K-shot NAS: Learnable Weight-Sharing for NAS with K-shot Supernets

Xiu Su [*1] **Shan You** [*2 3] **Mingkai Zheng** [2] **Fei Wang** [2] **Chen Qian** [2] **Changshui Zhang** [3] **Chang Xu** [1]

Abstract

In one-shot weight sharing for NAS, the weights of each operation (at each layer) are supposed to be identical for all architectures (paths) in the supernet. However, this rules out the possibility of adjusting operation weights to cater for different paths, which limits the reliability of the evaluation results. In this paper, instead of counting on a single supernet, we introduce K-shot supernets and take their weights for each operation as a dictionary. The operation weight for each path is represented as a convex combination of items in a dictionary with a simplex code. This enables a matrix approximation of the stand-alone weight matrix with a higher rank ($K > 1$). A *simplex-net* is introduced to produce architecture-customized code for each path. As a result, all paths can adaptively learn how to share weights in the K-shot supernets and acquire corresponding weights for better evaluation. K-shot supernets and simplex-net can be iteratively trained, and we further extend the search to the channel dimension. Extensive experiments on benchmark datasets validate that K-shot NAS significantly improves the evaluation accuracy of paths and thus brings in impressive performance improvements.

1. Introduction

Deep neural models have thrived in various walks of life with artificial intelligence and edge computing. Before deploying a model for a task of interest, practitioners need to consider the hardware budgets and specify a decent architecture or structure for the model. In this way, much effort have devoted to designing sophisticated structures, such as ResNet (He et al., 2016), SENet (Hu et al., 2018) and MobileNet (Sandler et al., 2018), and further boosting the performance within the hardware budgets, *e.g.*, knowledge distillation (Hinton et al., 2015; You et al., 2017; Du et al., 2020) and quantization (Gholami et al., 2021). Recently, neural architecture search (NAS) (Tan et al., 2019; Wan et al., 2020; Guo et al., 2020b) merges both sides and aims to directly generate architectures with a promising performance by liberating manual labor and surmounting the cognitive bottleneck of humans.

Pioneer works usually design architectures via a decision-making process with reinforcement learning (Zoph & Le, 2016) or pure evolutionary algorithms (Real et al., 2019). However, they often involve unbearable training costs and resource consumption. For the sake of searching efficiency, current methods mainly leverage a one-shot supernet with all architectures (paths) that can be derived and share their weights mutually. Gradient-based methods introduce architecture parameters to deferentially optimize the weights and architecture parameters, such as DARTS (Liu et al., 2018) and its variants (Xu et al., 2019; Huang et al., 2020; Yang et al., 2020; 2021). But they may suffer the significant memory cost and the detrimental gap when deriving the final architecture. Sample-based methods (You et al., 2020) thus propose to decouple the supernet weights and architecture parameters. They train the supernet by sampling paths from it (Guo et al., 2020b) and then take it as an evaluator for ranking different paths, with the more stable and promising performance achieved.

In one-shot NAS methods, the assumption of weight sharing plays an important role. Specifically, it postulates that the weights of an operation at a layer are identical for all the paths containing this operation. Thus, the training efficiency of the supernet can be largely ensured, as only one copy of the operation weight has to be maintained for the massive number of paths. However, this also rules out the possibility of adjusting the operation weight to cater for different paths. As a result, any two different paths will always be compulsively required to inherit the weights of the same operations. Nevertheless, they may be far from the *stand-alone* weights obtained by training each path from scratch. With such a harsh and limited space to adapt to the optimal operation weight, the resulting evaluation performance

[*]Equal contribution [1]School of Computer Science, Faculty of Engineering, The University of Sydney, Australia [2]SenseTime Research [3]Department of Automation, Tsinghua University, Institute for Artificial Intelligence, Tsinghua University (THUAI), Beijing National Research Center for Information Science and Technology (BNRist). Correspondence to: Shan You <youshan@sensetime.com>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

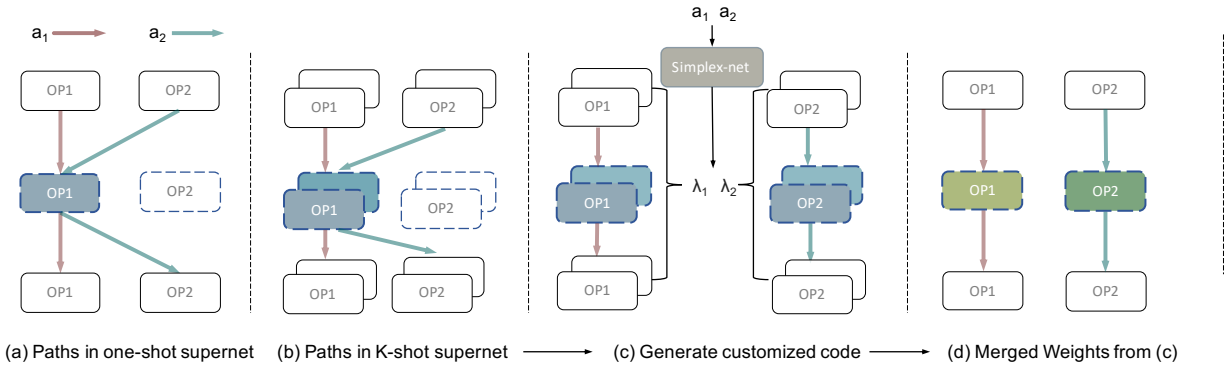

Figure 1. K-shot NAS directly learns the customized code (coefficient vector) $\boldsymbol{\lambda}$ for different paths (subnets or architectures) $\boldsymbol{a}$. Therefore, even with the same operation, *e.g.* OP1 in the second layer, the weights of each path is encouraged to be different from each other as in (d) and be a more accurate approximation for the stand-alone weights trained from scratch. However, as in (a), paths in one-shot supernet fully share a same set of weights even when architectures are different ($\boldsymbol{a}_1$ vs $\boldsymbol{a}_2$).

of paths could be unreliable and rarely reflects the Oracle ranking of different paths for supernet.

In this paper, we introduce a K-shot NAS framework to unleash the potential of sampled subnetworks (subnets). Instead of counting on a single operation weight of supernet, we maintain a dictionary of weights for an operation based on a group of K supernets (see Figure 1). All paths containing a certain operation are supposed to share the dictionary mutually, but we allow them to have their own customized manner to exploit it. Particularly, the operation weight for each path is represented as a convex combination of items in the dictionary with a code on the simplex. We encode the architecture in a code generator *simplex-net* to generate these codes customized for different paths. Then we can interpret the weight sharing as a matrix approximation problem, where the one-shot paradigm merely leverages a rank-1 matrix to approximate the stand-alone weight matrix for all paths while our K-shot case encourages the approximation through a matrix of higher rank. We further consider the joint searching with channel dimension (number of filters for operations). A channel branch can be encoded into the simplex-net to explore the context information or the difference among layers. We propose a non-parametric regularization to improve the discrimination between simplex codes of different widths. Extensive experiments on the benchmark ImageNet (Russakovsky et al., 2015) and NAS-Bench-201 (Dong & Yang, 2020) datasets validate the effectiveness of our K-shot NAS and advantages over other one-shot NAS methods. We can achieve a 77.9% Top-1 accuracy with the MobileNetV2 search space with only 412M FLOPs.

