 2021) proposes to duplicate parameters of each operation K times (*i.e.*, K copies of the supernet) without splitting the search space. In particular, it produces a subnet by computing a weighted average of the duplicated parameters, rather than sampling a single operation at each layer as in other methods (Guo et al. 2020; Zhao et al. 2021; Hu et al. 2022). Specifically, a generator is trained along with the copies of the supernet to produce K-dimensional probabilities for the weighted average. Similar to ours, K-shot NAS trains supernets simultaneously and does not require a specific scheme for initializing supernets. Although all the aforementioned methods for few-shot NAS alleviate the conflicts between subnets at training time, they are computationally expensive compared to one-shot approaches. Differently, based on our finding that effectively dividing the search space enables preserving the performance ranking between subnets, our approach reduces the number of channels for individual supernets to train multiple supernets efficiently. Note that this is different from early methods (Liu, Simonyan, and Yang 2019; Xu et al. 2020) that simply use a smaller supernet due to their unacceptable overhead.

Zero-Shot NAS. Several methods have been introduced to avoid training supernets. They rely on training-free measurements, typically referred to as zero-cost proxies, to evaluate the performance of each subnet. For example, inspired by neural tangent kernels (NTKs) (Jacot, Gabriel, and Hongler 2018; Lee et al. 2019) representing training dynamics of a neural network, the works of (Chen, Gong, and Wang 2021; Mok et al. 2022; Xu et al. 2021) employ the condition number of a NTK for each subnet as its trainability. However, calculating a NTK for each subnet is computationally demanding. Another line of work (Mellor et al. 2021; Chen, Gong, and Wang 2021) instead uses the number of linear regions divided by ReLU activations during the forward pass of a network to measure the representational power of the network (Hanin and Rolnick 2019a,b). While computing the number of linear regions is efficient in that it requires a single forward pass only, it cannot be applied for the case that neural networks adopt other activations (*e.g.*, a tanh function). Recently, AZ-NAS (Lee and Ham 2024) introduces a new measurement called an isotropy of a feature space. As the isotropy is obtained by computing similarities between intermediate features, it is applicable regardless of activation functions. Similar to AZ-NAS, our approach could be applicable to various activation functions, since it needs to count the number of nonlinear functions only. We differ from all the aforementioned methods in that our approach to counting the number of nonlinear functions focuses on dividing the search space for few-shot NAS rather than accurately measuring the performance of each subnet. In addition, our splitting criterion does not require processing forward and backward passes, which are computationally expensive for splitting the search space.

Method

In this section, we describe a weight-sharing technique for NAS briefly, and introduce our approach to using the number of nonlinear functions within a network to divide a search space. We then describe a SBS technique and how to sample optimal network architectures.

Problem Statement

Let us suppose a search space $\mathcal{A}$ consisting of subnets a_n as follows:

$$\mathcal{A} = \{a_n \mid n = 1, 2, \ldots, N\}, \tag{1}$$

where N is the total number of subnets. To avoid training individual subnets from scratch, we adopt a weight-sharing technique that employs an over-parameterized network (*i.e.*, a supernet) containing all candidate operations at each layer. The parameters of each subnet a_n are determined by selecting a corresponding operation at each layer of the supernet. Formally, we define the learnable parameters of the i-th operation at the j-th layer as follows:

$$w(i,j) \in \mathbb{R}^{C_{\text{out}}(i,j) \times C_{\text{in}}(i,j) \times s(i,j) \times s(i,j)}, \tag{2}$$

where C_{out} and C_{in} are the numbers of output and input channels, respectively, and s is the size of the filter. The parameters of the supernet for the entire search space $\mathcal{A}$ can then be defined as follows:

$$\mathcal{W}(\mathcal{A}) = \bigcup_{i,j} w(i,j). \tag{3}$$

The weight-sharing scheme reduces the computational cost significantly, since we train the single supernet only. It however suffers from conflicts between subnets at training time, as all architectures in the search space share the same set of parameters $\mathcal{W}(\mathcal{A})$. To reduce the conflicts between subnets, few-shot NAS methods (Zhao et al. 2021; Hu et al. 2022) propose to limit the extent of weight sharing by splitting the search space into subspaces and assigning an individual supernet to each subspace. Namely, subnets from different supernets do not share parameters. It is thus important to partition the search space effectively to minimize the interference between subnets sharing the same supernet. In the following, we describe our approach to splitting the search space in detail.

