The supplementary materials are organized as follows. In Appendix A, we illustrate the details of the training recipe. In Appendix B, we provide the implementation details of the evolutionary search. In Appendix C, we illustrate the definition of rank correlation coefficients and report the corresponding experimental results. We compare our K-shot NAS with an ensemble of K supernets in Appendix D. We report more visualizations of customized code $\boldsymbol{\lambda}$ in Appendix E. In Appendix F, we provide a detailed algorithm flow of iterative training with K-shot NAS. In section G, we report the ablations of iterative training of K-shot NAS. Finally, we show the visualization of searched architectures in Table 2.

A. Details of Training Recipe

In this section, we present the supernet training and train from scratch details with the proposed K-shot NAS w.r.t. ImageNet and NAS-Bench-201 datasets. In general, we adopt $m = 16$, $a = 1$, $\tau = 0.3$ for all experiments. Besides, we leverage an iterative training strategy for supernets and simplex-net, and details are illustrated in Algorithm 2.

Supernet training with K-shot NAS. For ImageNet dataset, we follow the same strategy as (You et al., 2020; Guo et al., 2020b). In detail, we adopt a batch size of 1024; supernets are trained via a SGD optimizer with 0.9 momentum and Nesterove acceleration. The initial learning rate is set to 0.12 with a cosine annealing strategy, which decays 120 epochs. For each sampled architecture, we use the same code $\boldsymbol{\lambda} = 1/K$ for the K-shot supernets within the first 5 epochs to warm up all supernets. Then we include the proposed simplex-net to learn the customized code $\boldsymbol{\lambda}$ through an alternate iterative procedure, which is presented in Algorithm 2. Especially, when training simplex-net while fix supernets, we divide the image batch into m groups, with each group shares the same architectures. Therefore, the input batch size for simplex-net will be as $m > 1$, which promotes a better optimization for simplex-net with Eq.(9). Besides, for NAS-Bench-201 dataset, we simply follow the same training strategy provided in (Dong & Yang, 2020) for supernet training.

Retraining of searched architectures. To train the searched architectures from scratch, we use the same retraining strategy as (Guo et al., 2020b) for a fair comparison. In detail, we train the searched architecture from scratch with RMSProp optimizer and 0.9 momentum; 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 adopted with a decay rate set to 0.9999.

Training of K-shot NAS. With Eq.(4), we merge the weights of all supernets θ_k to $\widetilde{\boldsymbol{\theta}} = \sum_{k=1}^{K} \lambda_k \boldsymbol{\theta}_k$, then the merged weights $\widetilde{\boldsymbol{\theta}}$ are optimized as same as One-shot NAS, as formulated in Eq. (7). Therefore, we keep almost the same budgets as One-shot NAS while training all K supernets simultaneously. The extra gradient calculation cost of θ_k is small, given a scalar value λ_k.

B. Details of Evolutionary Search

Since the search space is enormous (*e.g.*, $5^{24} \cdot 13^{21}$ for joint searching operations and channel width with 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. We set the population size as 50 and the number of generations as 20, which amounts to 1000 paths in our method. To perform the search, we randomly select a group of architectures within the target FLOPs. In each iteration, the top 20 architectures with the highest accuracy are selected as parents to generate new architectures via mutation and crossover. We evaluate each architecture by the inference with the weights from K-shot supernets with Eq.(12) and record its accuracy. In each generation, 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.

Note that batch normalization (BN) layers are incorporated in most operations (*e.g.*, 3 by 3 inverted residual in MobileNetV2 search space). However, due to the varying channel width, the mean and variance in BN layers are unsuitable for all width. Therefore, we simply use the mean and variance in batches instead, and we set the batch size to 2048 to induce an accurate estimation of mean and variance.

C. Details of Rank Correlation Coefficient

C.1. Defination of kendall's Tau

In section 5.2, we implement the search on NAS-Bench-201 and examine the result with Kendall's Tau coefficient. We rank the evaluation results on various validation sets with 15625 architectures and 50,000 samples. Indeed, the Kendall

Tau correlation is used to evaluate the K-shot supernet's ranking ability with the pairwise ranking performance. With any pair $(\boldsymbol{r}_i, \boldsymbol{r}_j)$ and $(\boldsymbol{s}_i, \boldsymbol{s}_j)$, if we have either both $\boldsymbol{r}_i > \boldsymbol{r}_j$ and $\boldsymbol{s}_i > \boldsymbol{s}_j$, or both $\boldsymbol{r}_i < \boldsymbol{r}_j$ and $\boldsymbol{s}_i < \boldsymbol{s}_j$, these two pairs are considered as **concordant**. Otherwise, it is said to be **disconcordant**. Therefore, the Kendall Tau can be formally defined as

$$K_\tau = \frac{n_{\text{con}} - n_{\text{discon}}}{n_{\text{all}}}, \tag{18}$$

where n_{con} and n_{discon} indicates the number of concordant and disconcordant pairs, and $n_{\text{all}} = \mathbb{C}_n^2$ is the total number of pairs.