2. Rethinking Weight-Sharing with Supernets

One-shot NAS methods mainly dedicate to optimize a supernet $\mathcal{N}$ which contains all the architectures from a given search space $\mathcal{A}$. The weight $\boldsymbol{\theta}$ of the supernet $\mathcal{N}$ is shared by all its sub-networks (a.k.a architectures or paths), *i.e.*, $\boldsymbol{a} \in \mathcal{A}$. With the trained supernet, the optimal architectures can be identified through searching and evaluating them. Following (Guo et al., 2020b; You et al., 2020), a general one-shot NAS can be formulated as a two-stage optimization problem, *i.e.*, weights optimization for supernets and architecture search,

$$\boldsymbol{a}^* = \arg\max_{\boldsymbol{a}\in\mathcal{A}} \ \mathrm{ACC}_{val}(\boldsymbol{a}, \boldsymbol{\theta}^*_{\mathcal{A}}(\boldsymbol{a})) \tag{1}$$

$$\text{s.t. } \boldsymbol{\theta}^*_{\mathcal{A}} = \arg\min_{\boldsymbol{\theta}} \ \mathcal{L}_{train}(\mathcal{A}, \boldsymbol{\theta}), \tag{2}$$

where ACC_{val} denotes the accuracy for each architecture $\boldsymbol{a}$ on the validation dataset $\mathcal{D}_{val}$, and $\mathcal{L}_{train}(\mathcal{A}, \boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{a}\in\mathcal{A}}[\![\mathcal{L}_{train}(\boldsymbol{a}, \boldsymbol{\theta})]\!]$ denotes the training loss for the supernet on the training dataset $\mathcal{D}_{train}$ by randomly sampling a path and optimizing its weights accordingly. Since it is computationally prohibited to traverse all architectures in a large search space, *e.g.*, 13^{21} possible networks in a MobileNetV2 search space, the optimal architecture search in Eq.(2) usually turns to efficient searching algorithms, such as evolutionary search (Deb et al., 2002; Guo et al., 2020b; You et al., 2020).

2.1. One-shot Weight Sharing

One-shot supernet adopts a fixed weight-sharing paradigm. That is, the weights of an operation at a layer are supposed to be identical (fully-shared) for all sampled architectures that contain this operation. Consider a supernet $\mathcal{N}$ with L layers and O operations for each layer, and then the overall search space size is O^L. Formally, in a one-shot weight sharing strategy, for a certain operation o at a layer, then it will be involved in $N = |\mathcal{A}|/O = O^{L-1}$ possible paths,

which we simply denote as $\{\boldsymbol{a}_1, ..., \boldsymbol{a}_N\}$. Besides, if we train all these N paths from scratch, we can have their Oracle or *stand-alone* weights on the operation o, *i.e.*, $\boldsymbol{W} = [\boldsymbol{w}_1, ..., \boldsymbol{w}_N] \in \mathbb{R}^{d\times N}$. As for the one-shot paradigm, all weights for different paths will amount to the same weight $\boldsymbol{\theta}_o$ of the supernet, and the stand-alone matrix $\boldsymbol{W}$ is assumed to be approximated as

$$\text{one-shot:} \quad \boldsymbol{W} = [\boldsymbol{w}_1, ..., \boldsymbol{w}_N] \approx \mathbf{1}^T \otimes \boldsymbol{\theta}_o, \tag{3}$$

where $\boldsymbol{\theta}_o$ represents the weight of operation o in the supernet, and $\otimes$ is the Kronecker product between two vectors with the vector $\mathbf{1} \in \mathbb{R}^N$ being all ones. The one-shot weight sharing can largely benefit the training efficiency of the supernet, as only one copy of the operation weight has to be maintained for the massive number of architectures. However, the one-shot weight sharing in Eq.(3) rules out the possibility of adjusting the operation weight to cater to different paths. For any two paths $\boldsymbol{a}_i, \boldsymbol{a}_j$ that involve the same operation o from the supernet, they will always share the same weight of o. With such a limited room to match a satisfactory operation weight, the resulting evaluation performance of the paths could be unconvincing and hardly reflects the potential of different paths.

2.2. K-shot Weight Sharing

Instead of relying on the single operation weight $\boldsymbol{\theta}_o$ from the supernet, we tend to maintain a dictionary of weights for an operation. Suppose there are K supernets $\{\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_k\}$. In terms of the operation o, we can set up a dictionary of weights $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_o^1, \cdots, \boldsymbol{\theta}_o^K\}$ based on these K supernets. For simplicity, we will omit the subscript o in the sequel. The dictionary $\boldsymbol{\Theta}$ can be shared by different paths that contain the operation o, but we allow them to have their own way to exploit the dictionary of operation weights. In particular, for the i-th path, we calculate the operation weight through a convex combination of items in the dictionary,

$$\boldsymbol{w}_i \approx \boldsymbol{\Theta}\boldsymbol{\lambda}_i = \sum_{k=1}^{K} \lambda_{i,k}\boldsymbol{\theta}_k, \tag{4}$$

where $\boldsymbol{\lambda}_i$ stands for the combination code of the i-th path and lies on the simplex Δ^{K-1},

$$\Delta^{K-1} = \{\boldsymbol{\lambda} | \sum_{k=1}^{K} \lambda_k = 1, \lambda_k \geq 0, \ \forall\, k \in \{1, \cdots, K\}\}.$$

By simultaneously considering all N paths that involve the operation o, we can approximate their operation weights as

$$K\text{-shot:} \quad \boldsymbol{W} = [\boldsymbol{w}_1, ..., \boldsymbol{w}_N] \approx \boldsymbol{\Theta}\boldsymbol{\Lambda}, \tag{5}$$

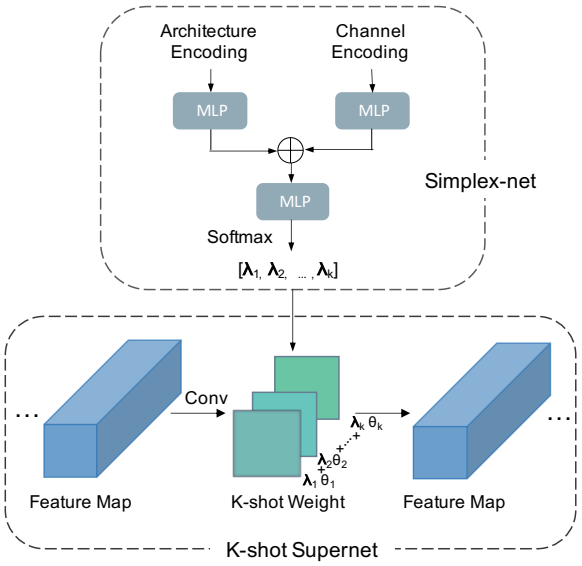

Figure 2. The framework of simlex-net and K-shot Supernet. With simplex-net, each architecture learns the customized code for constructing new supernets with K-shot weight, thus bridging the gap between supernets and stand-alone training.

where $\boldsymbol{\Theta}$[1] serve as a weight basis for all paths, and $\boldsymbol{\Lambda}$ indicates the customized code for paths $\{\boldsymbol{a}_1, ..., \boldsymbol{a}_N\}$. In this way, all architectures can inherit the weights from the same set of K supernets simultaneously, yet with their associated code $\boldsymbol{\lambda}$. We thus name this paradigm ***K-shot weight-sharing***.