Division Using the Number of Nonlinear Functions

Analysis of zero-cost proxies. To verify using the number of nonlinear functions as a splitting criterion, we present in Figs. 2 and 3 a statistical analysis of subnets on NAS201 (Dong and Yang 2020) that provides standalone accuracies of the subnets computed by training them from scratch. Specifically, we explore four zero-cost proxies to divide a search space into a set of disjoint subspaces: FLOPs (Ning et al. 2021; Li et al. 2023), the number of linear regions (Mellor et al. 2021), an isotropy of a feature space (Lee and Ham 2024), and the number of nonlinear functions. The proxy-based criteria provide three supernets on NAS201, while FS-NAS (Zhao et al. 2021) using a random splitting criterion employ five supernets. We can see in Fig. 2 that the criteria using the zero-cost proxies

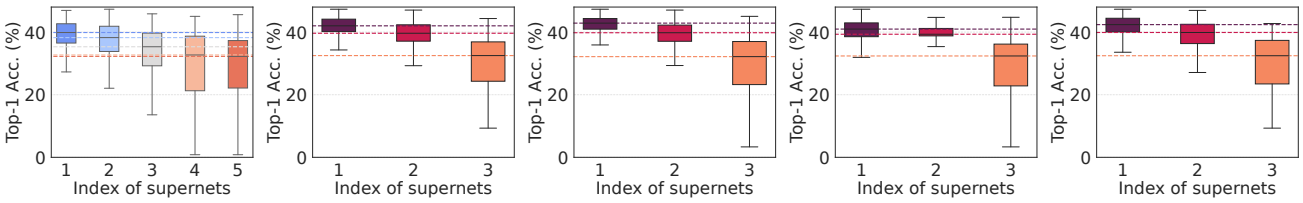


Figure 2: Distributions of top-1 test accuracies for subnets of each supernet on ImageNet-16-120 (Chrabaszcz, Loshchilov, and Hutter 2017) of NAS201 (Dong and Yang 2020). Each dotted line represents the median value for the corresponding distribution. (a) FS-NAS (Zhao et al. 2021) adopts five supernets, dividing the search space randomly. (b-e) We leverage zero-cost proxies to divide the space, resulting in three supernets. They are (from left to right) FLOPs (Ning et al. 2021; Li et al. 2023), the number of linear regions (Mellor et al. 2021), an isotropy of a feature space (Lee and Ham 2024), and the number of nonlinear functions. Best viewed in color.

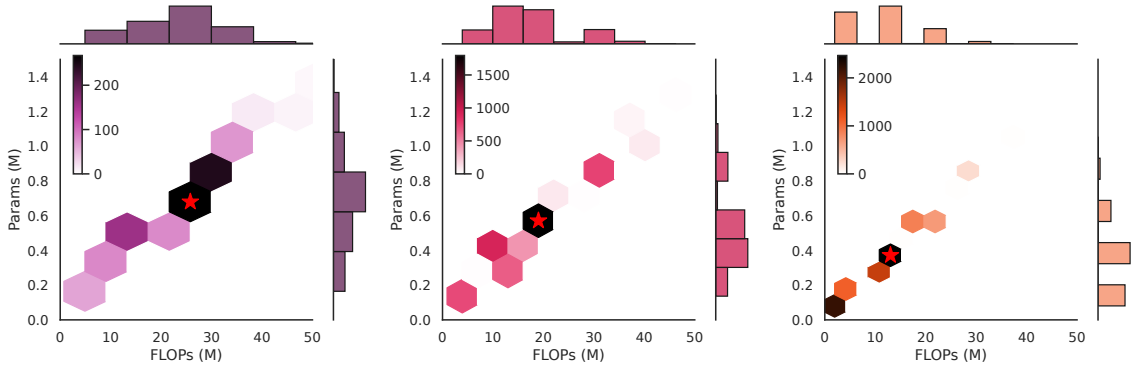


Figure 3: Histograms of two variables (*i.e.*, FLOPs and the number of parameters) for subnets of each supernet, where a red star indicates a bin with the highest frequency. We can see that our splitting criterion makes subnets from different supernets have unique structures in terms of FLOPs and the number of parameters. This suggests that our approach enables better training subnets, since the subnets with similar structures are less likely to suffer from the conflicts, compared with the ones with different structures. Best viewed in color.

show a clear difference between the supernets in terms of the median accuracy of the subnets, compared to the random splitting strategy in FS-NAS. This implies that the proxy-based criteria split the space effectively. In particular, it is worth noting that the criteria using other zero-cost proxies (*i.e.*, FLOPs, the number of linear regions, and the feature isotropy) require a single forward pass for each subnet, while our criterion does not need the computation of forward passes. Considering that the number of subnets is typically innumerable, our criterion is more suitable for splitting the space. In addition, Fig. 3 shows that our splitting criterion allows subnets from different supernets to have unique structures in terms of FLOPs and the number of parameters. This helps to mitigate conflicts between the subnets belonging to the same supernet, as subnets with similar structures are less likely to interfere with each other than those with different structures. Please refer to Sec. 3 in the supplementary material for a more detailed analysis.