Kendall's Tau for "The ranking ability for the high-performance architectures" in Section 5.2. To calculate the Kendall's Tau between high-performance architectures and others. We need to divide the architectures into a high-performance group and a low-performance group with the provided true ranks. As a result, for any pair (a_{high}, a_{low}) during calculation of Eq.(18), a_{high} and a_{low} are from the high-performance group and the low-performance group, respectively.

C.2. Other rank correlation coefficients

To evaluate the search results of K-shot NAS, we also introduce two more correlation coefficients, *i.e.*, Spearman rho (Pirie, 2004), and Pearson (Nahler & Gerhard, 2009). Spearman rho correlation coefficient is the Pearson correlation coefficient between random variable $\boldsymbol{r}$ and $\boldsymbol{s}$, *i.e.*

$$\rho_S = \frac{\text{cov}(r, s)}{\sigma_r \sigma_s}, \tag{19}$$

where $\text{cov}(\cdot, \cdot)$ is the covariance of two variables, and σ_r and σ_s are the standard deviations of r and s, respectively. Since the ranks are integers in our experiments, the Eq.(19) can be fulfilled more efficiently with:

$$\rho_S = 1 - \frac{6 \sum_{i=1}^{n} (\boldsymbol{r}_i - \boldsymbol{s}_i)^2}{n(n^2 - 1)}, \tag{20}$$

Where $n = 15625$ indicates the number of overlapped elements between r and s.

More experiments of ranking correlations of K-shot supernets on NAS-Bench-201. As shown in Table 7, we report the experiments with more ranking correlations of K-shot supernets for a detailed analysis of our method.

Table 7. The ablations of K-shot NAS w.r.t. more ranking correlations.

Method	CIFAR-10	CIFAR-100	ImageNet-16-120
Kendall's Tau	0.63	0.61	0.56
Spearman rho	0.81	0.80	0.73
Pearson	0.92	0.84	0.76

D. The Comparison between K-shot Supernets and Ensemble of K Supernets

Since our K-shot NAS proposes to learn the customized code for each subnet with K supernets. In this way, one natural question comes to *How does K-shot NAS compare to directly ensemble K supernets,* i.e.*, independently train K times of supernets and ensemble their results?* With this aim, with K supernets and ImageNet dataset, we compare the proposed K-shot NAS with two baseline methods, *i.e.*, Ensemble$_{max}$: we output the result from the supernet with maximum Top-1 probability. Ensemble$_{avg}$: we average the output result from K different supernets. We compare our methods with these two baselines in terms of search efficiency and accuracy performance, as shown in Table 8.[2] In detail, accuracy performance benefit from ensemble of K supernets, but it also puts a huge burden on the training cost and GPU usage. Nevertheless, our proposed K-shot NAS can efficiently boost the search results while introducing a negligible additional computation budget.

[2]Since the ensemble of K cost much more computation resolutions than our method, we only implement the search with $K = 2$ for ensemble method.

Table 8. The comparison of K-shot supernets and ensemble of K supernets. The 3-th column indicates the training cost (GPU hours) of 1 epoch. The 4-th column reports the GPU usage with the batch size 64 for each GPU.

Number of K	method	Training cost (h)	GPU usage (G)	Top-1 (%)
K = 1	one-shot NAS	2.27	8.7	77.12
K = 2	Ensemble$_{max}$	4.54	17.5	77.58
K = 2	Ensemble$_{avg}$	4.55	17.6	77.64
K = 2	K-shot NAS	2.28	8.8	77.56
K = 4	K-shot NAS	2.32	8.9	77.79
K = 8	K-shot NAS	2.37	9.2	77.92

E. More Experiments of Visualization of Customized Code $\boldsymbol{\lambda}$

For intuitively understanding, we visualize the proposed customized code $\boldsymbol{\lambda}$ with 1000 paths searched with Imagenet dataset with 420M FLOPs budget w.r.t. different number of K. From Figure 8, we show the customized code with $K = 2, 3, 4$. Concretely, the customized code distributes evenly within a large range w.r.t. different K, which indicates our method can customize the code for different architectures and promotes to distinguish their performance accurately.