Actually, the weight sharing paradigm can be interpreted as a matrix approximation problem (Tropp et al., 2017). As for one-shot case in Eq.(3), it leverages a rank-1 matrix $\mathbf{1}^T \otimes \boldsymbol{w}$ to approximate the optimal stand-alone weight matrix $\boldsymbol{W}$ for various paths. In contrast, we encourage an approximation of $\boldsymbol{W}$ through a bit higher rank matrix. Since $d \ll N$ in the size of $\boldsymbol{W}$, we can leverage a rank K $(1 \leq K \leq d)$ to fulfill the approximation, and it can be written as the multiplication of two small matrices, $\boldsymbol{\Theta} \in \mathbb{R}^{d\times K}$ and $\boldsymbol{\Lambda} \in \mathbb{R}^{K\times N}$ in Eq.(5). By dint of the low-rank approximation (De Silva & Lim, 2008; Kishore Kumar & Schneider, 2017; Clarkson & Woodruff, 2017), the gap between the ideal weight matrix and its approximation can be reduced compared to the one-shot paradigm. The classical rank-1 assumption is thus taken as a special case of ours. Note that the approximation in Eq.(5) depends on the optimality of the combination code $\boldsymbol{\Lambda}$. If all combination code $\boldsymbol{\lambda}$ are nearly the same, then K-shot supernets will degrade to a one-shot case with $\boldsymbol{w} = \boldsymbol{\Theta}\boldsymbol{\lambda}$. We thus hope the combination code of paths can be different and customized to reflect the characteristics of their architectures.

[1]Since $\boldsymbol{\Theta}$ indicates the weights of any operation from K supernets. Without loss of generality, we also use $\boldsymbol{\Theta}$ to indicate the weights of K supernets $\{\boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_k\}$ accordingly.

3. NAS with K-shot Supernets

As illustrated before, with K-shot supernets, weights of paths can have a better approximation to the ideal standalone weight matrix than one-shot paradigm, thus promoting the evaluation of paths to reflect their actual potential. Formally, by examining Eq.(4), each code $\boldsymbol{\lambda}_i$ can be seen as a point on the simplex to weigh the K supernets. The weight of an operation shall not be only determined by itself (chosen or not). Instead, the architecture $\boldsymbol{a}$ of the subnets as a kind of context information should influence the optimal weight of an individual operation. We thus re-parameterize $\boldsymbol{\lambda}_i$ by architecture $\boldsymbol{a}_i$ explicitly via a *simplex-net* π on simplex Δ^{K-1}, *i.e.*,

$$\pi : \mathcal{A} \to \Delta^{K-1}, \quad \boldsymbol{a}_i \mapsto \pi(\boldsymbol{a}_i; \boldsymbol{\sigma}) = \boldsymbol{\lambda}_i, \tag{6}$$

where $\boldsymbol{\sigma}$ is the parameters of the simplex-net, as shown in Figure 2.2, the simplex-net can be modeled with a simple multi-layer perception (MLP) that takes the one-hot architecture encoding vector $\boldsymbol{a}$ as the input to generate its customized code $\boldsymbol{\lambda}$. The objective function of K-shot supernets can now be formulated as follows:

$$\begin{aligned} \boldsymbol{a}^* &= \arg\max_{\boldsymbol{a}\in\mathcal{A}} \ \mathrm{ACC}_{val}(\boldsymbol{a}, \widetilde{\boldsymbol{\theta}}(\boldsymbol{\Theta}^*_{\mathcal{A}}, \boldsymbol{\sigma}^*_{\mathcal{A}}, \boldsymbol{a})), \\ \text{s.t. } \boldsymbol{\Theta}^*_{\mathcal{A}}, \boldsymbol{\sigma}^*_{\mathcal{A}} &= \arg\min_{\boldsymbol{\Theta},\boldsymbol{\sigma}} \ \mathcal{L}_{train}(\mathcal{A}, \widetilde{\boldsymbol{\theta}}(\boldsymbol{\Theta}, \boldsymbol{\sigma}; \mathcal{A})), \\ \widetilde{\boldsymbol{\theta}}(\boldsymbol{\Theta}, \boldsymbol{\sigma}; \mathcal{A}) &= \boldsymbol{\Theta} \cdot \boldsymbol{\lambda}_{\mathcal{A}} = \boldsymbol{\Theta} \cdot \pi(\mathcal{A}; \boldsymbol{\sigma}), \end{aligned} \tag{7}$$

where $\widetilde{\boldsymbol{\theta}} = \sum_{k=1}^{K} \lambda_k \boldsymbol{\theta}_k$ indicates the merged new weights from those of K supernets $\boldsymbol{\Theta}$ and customized code $\boldsymbol{\lambda}$. $\boldsymbol{\Theta}^*_{\mathcal{A}}, \boldsymbol{\sigma}^*_{\mathcal{A}}$ is the optimal weights for the supernet and simplex-net after training. As a result, we can learn how all architectures share their weights with K-shot supernets by the learned customized code adaptively. In this way, the approximation gap can be reduced compared to the one-shot paradigm, and the evaluation ability is expected to be boosted.

Iterative training of supernets and simplex-net. Commonly, the supernet is trained by sampling a path and a mini-batch b of images per step and optimizing the corresponding weights. However, with the introduced simplex-net, this end-to-end training might bring in difficulty for the simplex-net since the valid batch size for simplex-net amounts to 1 (one path per step). In this way, we propose to use a larger path size for the simplex-net but optimize in an iterative manner. In detail, we choose to optimize the simplex-net with the K-shot supernets fixed. Then we randomly sample m paths and optimize the simplex-net accordingly. Besides, to ensure the memory cost, we benchmark the same batch size for the images and divide the image batch into m groups, with b/m images for each path. Then the optimization of simplex-net in Eq.(7) for some iteration τ evolves into

$$\begin{aligned} \boldsymbol{\sigma}^{(\tau+1)}_{\mathcal{A}} &= \arg\min_{\boldsymbol{\sigma}} \ \mathcal{L}_{train}(\mathcal{A}, \widetilde{\boldsymbol{\theta}}(\boldsymbol{\Theta}^{(\tau)}, \boldsymbol{\sigma}; \mathcal{A})), \\ \widetilde{\boldsymbol{\theta}}(\boldsymbol{\Theta}^{(\tau)}, \boldsymbol{\sigma}; \mathcal{A}) &= \boldsymbol{\Theta}^{(\tau)} \cdot \pi(\mathcal{A}; \boldsymbol{\sigma}) \end{aligned} \tag{8}$$

As for the training of supernets, we just optimize the weights of supernet with the image mini-batch by fixing the simplex-net. The optimization process is thus:

$$\begin{aligned} \boldsymbol{\Theta}^{(\tau+1)}_{\mathcal{A}} &= \arg\min_{\boldsymbol{\Theta}} \ \mathcal{L}_{train}(\mathcal{A}, \widetilde{\boldsymbol{\theta}}(\boldsymbol{\Theta}, \boldsymbol{\sigma}^{(\tau)}; \mathcal{A})), \\ \widetilde{\boldsymbol{\theta}}(\boldsymbol{\Theta}, \boldsymbol{\sigma}^{(\tau)}; \mathcal{A}) &= \boldsymbol{\Theta} \cdot \pi(\mathcal{A}; \boldsymbol{\sigma}^{(\tau)}) \end{aligned} \tag{9}$$

For simplicity, we use the iterative training method by default till we converge to the optimal solution $\boldsymbol{\Theta}^*$ and $\boldsymbol{\sigma}^*$. And the searching with K-shot supernets is formulated as Eq.(10).

$$\begin{aligned} \boldsymbol{a}^* &= \arg\max_{\boldsymbol{a}\in\mathcal{A}} \ \mathrm{Acc}_{val}(\boldsymbol{a}, \widetilde{\boldsymbol{\theta}}^*), \\ \text{s.t. } \widetilde{\boldsymbol{\theta}}^*(\boldsymbol{\Theta}^*, \boldsymbol{\sigma}^*; \boldsymbol{a}) &= \boldsymbol{\Theta}^* \cdot \boldsymbol{\lambda}_a = \boldsymbol{\Theta}^* \cdot \pi(\boldsymbol{a}; \boldsymbol{\sigma}^*); \end{aligned} \tag{10}$$

Table 1. The training and search cost (GPU hours) of 1 epoch with different K on ImageNet dataset. The 4-th row reports the GPU usage with the batch size 128 for each GPU.