Supernets and subspaces. Here we introduce a simple yet effective criterion that splits the search space into a set of disjoint subspaces. Concretely, with a function $D(\cdot)$ counting the number of nonlinear functions (*e.g.*, ReLU (Krizhevsky, Sutskever, and Hinton 2012)) within a subnet, we set a subspace whose subnets have the same number of nonlinear functions, *i.e.*, k, as follows:

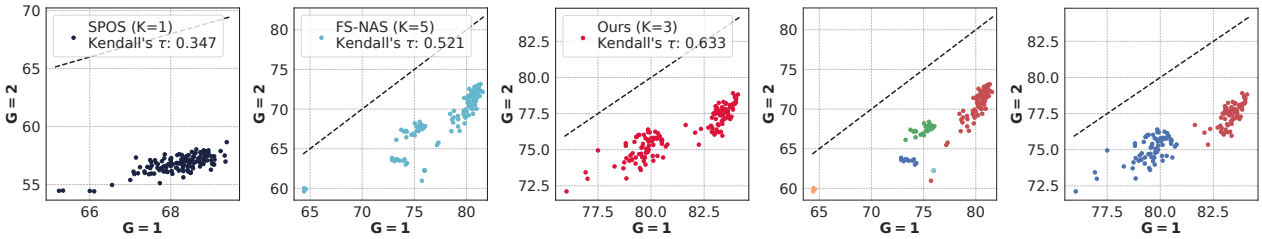
$$\mathcal{A}_k = \{a_n \mid D(a_n) = k \text{ and } n = 1, 2, \ldots, N_k\}, \quad (4)$$

where $\mathcal{A} = \bigcup_k \mathcal{A}_k$ and $N = \sum_k N_k$. We then assign a separate set of parameters for each subspace as follows:

$$\mathcal{W}(\mathcal{A}_k) = \bigcup_{i,j} w_k(i,j), \quad (5)$$

where we denote by $w_k(i,j)$ the parameters of the i-th operation at the j-th layer for the supernet covering the subspace $\mathcal{A}_k$. In this way, we can prevent subnets from different subspaces from interfering with each other, allowing us to better train the subnets.

Channel adjustment. Exploiting multiple supernets alleviates the interference between subnets remarkably, but at the expense of more computational costs than one-shot NAS methods (Guo et al. 2020; Chu, Zhang, and Xu 2021; You et al. 2020). Let us suppose that we have K supernets in total. Then, the total amount of parameters becomes K times more than that of one-shot methods, making it difficult to train supernets simultaneously on a single machine. As a result, existing few-shot methods (Zhao et al. 2021; Hu et al.



(a) SPOS. (b) FS-NAS. (c) Ours. (d) FS-NAS. (e) Ours.

Figure 4: Analysis of the performance of subnets with varying the number of channels for supernets. The x-axis shows accuracies of subnets sampled from supernets using full channel dimensions (*i.e.*, G=1), while the y-axis represents those sampled from supernets with reduced channel (*i.e.*, G=2). We measure the rank correlation in terms of Kendall's tau scores, particularly for high-performing subnets (*i.e.*, top 150 subnets). (a-c) We have observed that exploiting multiple supernets enables better preserving the performance ranking. (d-e) We speculate that this is because the performance ranking among subnets from different supernets is likely to be maintained, as the subnets belonging to different supernets do not interfere with each other. Note that we highlight each supernet in a different color. Best viewed in color.

2022) typically train supernets in a sequential manner, which is time-consuming. A straightforward way to handle this problem is to reduce the number of channels for each supernet, but the reduced channel dimensions could have detrimental effects on NAS as shown in (Liu, Simonyan, and Yang 2019; Xu et al. 2020). To verify this, we compare in Fig. 4 the ranking correlation between two cases: one is with the full channels and the other is with the reduced channels. To this end, we compute Kendall's tau scores (Kendall 1938), particularly for high-performing subnets (*i.e.*, top 150 subnets). We can see in Fig. 4(a) that SPOS (Guo et al. 2020) using a single supernet suffers from the ranking inconsistency after reducing the number of channels. On the contrary, we can see in Figs. 4(b-c) that both FS-NAS (Zhao et al. 2021) and our method better preserve the performance ranking between the high-performing subnets. This suggests that effectively dividing the search space provides the robustness to reducing the channel dimensions. A plausible reason could be that the performance ranking between subnets sampled from different supernets is likely to be maintained (See Figs. 4(d-e)).