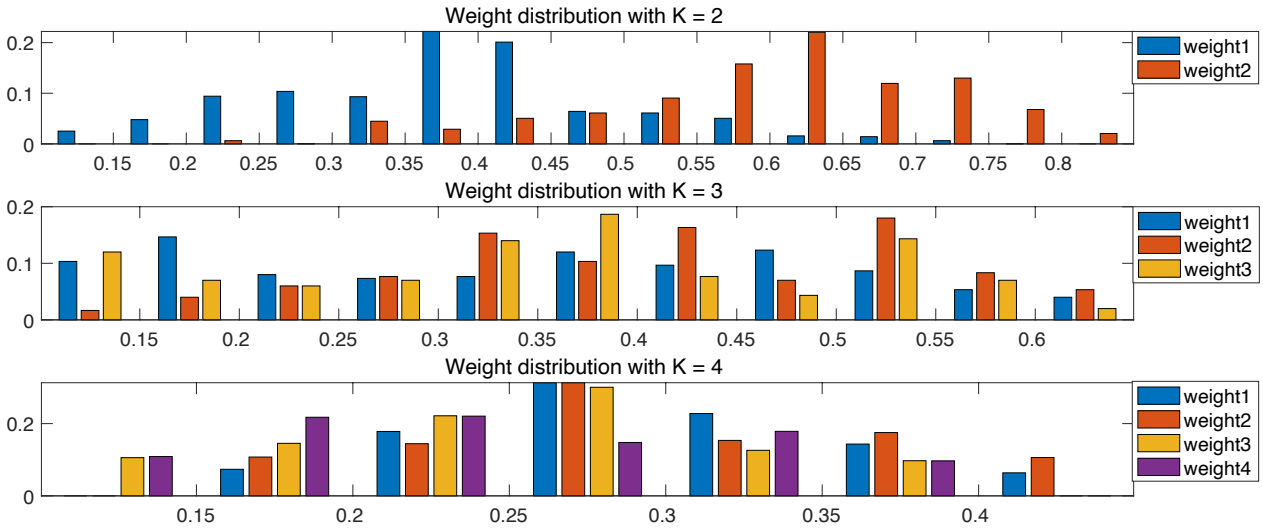


Figure 8. Histogram of weight distribution with different K of 1000 sub-networks during evolutionary search w.r.t. 420M FLOPs on ImageNet dataset.

F. Details of Algorithms Flow with the Iterative Training of K-shot NAS

With K-shot NAS, we propose an iterative training method for K-shot supernets and simplex-net. For intuitively understanding, we provide a detailed algorithm flow about this process as shown in Algorithm 2.

Algorithm 2 Iteratively training of K-shot supernets and simplex-net

Input: K-shot supernets with weights $\mathbf{\Theta}$, simplex-net π with weights $\boldsymbol{\sigma}$, maximum training epochs T.
Init current batch τ = 0;
while $\tau \leq T$ **do**
  if $\tau \,\%\, 2 == 0$ **then**
    randomly sample an subnets with architecture $\boldsymbol{a}$;
    make one-hot architecture parameters of $\boldsymbol{a}$;
    calculate the customized code $\boldsymbol{\lambda}$ with architecture parameters;
    train K-shot supernets with weights of $\mathbf{\Theta}\boldsymbol{\lambda}$;
  else
    randomly sample m groups of architectures $\boldsymbol{a}$;
    make a batch (m) of one-hot architecture parameters with $\boldsymbol{a}$;
    calculate the customized code $\boldsymbol{\lambda}$ with architecture parameters;
    calculate regularization term r_c with Eq.(16);
    forward and backward for updating weights $\boldsymbol{\sigma}$ of simplex-net π;
  end
end
Output: The architecture $\boldsymbol{a}^*$ searched with Eq.(12).

G. Ablations of Iterative Training of K-shot NAS

Since the batch size of simplex-net is inversely proportional to the batch size of a single architecture for K-shot supernets, we adopt an iterative training strategy to balance the training of K-shot supernets and simplex-net in our method. To explore the effectiveness of K-shot NAS, we also implement the search by jointly optimizing K-shot supernets (*e.g.*, $K = 8$) and simplex-net w.r.t. different group numbers m as formulated in Eq.(9).