Stage	K = 1	K = 2	K = 4	K = 8	K = 12
Training	2.27	2.28	2.32	2.37	2.43
Search	0.233	0.234	0.237	0.242	0.246
GPU usage	17.5G	17.7G	17.9G	18.4G	18.9G

Analysis on Training Cost. Although we introduced a simplex-net to generate the customized code, it only involves several fully connected layers; the additional computational cost is negligible. Besides, our K-shot supernets have $(K-1)$ times more parameters than the ordinary one-shot weight sharing strategy. Intuitively, we might think the K-shot will increase the GPU memory usage and training time a lot, but with Eq.(5), the K-shot weight is merged before the model forwarding, which means there is no additional feature map being generated during the training process. As shown in Table 1, from the perspective of GPU memory usage and training speed, the cost of K-shot supernets is almost the same as the one-shot method ($K = 1$). However, introducing a large number of parameters will make the model very hard to optimize. We may need more training epochs to ensure the model has converged. Please see the experimental ablation studies in Section 5.3.

Search the optimal path with an Evolutionary algorithm. Since the search space is enormous (*e.g.*, 13^{21} for searching operations in MobileNetV2 search space), to boost the search efficiency, we leverage the multi-objective NSGA-II (Deb et al., 2002) algorithm for evolutionary search, which is easy to integrate hard FLOPs constraint.

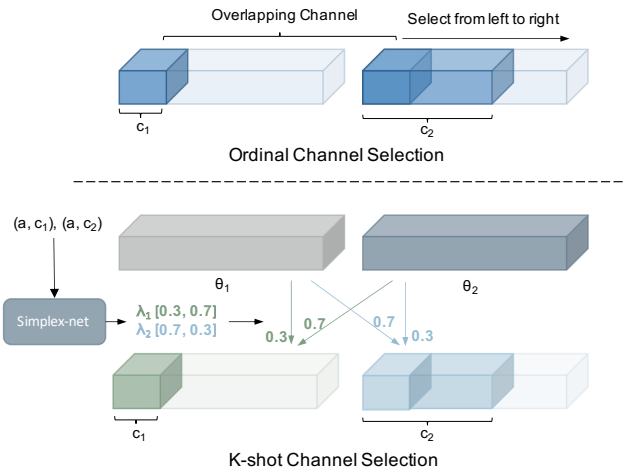

Figure 3. Examples of weight sharing patterns for width. For ordinal weight sharing pattern, leftmost c channels are assigned for the subnets of width c. For the width in K-shot pattern, each width can customize code for constructing its suitable weight. Therefore, each channel amounts to different weights in K-shot pattern.

We set the population size as 50 and the number of generations as 20. We randomly select a group of architectures within the target FLOPs as the first generation. Then in each iteration, the top 20 architectures with the highest accuracy are selected as parents to generate new architectures via mutation and crossover. After the search, we only retrain the architecture with the highest accuracy from scratch and report its performance. More details about evolutionary search are illustrated in supplementary materials.

4. Joint Searching with Channel Dimension

In previous sections, we have leveraged the K-shot supernets and a simplex-net to enable different architectures to have adaptively different weights for some specific operation. Now, we further extend our method to jointly searching with channels for a more fine-grained architecture with decent performance. For clarity, we use $\boldsymbol{a}$ to denote the architecture (operation) and $\boldsymbol{c}$ to denote the width structure over all layers, and $(\boldsymbol{a}, \boldsymbol{c})$ refers to a subnet with architecture $\boldsymbol{a}$ with width $\boldsymbol{c}$.

Basically, if we choose different channel width under the same operations, current methods (Yu & Huang, 2019; Dong & Yang, 2019b; Su et al., 2021b;c) mainly follow the ordinary weight sharing pattern, which only takes channels from one side (*e.g.*, left). Specifically, to allocate c channels at a certain layer, an ordinary weight sharing pattern assigns the leftmost c channels as Figure 3. Such a hard assignment imposes a strong constraint for different channels; that is, though we choose different channel width, the overlapping channels will always share the same set of weights. We expect to decouple the shared weights when we select different channel width by following our proposed idea. To achieve this, we add an additional "channel branch" in simplex-net, which takes a channel encoding vector and merged with the "architecture branch" to generate the final customized code (see Figure 2.2). Now, we can include the "channel branch" to our objective function:

Algorithm 1 Training and search with K-shot supernets

Input: K-shot supernets with weights $\boldsymbol{\Theta}$, Simplex-net π with weights $\boldsymbol{\sigma}$, maximum training epochs T, warmup epochs T_w.

Init $\tau = 0$, $\boldsymbol{\lambda} = [\frac{1}{K}, ..., \frac{1}{K}]$;

while $\tau \leq T$ **do**
  randomly sample an subnets with architecture $\boldsymbol{a}$;
  if $\tau \leq T_w$ **then**
    train K-shot supernets with weights of $\boldsymbol{\Theta\lambda}$;
  else
    generate customized code $\boldsymbol{\lambda} = \pi(\boldsymbol{a}; \sigma)$;
    calculate regularization term r_c with Eq.(16);
    Iteratively train k-shot supernets and simplex-net π with Eq.(9) and Eq.(11);
  end
end

Output: The architecture $\boldsymbol{a}^*$ searched with Eq.(12).

$$\begin{aligned} \boldsymbol{\Theta}^*, \boldsymbol{\sigma}^* &= \arg\min_{\boldsymbol{\Theta}, \boldsymbol{\sigma}} \mathop{\mathbb{E}}_{(\boldsymbol{a}, \boldsymbol{c}) \in (\mathcal{A}, \mathcal{C})} \left[\mathcal{L}_{train}(\boldsymbol{a}, \boldsymbol{c}, \widetilde{\boldsymbol{\theta}}) \right] \\ \text{s.t. } \widetilde{\boldsymbol{\theta}}(\boldsymbol{\Theta}, \boldsymbol{\sigma}; \boldsymbol{a}, \boldsymbol{c}) &= \boldsymbol{\Theta} \cdot \boldsymbol{\lambda}_{\boldsymbol{a}, \boldsymbol{c}} = \boldsymbol{\Theta} \cdot \pi(\boldsymbol{a}, \boldsymbol{c}; \boldsymbol{\sigma}) \end{aligned} \tag{11}$$

With the channel search included in our objective, we can rewrite the Eq.(10) as:

$$\begin{aligned} \boldsymbol{a}^*, \boldsymbol{c}^* &= \arg\max_{(\boldsymbol{a}, \boldsymbol{c}) \in (\mathcal{A}, \mathcal{C})} \text{Acc}_{val}(\boldsymbol{a}, \boldsymbol{c}, \widetilde{\boldsymbol{\theta}}^*), \\ \text{s.t. } \widetilde{\boldsymbol{\theta}}^*(\boldsymbol{\Theta}^*, \boldsymbol{\sigma}^*; \boldsymbol{a}, \boldsymbol{c}) &= \boldsymbol{\Theta}^* \cdot \boldsymbol{\lambda}_{\boldsymbol{a}, \boldsymbol{c}} = \boldsymbol{\Theta}^* \cdot \pi(\boldsymbol{a}, \boldsymbol{c}; \boldsymbol{\sigma}^*) \end{aligned} \tag{12}$$

Similarity of architectures with different channel width. Despite the simplex-net can directly learn the customized code for different channel encoding vectors, we can still add a regularization term to further help it to distinguish different channel width.