Based on this observation, we propose to reduce the number of channels required for each supernet as follows:

$$\mathcal{W}_G(\mathcal{A}_k) = \bigcup_{i,j} w_k(G,i,j), \tag{6}$$

where we denote by G a hyperparamter adjusting the number of channels for each operation, and we define the parameters of the i-th operation at the j-th layer with reduced channel dimensions as follows:

$$w_k(G,i,j) \in \mathbb{R}^{\frac{C_{\text{out}}(i,j)}{G} \times \frac{C_{\text{in}}(i,j)}{G} \times s(i,j) \times s(i,j)}, \tag{7}$$

which enables training supernets simultaneously on a single machine. The comparison between our approach and one-shot NAS methods in terms of the total amount of parameters can be represented as follows:

$$\frac{K|\mathcal{W}_G(\mathcal{A}_k)|}{|\mathcal{W}(\mathcal{A})|} = \frac{K|w_k(G,i,j)|}{|w(i,j)|} = \frac{K}{G^2}. \tag{8}$$

Note that the number of parameters for our method is comparable to that of one-shot methods by setting the proper value of G (*e.g.*, G=2 if K=4).

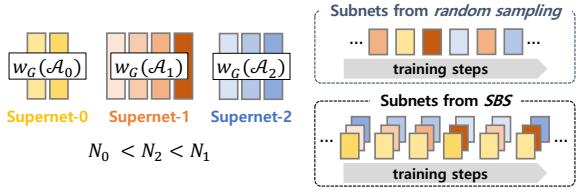


Figure 5: **Left:** We visualize three supernets, where the supernet colored in orange has the largest number of subnets N_1. **Right:** Randomly sampling a single subnet from the entire space $\mathcal{A}$ (=$\mathcal{A}_0 \cup \mathcal{A}_1 \cup \mathcal{A}_2$) at each training step causes the training imbalance between supernets. That is, the training process is biased towards sampling subnets from the space of $\mathcal{A}_1$ (top). Instead, our SBS samples multiple subnets at each training step, where each subnet is sampled from a different supernet (bottom). This allows us to train supernets evenly within a finite number of training steps. Best viewed in color.

Training With SBS

A straightforward way to train supernets is to sample a single subnet randomly from the entire search space $\mathcal{A}$ at each training step as in (Guo et al. 2020; You et al. 2020; Zhao et al. 2021; Hu et al. 2022). However, this might be problematic in that the total number of subnets within each supernet varies significantly (Fig. 5 (left)). That is, subnets sampled from the entire space are likely to belong to the supernet with the largest number of subnets (Fig. 5 (right-top)), causing the training imbalance between supernets. To address this, we propose to sample one subnet from each supernet, resulting in a total of K subnets at each training step (Fig. 5 (right-bottom)). Concretely, we optimize the following objective at each training step:

$$\sum_k \mathcal{L}_{\text{tr}}(a_n; \mathcal{W}_G(\mathcal{A}_k)), \tag{9}$$

which aggregates each training loss $\mathcal{L}_{\text{tr}}$ using the subnet a_n whose parameters are inherited from the corresponding supernet $\mathcal{W}_G(\mathcal{A}_k)$. This allows us to train all supernets equally during training, avoiding the training imbalance.

Searching

After training supernets, we search the best-performing subnet from the search space. To this end, we estimate the performance of a subnet by inheriting its parameters from a corresponding supernet. Concretely, we can obtain the optimal architecture a_* as follows:

$$a_* = \underset{a_n \in \mathcal{A}}{\operatorname{argmin}} \mathcal{L}_{\text{val}}(a_n; \mathcal{W}_G(\mathcal{A}_m)), \tag{10}$$

where $\mathcal{L}_{\text{val}}$ is a validation loss and we denote by m the number of nonlinear functions within the sampled subnet a_n. However, traversing all subnets is infeasible, since the total number of subnets N is typically innumerable (*e.g.*, $6^6 \times 7^{15}$ in the MobileNet search space (Cai, Zhu, and Han 2019; Sandler et al. 2018)). Following the common practice (Guo et al. 2020; Su et al. 2021), we exploit an evolutionary search algorithm as an efficient compromise.