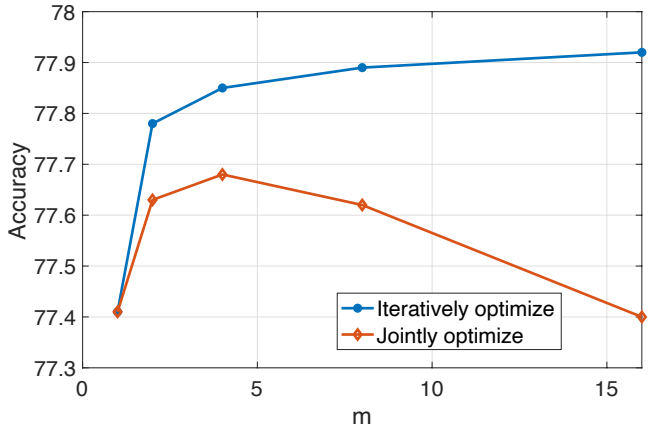


Figure 9. Top-1 accuracy of searched models on ImageNet dataset by different methods with the increasing of group numbers m.

From Figure 9, our algorithm achieves superior performance with an iteratively training strategy. Concretely, the accuracy performance benefits from the increase of m when m selected from $[1, 2, 4]$, which is because the increase of batch size m for simplex-net promotes the optimization of customized code $\boldsymbol{\lambda}$. However, after m goes beyond 4, the searched results of the jointly optimizing method drop gradually, which is because a larger m is not suitable for optimizing K-shot supernets. For example, by jointly optimizing of K-shot supernets and simplex-net, with the 1024 batch size and m is set to 16, the weights of supernets are optimized with batch size $\frac{1024}{16} = 64$ for each architecture, which is far from enough to optimize supernets.

H. Ablation study of K for Figure 4

The statistical results of K in Figure 4 are reported in Table 9. With K increasing from 1 to 12, the Kendall tau is increased by 2%$\sim$7% for 3 datasets, which is a significant improvement since K-shot NAS can achieve better performance due to the more accurate approximation between supernets and true performance.

Table 9. Statistical results of Kendall tau w.r.t. K in Figure 4.

K	1	2	4	8	12
CIFAR-10	55.02 $\pm$ 0.27	58.23 $\pm$ 0.21	60.96 $\pm$ 0.19	62.64 $\pm$ 0.17	62.13 $\pm$ 0.13
CIFAR-100	56.07 $\pm$ 0.18	58.56 $\pm$ 0.17	60.03 $\pm$ 0.11	60.40 $\pm$ 0.13	60.19 $\pm$ 0.09
ImageNet-16	53.97 $\pm$ 0.20	54.36 $\pm$ 0.17	55.91 $\pm$ 0.14	56.33 $\pm$ 0.12	55.89 $\pm$ 0.13

I. Visualization of Searched Architectures

We visualize the searched architectures (*i.e.*, operations and channel width) with our K-shot NAS as Figure 10 and Figure 11. In detail, the searched optimal architectures share similar characteristics as follows:

1. The optimal architecture generally tends to use more 5$\times$5 or 7$\times$7 convolutions with full channel width at the layers close to the last layer. Significantly, the last layer is always with 7$\times$7 kernel size.
2. The channel width in the layers close to the input and output generally tends to be fully preserved. Besides, we also obverse that the channel width in the layers with stride 2 also has the full width.
3. If the computation budget is insufficient (*i.e.*, 343M and 145M), more ID operations will be used at the layers close to the first layer for searching optimal structures.

Notably, we notice that the 412M searched architecture keeps most of its channels for all layers, which may result from two reasons. First, for operations, we search architectures with different expansion ratios (*i.e.*, 3 or 6), which also determines the width for layers. Second, 412M FLOPs is a relatively large budget, and it thus promotes more channels in each layer to boost the performance.