Suppose we have two set of channels $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ (under the same architecture), the corresponding supernets can be expressed by $\boldsymbol{\Theta\lambda}_1$ and $\boldsymbol{\Theta\lambda}_2$ (Eq.(11) and Eq.(12)), then we can define the similarity of the two supernets by:

$$S((\boldsymbol{a}, \boldsymbol{c}_1), (\boldsymbol{a}, \boldsymbol{c}_2)) = \boldsymbol{\lambda}_1^T \boldsymbol{\Theta}^T \boldsymbol{\Theta} \boldsymbol{\lambda}_2 = \langle \boldsymbol{\Theta}^T \boldsymbol{\Theta}, \boldsymbol{\lambda}_1^T \boldsymbol{\lambda}_2 \rangle \tag{13}$$

with Cauchy–Schwarz inequality (Alzer, 1999):

$$S((\boldsymbol{a}, \boldsymbol{c}_1), (\boldsymbol{a}, \boldsymbol{c}_2)) \leq \left\| \boldsymbol{\Theta}^T \boldsymbol{\Theta} \right\| \left\| \boldsymbol{\lambda}_1^T \boldsymbol{\lambda}_2 \right\| \tag{14}$$

therefore, the supremacy for the similarity S lies in the weights from supernets and the inner product of $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$.

Since we iteratively update supernets and simplex-net, the first term of $\left\|\mathbf{\Theta}^T\mathbf{\Theta}\right\|$ are fixed during updating simplex-net. As a result, only $\boldsymbol{\lambda}$ affects their inner product.

With ordinary sharing pattern, we can measure the difference of any two sets of channels $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ with ℓ_1 distance $d(\boldsymbol{c}_1, \boldsymbol{c}_2) = \|\boldsymbol{c}_1 - \boldsymbol{c}_2\|_1$. Ideally, if the distance $d(\boldsymbol{c}_1, \boldsymbol{c}_2)$ is smaller, the representation of channels $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ should be closer and vice versa. For this reason, we pre-defined a threshold distance m, if $d(\boldsymbol{c}_1, \boldsymbol{c}_2) < m$, $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ should have similar representation, otherwise, they should be different. With this formulation, we can obtain a conditional distribution by $\boldsymbol{\lambda}$:

$$P(\boldsymbol{c}_i|\boldsymbol{c}_k) = \frac{\exp(\boldsymbol{\lambda}_i^T\boldsymbol{\lambda}_k/\tau)}{\exp(\boldsymbol{\lambda}_i^T\boldsymbol{\lambda}_k/\tau) + \sum_{j=1}^{n} \mathbb{1}_{[j\neq k]}\exp(\boldsymbol{\lambda}_i^T\boldsymbol{\lambda}_j/\tau)} \tag{15}$$

which represents the likelihood that $d(\boldsymbol{c}_i, \boldsymbol{c}_k) < m$, and τ is a temperature parameter that controls this loss's concentration level. With Eq.(14), we proposed a non-parametric regularization term for the simplex-net to distinguish different channel width under the same architecture(operation). The learning objective is then to optimize the conditional probability of similarity for different channels, which can be expressed by the negative log-likelihood as follows:

$$r_{\boldsymbol{c}}(\boldsymbol{\sigma}) = -\log P(\boldsymbol{c}_i|\boldsymbol{c}_k) \tag{16}$$

Hence, the objective function for simplex-net is as follows:

$$L(\boldsymbol{\sigma}) = \mathcal{L}_{train}(\boldsymbol{a}, \boldsymbol{c}, \widetilde{\boldsymbol{\theta}}(\boldsymbol{\sigma}; \mathbf{\Theta}, \boldsymbol{a}, \boldsymbol{c})) + \alpha \cdot r_{\boldsymbol{c}}(\boldsymbol{\sigma}), \tag{17}$$

where α is the trade-off hyperparameter (we find $\alpha = 1$ suffices in our experiments). The overall training procedure for our K-shot supernets and simplex-net is summarized as Algorithm 2.

5. Experimental Results

5.1. Search on ImageNet

Dataset. We perform the architecture search on the large-scale dataset ImageNet (ILSVRC-12) (Russakovsky et al., 2015), which contains 1.28M training images from 1000 categories. Specifically, following (Guo et al., 2020b), we randomly sample 50K images from the training set as the local validation set, with the rest images used for training. Finally, we report the accuracy of our searched architecture on the test dataset (which is the public validation set of the original ILSVRC2012 ImageNet dataset). All experiments are implemented with PyTorch (Paszke et al., 2019) and trained on 8 NVIDIA Tesla V100 GPUs.

Search space. In this paper, we implement the joint search of operations and channels for the fine-grained NAS. We follow the same search space as SPOS (Guo et al., 2020b), which has 21 to-be-searched layers and 3 fixed layers. For channels, we search for all 24 layers. For operation search, we adopt the same macro search space as other one-shot methods for a fair comparison. Concretely, to construct the supernet, we leverage the MobileNetV2 inverted bottleneck (Sandler et al., 2018) with an optional squeeze-and-excitation (SE) (Hu et al., 2018) module. For each search block, the convolutional kernel size is determined within 3,5,7 with the expansion ratio selected in 3,6, and each block can choose to whether use SE module or not. An additional identity block is also attached for a flexible depth search. Moreover, to accommodate the channel search, the width coefficient of each operation is selected within $\{0.2, 0.4, 0.6, 0.8, 1.0\}$ for all 24 layers. As a result, the overall search space amounts to be $5^{24} \times 13^{21}$.

Supernet training. We adopt b_s and τ as 16 and 0.3 for Eq.(9) and Eq.(15), respectively. For training K-shot supernets, we follow the same training recipe as (You et al., 2020; Guo et al., 2020b) for a fair comparison. With a batch size of 1024, the supernets are trained using a SGD optimizer with 0.9 momentum and Nesterov acceleration. The learning rate is initialized as 0.12 and decay with cosine annealing for 120 epochs. For each sampled architecture, we use the same code $\boldsymbol{\lambda}$ (*i.e.*, $1/K$) for the K-shot supernets within the first 5 epochs to warm up all the weights. As for the simplex-net, we adopt a two-layer MLP for both encodings (*i.e.*, Architecture and Channel). We empirically find that more layers of MLP do not bring further significant improvement, which infers that a two-layer MLP may be enough for investigating the encodings. Then, we include our proposed simplex-net to learn the customized code $\boldsymbol{\lambda}$ via an alternate iterative procedure. More details of training and search are illustrated in supplementary materials.