Experiments

Implementation Details

Datasets and search spaces. We perform experiments on standard NAS benchmarks for image classification: CIFAR10 (Krizhevsky, Hinton et al. 2009) and ImageNet (Krizhevsky, Sutskever, and Hinton 2012). CIFAR10 provides 50K training and 10K test samples for 10 object classes, while ImageNet consists of 1.2M training and 50K validation samples for 1K object classes. Following the standard protocol in (Liu, Simonyan, and Yang 2019; Xu et al. 2020; Guo et al. 2020), we split the training set of CIFAR10 in half and use each for training and validation, respectively. For ImageNet, we sample 50K images from the training set to construct a new validation set, and use the original validation set for testing. We adopt NAS201 (Dong and Yang 2020) and MobileNet (Cai, Zhu, and Han 2019; Sandler et al. 2018) search spaces for CIFAR10 and ImageNet, respectively. Specifically, NAS201 is a micro search space using a fixed cell structure, where each cell has five operations with six layers (*i.e.*, edges), resulting in 5^6 architectures. For ImageNet, we use a macro search space consisting of $6^6 \times 7^{15}$ architectures in total. We use the number of nonlinear functions to split a search space, unless otherwise specified. Since both NAS201 and MobileNet search spaces have candidate operations using ReLU (Krizhevsky, Sutskever, and Hinton 2012) as a nonlinear function, we thus count the number of ReLU functions within a subnet. Note that our splitting criterion is applicable regardless of the types of activation functions. Please see the supplementary material for a detailed description of the candidate operations.

Training and evaluation. We follow the common practice (Zhao et al. 2021; Hu et al. 2022) for training supernets and retraining optimal architectures. Specifically, we train supernets for 200 epochs with a batch size of $1,024$. We adopt a SGD optimizer with an initial learning rate of 0.12, a momentum of 0.9, and a weight decay of $4e$-5. The learning rate is adjusted by a cosine annealing strategy without restart. The number of supernets K is set to 3 and 6 on NAS201 and ImageNet, respectively. We set G to 2 for all experiments, unless otherwise specified. After applying the evolutionary search algorithm (Guo et al. 2020), we train the chosen architectures for 450 epochs with a batch size of $1,024$. We use a RMSProp optimizer with an initial learning rate and a weight decay of 0.064 and $1e$-5, respectively. The learning rate decays by a factor of 0.97 per 2.4 epochs. For evaluation, we compute top-1 and top-5 classification accuracies, and report average scores using 3 different seeds for all experiments. All experiments are performed with 8 NVIDIA A5000 GPUs.

Method	K	Params (M)	Kendall's τ
One-shot NAS			
SPOS (Guo et al. 2020)	1	<u>1.7</u>	0.554
AngleNet (Hu et al. 2020)	1	<u>1.7</u>	0.575
Few-shot NAS			
FS-NAS (Zhao et al. 2021)	5	8.4	0.653
GM-NAS (Hu et al. 2022)	8	13.6	0.656
K-shot NAS (Su et al. 2021)	8	13.6	0.626
Ours			
FLOPs	3	**1.3**	0.711
# of linear regions	3	**1.3**	<u>0.712</u>
Feature isotropy	3	**1.3**	0.693
# of nonlinear functions	3	**1.3**	**0.735**

Table 1: Comparison of Kendall's tau scores on the test set of CIFAR10. Our approach outperforms one- and few-shot NAS methods by a large margin, while adopting three supernets with half channel dimensions (*i.e.*, G=2). Note that it requires fewer parameters than one-shot NAS methods. Numbers in bold are the best performance and underlined ones are the second best. Params: the number of parameters required for supernets.

Results

NAS201. We compare in Table 1 our approach with one- and few-shot NAS methods on the test set of CIFAR10 (Krizhevsky, Hinton et al. 2009). Specifically, we measure the Kendall rank correlation (Kendall 1938) between stand-alone accuracies of subnets provided by NAS201 (Dong and Yang 2020) and estimated ones from supernets. From this table, we can see that few-shot NAS approaches (Zhao et al. 2021; Su et al. 2021) achieve better results than one-shot baselines, suggesting that reducing the extent of weight sharing improves the rank consistency remarkably. We can also see that our approach, regardless of the types of zero-cost proxies, outperforms other few-shot methods by a large margin, demonstrating its effectiveness. This is particularly significant in that we require fewer parameters for supernets than one-shot NAS methods, while the total number of parameters for current few-shot methods increases with the number of supernets. Among the zero-cost proxies, using the number of nonlinear functions gives the best performance, suggesting that it groups subnets effectively. We show in Table 2 top-1 test accuracies of chosen architectures. We can see that our approach provides the high-performing architectures on each dataset of NAS201. This validates once again that our splitting criterion is simple yet effective.