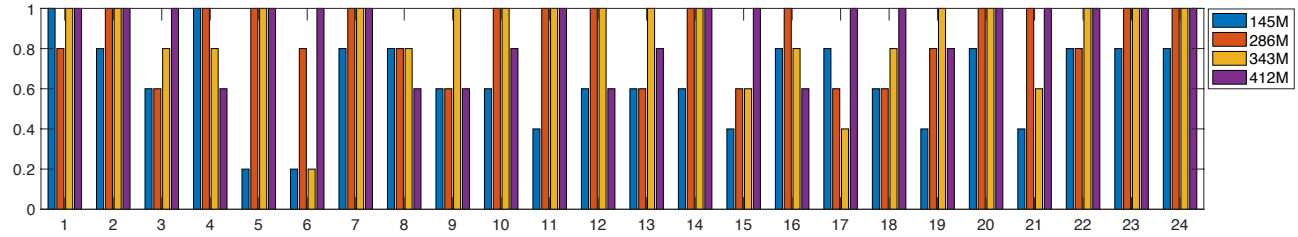
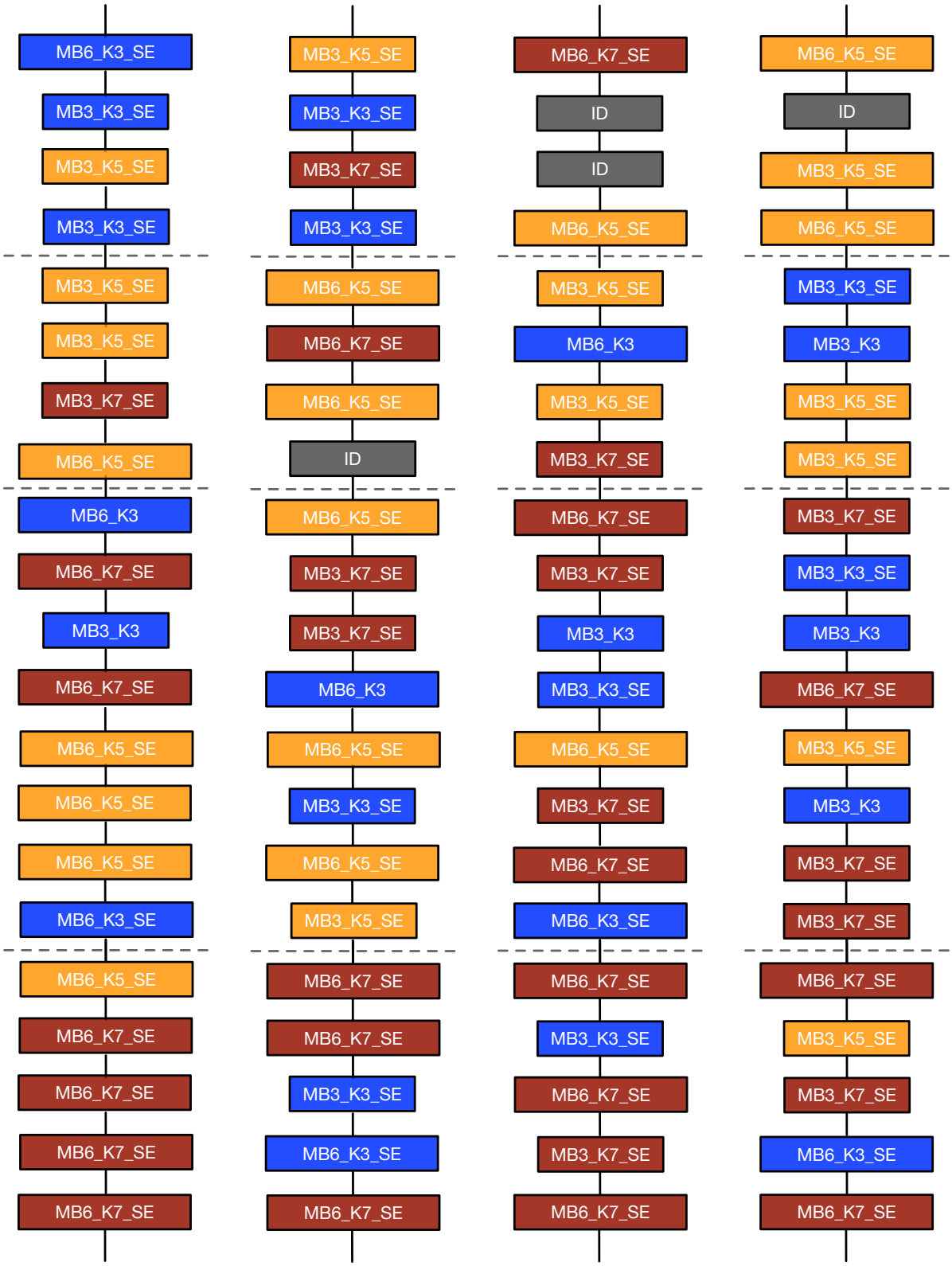


Figure 11. Visualization of channel width searched by K-shot NAS in Table 2.



(a) K-shot-NAS-A* (b) K-shot-NAS-B* (c) K-shot-NAS-C* (d) K-shot-NAS-D*

Figure 10. Visualization of architectures searched by K-shot NAS in Table 2.