Retraining. For retraining the searched architectures, we follow the same strategy as previous works (Tan et al., 2019; You et al., 2020). Specifically, we adopt a RMSProp optimizer with 0.9 momentum and 1024 batch size. The learning rate is increased from 0 to 0.128 linearly for 5 epochs and then decays 0.03 every 2.4 epochs. Besides, the exponential moving average is also adopted with a decay rate of 0.9999.

Performance of searched structures. We perform the search with 4 different groups of FLOPs constraints; each group is composed of a pair of experiments with/without channel width search. As shown in Table 2, with the same search space, our searched 343M K-shot-B achieves 77.2% on Top-1 accuracy, which surpasses other methods by more than 0.4%. Moreover, we conduct the joint search of operations and channels; the accuracy is further boosted by 0.2%$\sim$0.3%, which validates our method's effectiveness.

5.2. Search on NAS-Bench-201

The key challenge of the NAS algorithm lies in the evaluation and ranking reliability of the supernet, which can

Table 2. Comparison of searched architectures w.r.t. different state-of-the-art NAS methods. Search number means the number of evaluated architectures during searching. ‡: TPU, ⋆: joint search of operations and channel width.

Methods	Top-1 (%)	Top-5 (%)	FLOPs (M)	Params (M)	Memory cost	training cost (GPU days)	search number	search cost (GPU days)
ShuffleNetV2-1.0 (Ma et al., 2018)	69.4	-	146	-	-	-	-	-
MbnV2-0.5 (Sandler et al., 2018)	63.3	-	150	1.3	-	-	-	-
FBNetV2-F3* (Wan et al., 2020)	73.2	-	126	-	single path	12	-	-
K-shot-NAS-D	**73.4**	90.7	133	2.6	single path	12	1000	< 1
K-shot-NAS-D*	**73.7**	90.9	145	2.8	single path	12	1000	< 1
SCARLET-C (Chu et al., 2019a)	75.6	92.6	280	6.0	single path	10	8400	12
MbnV2-1.0 (Sandler et al., 2018)	72.0	91.0	300	3.4	-	-	-	-
MnasNet-A1 (Tan et al., 2019)	75.2	92.5	312	3.9	single path + RL	288‡	8000	-
K-shot-NAS-C	**76.3**	92.6	281	4.4	single path	12	1000	< 1
K-shot-NAS-C*	**76.5**	92.7	286	4.7	single path	12	1000	< 1
Proxyless-R (Cai et al., 2018)	74.6	92.2	320	4.0	two paths	15	1000	-
AngleNet (Hu et al., 2020)	74.2	-	325	-	-	10	-	-
SPOS (Guo et al., 2020b)	76.2	-	328	-	single path	12	1000	< 1
MnasNet-A2 (Tan et al., 2019)	75.6	92.7	340	4.8	single path + RL	288‡	8000	-
ST-NAS-A (Guo et al., 2020a)	76.4	93.1	326	5.2	single path	-	990	-
SCARLET-B (Chu et al., 2019a)	76.3	93.0	329	6.5	single path	10	8400	12
GreedyNAS-B (You et al., 2020)	76.8	93.0	324	5.2	single path	7	1000	< 1
MCT-NAS-B (Su et al., 2021a)	76.9	93.4	327	6.3	single path	12	100	< 1
FairNAS-C (Chu et al., 2019b)	76.7	93.3	325	5.6	single path	-	-	
BetaNet-A (Fang et al., 2019)	75.9	92.8	333	4.1	single path	7	-	-
K-shot-NAS-B	**77.2**	93.3	332	6.2	single path	12	1000	< 1
K-shot-NAS-B*	**77.4**	93.5	343	6.4	single path	12	1000	< 1
AngleNet (Hu et al., 2020)	76.1	-	470	-	-	10	-	-
MnasNet-A3 (Tan et al., 2019)	76.7	93.3	403	5.2	single path + RL	288‡	8000	-
EfficientNet-B0 (Tan & Le, 2019)	76.3	93.2	390	5.3	single path	-	-	-
K-shot-NAS-A	**77.6**	93.6	422	6.5	single path	12	1000	< 1
K-shot-NAS-A*	**77.9**	93.8	412	7.8	single path	12	1000	< 1

be reflected by the ranking correlation between the evaluation performances of all architectures on supernets and their actual performances. Hence, for a better evaluation of our methods, we tend to use the NAS-Bench-201 (Dong & Yang, 2020) to analyze our supernets and architecture search. NAS-Bench-201 is a NAS benchmark that contains 15625 architectures and provides the train-from-scratch performances of these architectures evaluated on ImageNet-16-120, CIFAR-100, and CIFAR-10.

Searching Results of NAS-Bench-201 and comparison of ranking ability. In K-shot supernets, each architecture can acquire its customized code $\boldsymbol{\lambda}$ to migrate the gap between stand-alone training and joint training on supernets. As a result, our method may lead to a more accurate ranking on different architectures and better performance. As Table 3 shows, our searched results achieve much higher performance on NAS-Bench-201 set, *e.g.* 0.7% higher than AngleNet with ImageNet dataset. Moreover, we also calculate the Kendall's Tau correlation coefficients between the validation accuracy on supernet and their ground-truth performances. As in Table 4, we run each method 10 times and report its average Kendall's Tau value. The results show that our method achieves a much higher Kendall's Tau with the proposed K-shot supernets, *e.g.*, 7.64% higher than the Single-path method with the CIFAR-10 dataset, which indicates that our K-shot supernets estimates the validation accuracy better, and thus searches for the good architectures more accurately.

Table 3. Searching results (mean±std) on NAS-Bench-201 dataset.

Method	CIFAR-10	CIFAR-100	ImageNet-16
GDAS (Dong & Yang, 2019a)	93.52 ± 0.15	67.52 ± 0.15	40.91 ± 0.12
DARTS- (Chu et al., 2021)	93.80 ± 0.40	71.36 ± 1.51	45.12 ± 0.82
SPOS (Guo et al., 2020b)	93.67 ± 0.26	69.83 ± 0.21	44.71 ± 0.17
AngleNet (Hu et al., 2020)	94.01 ± 0.37	72.96 ± 0.26	45.83 ± 0.19
K-shot NAS	94.19 ± 0.16	73.45 ± 0.05	46.53 ± 0.11
optimal	94.37	73.51	47.31

Table 4. The comparison of Kendall's Tau w.r.t different methods on NAS-Bench-201.

Method	CIFAR-10	CIFAR-100	ImageNet-16
SPOS (Guo et al., 2020b)	55.00%	56.00%	54.00%
AngleNet (Hu et al., 2020)	57.48%	60.40%	54.45%
K-shot NAS	62.64%	61.22%	56.33%

Comparison between number of K for supernets. Since the proposed supernets impose a rank-K approximation for architectures within search space, the number of K is a critical hyperparameter to discuss. Figure 4 shows the searching result for different number K. As we can see clearly, the

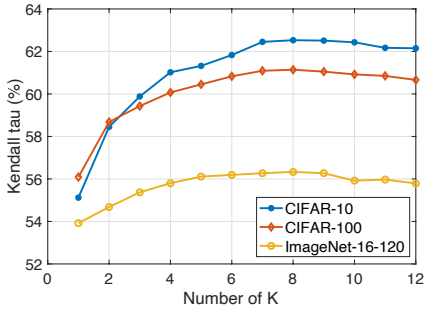

Figure 4. Top-1 accuracy of searched models on ImageNet dataset by different methods with the increasing of search numbers.