Method	C10	C100	IN
One-shot NAS			
DARTS (Liu, Simonyan, and Yang 2019)	54.30	15.61	16.32
PC-DARTS (Xu et al. 2020)	93.41	67.48	41.31
SPOS (Guo et al. 2020)	93.67	69.83	44.71
AngleNet (Hu et al. 2020)	94.01	72.96	45.83
AGNAS (Sun et al. 2022)	94.05	72.41	45.98
Few-shot NAS			
FS-NAS (Zhao et al. 2021)	89.11	58.69	33.85
GM-NAS (Hu et al. 2022)	92.70	68.81	43.47
K-shot NAS (Su et al. 2021)	94.19	**73.45**	46.53
Ours	**94.30**	73.20	**46.60**
Upper bound	94.37	73.51	47.31

Table 2: Test accuracies of searched architectures on NAS201. Different from current few-shot methods using multiple supernets with full channel dimensions, our method uses three supernets with half channel dimensions (*i.e.*, G=2). C10, C100, and IN represent CIFAR10, CIFAR100, and ImageNet-16-120, respectively.

MobileNet. We report in Table 3 top-1 and top-5 accuracies of our architectures chosen from the MobileNet search space. To this end, we perform the evolutionary search (Guo et al. 2020) using FLOPs as a hardware constraint. We can see from this table that our method with a constraint of 530M FLOPs provides better results than FS-NAS (Zhao et al. 2021) and GM-NAS (Hu et al. 2022) in terms of test accuracy, FLOPs, and the number of parameters. This is remarkable in that our method exploits six supernets with reduced channel dimensions to alleviate the computational cost, while FS-NAS and GM-NAS adopt five and six supernets with full channel dimensions, respectively. We can also see that our architecture searched with a constraint of 600M FLOPs achieves the highest test accuracy. This suggests the importance of effectively dividing the search space to improve the search performance.

Analysis of the numbers of supernets. We provide in Table 4 results for the selected architectures with varying the values of K on ImageNet (Krizhevsky, Sutskever, and Hinton 2012). For comparison, we report the results of FS-NAS (Zhao et al. 2021) and GM-NAS (Hu et al. 2022). We can see from this table three things: (1) Similar to FS-NAS, our splitting criterion induces a negligible overhead. On the contrary, GM-NAS incurs a lot of computational cost particularly for splitting the search space. In addition, the training cost of GM-NAS is approximately 3.5 times higher than our method using the same number of supernets (*i.e.*, K=6). This is because GM-NAS trains supernets sequentially due to the large memory requirements. (2) We can see from the last three rows that using more supernets provides better results in terms of test accuracy at the cost of increasing the training time. For example, our method using eight supernets gives an accuracy gain of 0.2% over GM-NAS with a much lower training cost. (3) We can save the training cost remarkably by reducing the channel dimensions for supernets (*i.e.*, G=2). For example, setting G to 2 for training six supernets induces the training time comparable to FS-NAS, while outperforming GM-NAS in terms of test accuracies.

Method	Acc. (%)		FLOPs	Params
	Top-1	Top-5	(M)	(M)
One-shot NAS				
GAEA (Li et al. 2021)	76.0	92.7	-	5.6
SPOS (Guo et al. 2020)	74.7	-	328	3.4
ProxylessNAS (Cai, Zhu, and Han 2019)	75.1	92.5	465	7.1
AngleNet (Hu et al. 2020)	76.1	-	470	-
Shapley-NAS (Xiao et al. 2022)	76.1	-	582	5.4
PC-DARTS (Xu et al. 2020)	75.8	92.7	597	5.3
DrNAS (Chen et al. 2021)	76.3	92.9	604	5.7
ISTA-NAS (Yang et al. 2020)	76.0	92.9	638	5.7
Few-shot NAS				
FS-NAS (Zhao et al. 2021)	75.9	-	521	4.9
GM-NAS (Hu et al. 2022)	76.6	93.0	530	4.9
Ours ($\leq$530M)	76.7	**93.2**	516	4.8
Ours ($\leq$600M)	**76.9**	**93.2**	544	4.9

Table 3: Quantitative results of searched architectures on ImageNet. We use two constraints in terms of FLOPs for the evolutionary search algorithm (Guo et al. 2020). FS-NAS (Zhao et al. 2021) and GM-NAS (Hu et al. 2022) exploit five and six supernets with full channel dimensions, respectively, while our method adopts six supernets with half channel dimensions (*i.e.*, G=2). Params: the number of network parameters for the chosen architecture.