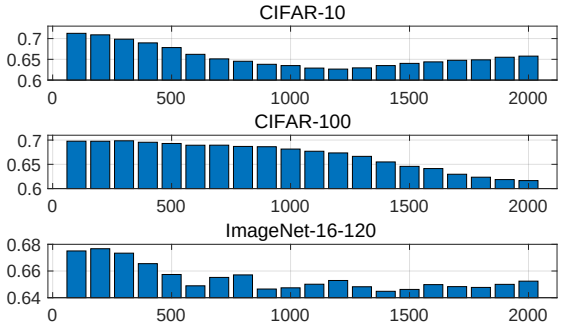

Figure 5. The Kendall's Tau coefficient of K-shot supernets for high-performance architectures. The abscissa value indicates the number of architectures in high-performance group $\boldsymbol{a}_{high}$.

Kendall's Tau coefficient of K-shot supernets benefits from the increase of K (especially when K is small), demonstrating our proposed K-shot supernets can effectively bridge the gap between supernets and true performance. Such observation further verified our hypothesis that the number of K determines the approximation level between supernets and true performance, and thus larger K induces a more accurate ranking for sub-networks. As a result, the ranking ability of supernets will be closer to the Oracle optimal with a larger number of K, *e.g.* $K = 8$. Moreover, the performance tends to be optimal when K selected from $[6 - 10]$ but decreases afterward; this is because the larger K induces more weights for K-shot supernets, and thus K-shot supernets need longer training time. See more experiments about training K-shot supernets in supplementary materials.

The ranking ability for the high-performance architectures. We have used the Kendall's Tau to show the powerful ranking ability of our K-shot supernet. However, a high Kendall's Tau score cannot guarantee that the best architecture has the highest ranking. In NAS, we care more about the optimal architectures with the given computational budgets. Intuitively, a natural question comes to *whether the proposed K-shot supernets enable to rank high-performance architectures more accurately?* With this aim, we divide the architectures into the high-performance group and low-performance group based on their stand-alone accuracy. Now, we can reformulate Kendall's Tau score as follows: "For any pair ($\boldsymbol{a}_{high}$, $\boldsymbol{a}_{low}$), it is said to be concordant if $\boldsymbol{a}_{high}$ better than $\boldsymbol{a}_{low}$, otherwise, it will be called disconcordant". The new Kendall's Tau for evaluating the high-performance architecture can be calculated by $(concordant - disconcordant)/allpairs$. As shown in Figure 5, we show Kendall's Tau with different dividing ranks. In detail, with the proposed K-shot supernets, the top performance architectures can be more accurately ranked, *e.g.*, the top 100 architectures can be ranked with Kendall's Tau coefficient of 0.68 while 0.56 for the overall architectures. More details about Kendall's Tau and its experiments are formulated in the supplementary materials.

Table 5. The performance gain of each part in K-shot supernets with 420M FLOPs on MobileNet search space.

#	K-shot	One-shot	simplex-net	Channel	r_c	Top-1	Top-5
1		✓				77.0%	93.4%
2		✓		✓		77.1%	93.4%
3	✓					77.2%	93.5%
4	✓		✓			77.5%	93.6%
5	✓		✓	✓		77.6%	93.6%
6	✓		✓	✓	✓	77.9%	93.8%

5.3. Ablation Studies

Effect of each proposed techniques in K-shot supernets. As shown in Table 5, we conduct the ablations about the proposed K-shot supernets. We evaluate the performance with different combinations of these strategies on ImageNet and report their Top-1 accuracy in Table 5. With 420M FLOPs budget, we re-implement Single-path (Guo et al., 2020b), and it achieves 77.02% top-1 accuracy on ImageNet dataset in Table 5(#1). Besides, if we joint search operations and widths with a one-shot strategy, it achieves 77.14% top-1 accuracy(#2). When searching with K-shot supernets but with the coefficient vector fixed as $1/K$ (#3), the search result only improves by 0.12%. However, by comparing #3 and #4, if we use simplex-net to generate the customized code for K-shot supernets, we observe that the simplex-net and K-shot supernets are indeed helpful for the search result. Moreover, by comparing #5 and #6, if we promote our method to the channel width search and incorporate the non-parametric regularization, we can further boot our search result with 0.28%.

Visualization of customized code $\boldsymbol{\lambda}$. In our method, we aim to encourage different architectures to have specialized weights, which are assumed to be represented with a convex combination of K-shot supernets via the customized code $\boldsymbol{\lambda}$. In this way, the code actually reflects the preference

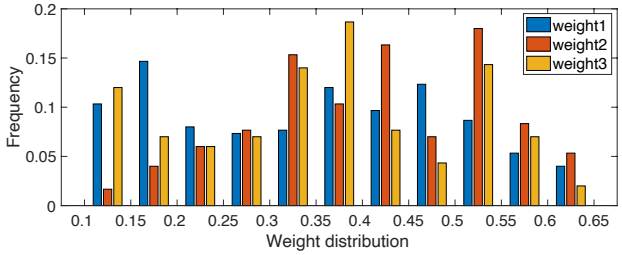

Figure 6. Histogram of weight distribution with $K = 3$ of 1000 sub-networks during evolutionary search w.r.t.420M FLOPs on ImageNet dataset.

Table 6. Comparison of K-shot supernets with baseline methods on ImageNet dataset w.r.t.330M and 420M FLOPs budget.

Method	FLOPs	Accuracy(%)	FLOPs	Accuracy(%)
One-shot-NAS	330M	76.2	420M	77.0
Random K-shot-NAS		76.2		77.1
fixed K-shot-NAS		76.4		77.2
K-shot-NAS		77.2		77.6

over all supernets for each path. To examine this *preference*, we record the customized codes $\boldsymbol{\lambda}$ of all searched 1K architectures (420M FLOPs) during the evolutionary searching on ImageNet dataset with $K = 3$, then we show their histogram in Figure 6. Concretely, the value of these 3 customized codes $\boldsymbol{\lambda}$ distributes evenly within a large range from 0.1 to 0.65, which indicates our method can allocate customized code in a wide range for different architectures. Besides, the frequency of these 3 customized codes $\boldsymbol{\lambda}$ varies accordingly for different architectures, which implies that these 3 customized codes are learned to reflect the different potential of architectures by various preferences. More visualization of customized code with detailed explanations is illustrated in supplementary materials.

Effect of simplex-net. We introduce two baseline methods to investigate the effect of simplex-net, namely, *fixed K-shot supernets*: the customized code $\boldsymbol{\lambda}$ is fixed with $1/K$ for all architectures. *Random K-shot supernets*: we randomly initial parameters $\boldsymbol{\lambda}$ and never change it during training for 10 times, and record the average searched performance. From Table 6, our method achieves much higher performance than baseline methods; while randomly or fixed (*e.g.*, $1/K$) assign $\boldsymbol{\lambda}$ for the introduced K times of supernets nearly do not affect the accuracy performance, which indicates that our simplex-net can effectively boost the performance by providing customized code $\boldsymbol{\lambda}$ for each architecture.

Tradeoff between Training Epochs and Number of K. In K-shot NAS, we leverage K supernets and only introduce negligible training cost and GPU memory as shown in Table 1. However, with (K-1) times more parameters than the ordinary one-shot NAS, the K-shot supernets may become hard to be optimized. To investigate the trade-off between training epochs and the number of K, we implement the search with different K and supernet training epochs on ImageNet dataset. From Figure 7 we can see that, on the one hand, with a larger K, supernets need to be trained with more epochs for a more accurate ranking of subnets. On the other hand, with a tight training budget, smaller K may induce better training results. As a result, for the trade-off between training cost and the number of K, we simply use $K = 8$ in our method.

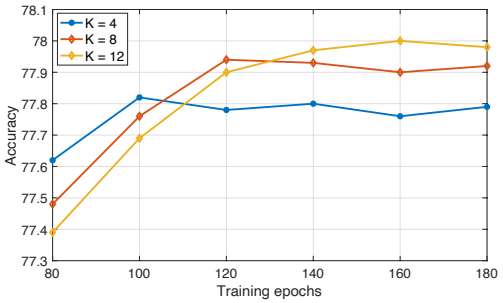

Figure 7. Top-1 accuracy of searched models on ImageNet dataset w.r.t.different K and supernet training epochs.

6. Conclusion

In this work, we discussed the drawbacks of the one-shot NAS and proposed a novel K-shot weight sharing strategy for K-shot supernets. Besides, we introduced a simplex-net which learns context information from architectures/channel encoding vectors, and generate the customized code that cooperate with the K-shot supernets to approximate the optimal stand-alone weights. Furthermore, we designed a regularization term that helps the simplex-net generate different code for different channel width. Extensive experiments have been implemented on the large-scaled ImageNet dataset and NAS-Bench-201 to show the superiority of our propose method to other state-of-the-art NAS methods.

Acknowledgments

This work is funded by the National Key Research and Development Program of China (No. 2018AAA0100701) and the NSFC 61876095. Chang Xu was supported in part by the Australian Research Council under Projects DE180101438 and DP210101859. Shan You is supported by Beijing Postdoctoral Research Foundation.

References

Alzer, H. On the cauchy-schwarz inequality. *Journal of Mathematical Analysis and Applications*, 234(1):6 – 14, 1999.

Cai, H., Zhu, L., and Han, S. Proxylessnas: Direct neural

architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

Chu, X., Zhang, B., Li, J., Li, Q., and Xu, R. Scarletnas: Bridging the gap between scalability and fairness in neural architecture search. *arXiv preprint arXiv:1908.06022*, 2019a.

Chu, X., Zhang, B., Xu, R., and Li, J. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845*, 2019b.

Chu, X., Wang, X., Zhang, B., Lu, S., Wei, X., and Yan, J. {DARTS}-: Robustly stepping out of performance collapse without indicators. In *International Conference on Learning Representations*, 2021.

Clarkson, K. L. and Woodruff, D. P. Low-rank approximation and regression in input sparsity time. *Journal of the ACM (JACM)*, 63(6):1–45, 2017.

De Silva, V. and Lim, L.-H. Tensor rank and the ill-posedness of the best low-rank approximation problem. *SIAM Journal on Matrix Analysis and Applications*, 30 (3):1084–1127, 2008.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2): 182–197, 2002.

Dong, X. and Yang, Y. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1761–1770, 2019a.

Dong, X. and Yang, Y. Network pruning via transformable architecture search. In *Advances in Neural Information Processing Systems*, pp. 759–770, 2019b.

Dong, X. and Yang, Y. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*, 2020.

Du, S., You, S., Li, X., Wu, J., Wang, F., Qian, C., and Zhang, C. Agree to disagree: Adaptive ensemble knowledge distillation in gradient space. *Advances in Neural Information Processing Systems*, 33, 2020.

Fang, M., Wang, Q., and Zhong, Z. Betanas: Balanced training and selective drop for neural architecture search. *arXiv preprint arXiv:1912.11191*, 2019.

Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M. W., and Keutzer, K. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

Guo, R., Lin, C., Li, C., Tian, K., Sun, M., Sheng, L., and Yan, J. Powering one-shot topological nas with stabilized share-parameter proxy. *arXiv preprint arXiv:2005.10511*, 2020a.

Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., and Sun, J. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*, pp. 544–560. Springer, 2020b.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Hu, J., Shen, L., and Sun, G. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

Hu, Y., Liang, Y., Guo, Z., Wan, R., Zhang, X., Wei, Y., Gu, Q., and Sun, J. Angle-based search space shrinking for neural architecture search. In *European Conference on Computer Vision*, pp. 119–134. Springer, 2020.

Huang, T., You, S., Yang, Y., Tu, Z., Wang, F., Qian, C., and Zhang, C. Explicitly learning topology for differentiable neural architecture search. *arXiv preprint arXiv:2011.09300*, 2020.

Kishore Kumar, N. and Schneider, J. Literature survey on low rank approximation of matrices. *Linear and Multilinear Algebra*, 65(11):2212–2244, 2017.

Liu, H., Simonyan, K., and Yang, Y. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

Ma, N., Zhang, X., Zheng, H.-T., and Sun, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 116–131, 2018.

Nahler and Gerhard. Pearson correlation coefficient. 10.1007/978-3-211-89836-9(Chapter 1025):132–132, 2009.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E.,

and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. 2019.

Pirie, W. Spearman rank correlation coefficient. *Encyclopedia of statistical sciences*, 12, 2004.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

Su, X., Huang, T., Li, Y., You, S., Wang, F., Qian, C., Zhang, C., and Xu, C. Prioritized architecture sampling with monto-carlo tree search. *arXiv preprint arXiv:2103.11922*, 2021a.

Su, X., You, S., Huang, T., Wang, F., Qian, C., Zhang, C., and Xu, C. Locally free weight sharing for network width search. *arXiv preprint arXiv:2102.05258*, 2021b.

Su, X., You, S., Wang, F., Qian, C., Zhang, C., and Xu, C. Bcnet: Searching for network width with bilaterally coupled network. *arXiv preprint arXiv:2105.10533*, 2021c.

Tan, M. and Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

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

Tropp, J. A., Yurtsever, A., Udell, M., and Cevher, V. Practical sketching algorithms for low-rank matrix approximation. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1454–1485, 2017.

Wan, A., Dai, X., Zhang, P., He, Z., Tian, Y., Xie, S., Wu, B., Yu, M., Xu, T., Chen, K., et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12965–12974, 2020.

Xu, Y., Xie, L., Zhang, X., Chen, X., Qi, G.-J., Tian, Q., and Xiong, H. Pc-darts: Partial channel connections for memory-efficient architecture search. *arXiv preprint arXiv:1907.05737*, 2019.

Yang, Y., Li, H., You, S., Wang, F., Qian, C., and Lin, Z. Ista-nas: Efficient and consistent neural architecture search by sparse coding. *Advances in Neural Information Processing Systems*, 33, 2020.

Yang, Y., You, S., Li, H., Wang, F., Qian, C., and Lin, Z. Towards improving the consistency, efficiency, and flexibility of differentiable neural architecture search. *arXiv preprint arXiv:2101.11342*, 2021.

You, S., Xu, C., Xu, C., and Tao, D. Learning from multiple teacher networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1285–1294, 2017.

You, S., Huang, T., Yang, M., Wang, F., Qian, C., and Zhang, C. Greedynas: Towards fast one-shot nas with greedy supernet. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1999–2008, 2020.

Yu, J. and Huang, T. Autoslim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 8, 2019.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.