Method	Cost		Top-1	FLOPs
	splitting	training	Acc. (%)	(M)
FS-NAS (Zhao et al. 2021)	-	23.1	75.9	521
GM-NAS (Hu et al. 2022)	17.0	81.0	76.6	530
Ours: K=4 & G=2	-	17.3	76.5	527
Ours: K=6 & G=2	-	23.3	76.7	516
Ours: K=8 & G=2	-	34.1	76.8	522

Table 4: Analysis of different values of K on ImageNet. We report the computational cost in terms of GPU days along with the top-1 accuracies of the searched architectures. Specifically, we denote by splitting and training the amount of time to split a search space and train supernets, respectively.

Conclusion

We have introduced a novel few-shot NAS method that exploits the number of nonlinear functions to split a search space into a set of disjoint subspaces. Our splitting criterion is simple yet effective, making subnets from different supernets have distinct characteristics in terms of test accuracy, FLOPs, and the number of parameters. Based on our novel observation that effectively dividing the space provides the robustness to reducing the number of channels, we have proposed to adjust the channel dimensions required for supernets to alleviate the computational cost, allowing us to train supernets on a single machine. We have also presented a SBS technique, which samples multiple subnets belonging to different supernets at each training step, to train all supernets evenly within a limited number of training steps. Finally, we have demonstrated the effectiveness of our approach on standard NAS benchmarks.

Acknowledgments

This work was supported in part by the NRF and IITP grants funded by the Korea government (MSIT) (No.2023R1A2C2004306, No.RS-2022-00143524, Development of Fundamental Technology and Integrated Solution for Next-Generation Automatic Artificial Intelligence System), the KIST Institutional Program (Project No.2E31051-21-203), and the Yonsei Signature Research Cluster Program of 2024 (2024-22-0161).

References

Baker, B.; Gupta, O.; Naik, N.; and Raskar, R. 2017. Designing neural network architectures using reinforcement learning. In *ICLR*.

Cai, H.; Zhu, L.; and Han, S. 2019. ProxylessNAS: Direct neural architecture search on target task and hardware. In *ICLR*.

Chen, W.; Gong, X.; and Wang, Z. 2021. Neural architecture search on ImageNet in four gpu hours: A theoretically inspired perspective. In *ICLR*.

Chen, X.; Wang, R.; Cheng, M.; Tang, X.; and Hsieh, C.-J. 2021. DrNAS: Dirichlet neural architecture search. In *ICLR*.

Chrabaszcz, P.; Loshchilov, I.; and Hutter, F. 2017. A downsampled variant of ImageNet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*.

Chu, X.; Zhang, B.; and Xu, R. 2021. FairNAS: Rethinking evaluation fairness of weight sharing neural architecture search. In *ICCV*.

Ci, Y.; Lin, C.; Sun, M.; Chen, B.; Zhang, H.; and Ouyang, W. 2021. Evolving search space for neural architecture search. In *ICCV*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. ImageNet: A large-scale hierarchical image database. In *CVPR*.

Dong, X.; and Yang, Y. 2020. NAS-bench-201: Extending the scope of reproducible neural architecture search. In *ICLR*.

Guo, Z.; Zhang, X.; Mu, H.; Heng, W.; Liu, Z.; Wei, Y.; and Sun, J. 2020. Single path one-shot neural architecture search with uniform sampling. In *ECCV*.

Hanin, B.; and Rolnick, D. 2019a. Complexity of linear regions in deep networks. In *ICML*.

Hanin, B.; and Rolnick, D. 2019b. Deep ReLU networks have surprisingly few activation patterns. In *NeurIPS*.

Hu, S.; Wang, R.; Hong, L.; Li, Z.; Hsieh, C.-J.; and Feng, J. 2022. Generalizing few-shot nas with gradient matching. In *ICLR*.

Hu, Y.; Liang, Y.; Guo, Z.; Wan, R.; Zhang, X.; Wei, Y.; Gu, Q.; and Sun, J. 2020. Angle-based search space shrinking for neural architecture search. In *ECCV*.

Jacot, A.; Gabriel, F.; and Hongler, C. 2018. Neural tangent kernel: Convergence and generalization in neural networks.

Kendall, M. G. 1938. A new measure of rank correlation. *Biometrika*, 30(1/2): 81–93.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet classification with deep convolutional neural networks.

Lee, J.; and Ham, B. 2024. AZ-NAS: Assembling Zero-Cost Proxies for Network Architecture Search. In *CVPR*.

Lee, J.; Xiao, L.; Schoenholz, S.; Bahri, Y.; Novak, R.; Sohl-Dickstein, J.; and Pennington, J. 2019. Wide neural networks of any depth evolve as linear models under gradient descent.

Li, G.; Yang, Y.; Bhardwaj, K.; and Marculescu, R. 2023. ZiCo: Zero-shot nas via inverse coefficient of variation on gradients. In *ICLR*.

Li, L.; Khodak, M.; Balcan, M.-F.; and Talwalkar, A. 2021. Geometry-aware gradient algorithms for neural architecture search. In *ICLR*.

Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: Differentiable architecture search. In *ICLR*.

Lu, S.; Hu, Y.; Yang, L.; Sun, Z.; Mei, J.; Tan, J.; and Song, C. 2023. PA&DA: Jointly Sampling PAth and DAta for Consistent NAS. In *CVPR*.

Mellor, J.; Turner, J.; Storkey, A.; and Crowley, E. J. 2021. Neural architecture search without training. In *ICML*.

Mok, J.; Na, B.; Kim, J.-H.; Han, D.; and Yoon, S. 2022. Demystifying the neural tangent kernel from a practical perspective: Can it be trusted for neural architecture search without training? In *CVPR*.

Ning, X.; Tang, C.; Li, W.; Zhou, Z.; Liang, S.; Yang, H.; and Wang, Y. 2021. Evaluating efficient performance estimators of neural architectures. In *NeurIPS*.

Pham, H.; Guan, M.; Zoph, B.; Le, Q.; and Dean, J. 2018. Efficient neural architecture search via parameters sharing. In *ICLR*.

Radosavovic, I.; Kosaraju, R. P.; Girshick, R.; He, K.; and Dollár, P. 2020. Designing network design spaces. In *CVPR*.

Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; and Chen, L.-C. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *CVPR*.

Stoer, M.; and Wagner, F. 1997. A simple min-cut algorithm. *J. ACM*, 44(4): 585–591.

Su, X.; You, S.; Zheng, M.; Wang, F.; Qian, C.; Zhang, C.; and Xu, C. 2021. K-shot NAS: Learnable weight-sharing for nas with k-shot supernets. In *ICML*.

Sun, Z.; Hu, Y.; Lu, S.; Yang, L.; Mei, J.; Han, Y.; and Li, X. 2022. AGNAS: Attention-Guided Micro and Macro-Architecture Search. In *ICML*.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine learning*.

Xiao, H.; Wang, Z.; Zhu, Z.; Zhou, J.; and Lu, J. 2022. Shapley-NAS: discovering operation contribution for neural architecture search. In *CVPR*.

Xu, J.; Zhao, L.; Lin, J.; Gao, R.; Sun, X.; and Yang, H. 2021. KNAS: green neural architecture search. In *ICML*.

Xu, Y.; Xie, L.; Zhang, X.; Chen, X.; Qi, G.-J.; Tian, Q.; and Xiong, H. 2020. PC-DARTS: Partial channel connections for memory-efficient architecture search. In *ICLR*.

Yang, Y.; Li, H.; You, S.; Wang, F.; Qian, C.; and Lin, Z. 2020. ISTA-NAS: Efficient and consistent neural architecture search by sparse coding.

You, S.; Huang, T.; Yang, M.; Wang, F.; Qian, C.; and Zhang, C. 2020. GreedyNAS: Towards fast one-shot nas with greedy supernet. In *CVPR*.

Yu, J.; Jin, P.; Liu, H.; Bender, G.; Kindermans, P.-J.; Tan, M.; Huang, T.; Song, X.; Pang, R.; and Le, Q. 2020. BigNAS: Scaling up neural architecture search with big single-stage models. In *ECCV*.

Zhao, Y.; Wang, L.; Tian, Y.; Fonseca, R.; and Guo, T. 2021. Few-shot neural architecture search. In *ICML*.

Zhou, D.; Jin, X.; Lian, X.; Yang, L.; Xue, Y.; Hou, Q.; and Feng, J. 2021. AutoSpace: Neural architecture search with less human interference. In *ICCV*.

Zoph, B.; and Le, Q. V. 2017. Neural architecture search with reinforcement learning. In *ICLR*.

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2018. Learning transferable architectures for scalable image recognition. In *CVPR*.