Distribution Consistent Neural Architecture Search

Junyi Pan, Chong Sun*, Yizhou Zhou, Ying Zhang, and Chen Li
WeChat, Tencent Inc.
junyipan@tencent.com, waynecsun@tencent.com, harryizzhou@tencent.com, yinggzhang@tencent.com, chaselli@tencent.com

Abstract

Recent progress on neural architecture search (NAS) has demonstrated exciting results on automating deep network architecture designs. In order to overcome the unaffordable complexity of training each candidate architecture from scratch, the state-of-the-art one-shot NAS approaches adopt a weight-sharing strategy to improve training efficiency. Although the computational cost is greatly reduced, such one-shot process introduces a severe weight coupling problem that largely degrades the evaluation accuracy of each candidate. The existing approaches often address the problem by shrinking the search space, model distillation, or few-shot training. Instead, in this paper, we propose a novel distribution consistent one-shot neural architecture search algorithm. We first theoretically investigate how the weight coupling problem affects the network searching performance from a parameter distribution perspective, and then propose a novel supernet training strategy with a Distribution Consistent Constraint that can provide a good measurement for the extent to which two architectures can share weights. Our strategy optimizes the supernet through iteratively inferring network weights and corresponding local sharing states. Such joint optimization of supernet’s weights and topologies can diminish the discrepancy between the weights inherited from the supernet and the ones that are trained with a stand-alone model. As a result, it enables a more accurate model evaluation phase and leads to a better searching performance. We conduct extensive experiments on benchmark datasets with multiple searching spaces. The resulting architecture achieves superior performance over the current state-of-the-art NAS algorithms with comparable search costs, which demonstrates the efficacy of our approach.

1. Introduction

Neural architecture search (NAS) has drawn massive research attention due to its efficacy in automating architecture engineering. A line of NAS algorithms have been successfully applied in the image classification [14, 18, 21, 41, 42] and other related fields (e.g., object detection [4, 35], segmentation [20, 39]). Among many others, early NAS approaches struggle to solve the weight optimization and automated architecture engineering problems in a nested manner. A large number of candidate architectures are sampled and trained from scratch, and the computation cost is thus unaffordable on large datasets.

Recent research hotspot lies in the one-shot NAS algorithms with an additional weight-sharing supernet for architecture performance evaluation. A supernet that contains all candidate architectures is trained only once. Each architecture inherits its weights from the supernet. After that, a search strategy (e.g., reinforcement learning, evolutionary algorithms, etc.) is applied to select the best-performing sub-architecture. The computation cost is thus greatly reduced. Though promising results have been achieved, this global weight-sharing mechanism introduced by the one-shot algorithms leads to severe weight coupling problem [11, 14, 34], and results in at least two limitations. First of all, sub-architectures with shared weights influence each other during the supernet training process. The same operator in different architectures may have different or even
opposite gradient directions. The gradient descent direction for one architecture may be the gradient ascent direction for another one. The global weight sharing will lead to a zigzag optimization process. Second, operators with high FLOPs are less frequently sampled than others under a predefined latency constraint. Architectures with such operators are usually insufficiently trained. Both limitations would cause non-trivial parameter distribution variance between the model trained from scratch and that inherited from the supernet, leading to inaccurate architecture evaluation.

Several recent works are proposed to address such a problem from various perspectives, including search space narrowing [21, 27], knowledge distillation [18, 26, 38], few-shot supernet training [40], to name a few. Though promising, these methods are designed to improve the evaluation performance of the candidate sub-architectures in an empirical way, but the underlying reason, i.e., the distribution gap between the weights inherited from supernet and the weights trained with stand-alone networks, has not been touched and properly addressed.

In this paper, we make the first attempt to theoretically analyze how weight sharing affects such a distribution gap. We manage to prove that the distribution gap is actually determined by the accumulated likelihood probability of each candidate architecture and the joint likelihood of pairwise architectures (i.e., any two architectures with shared weights). However, previous NAS algorithms only take into account the first likelihood probability for network training. Therefore, we propose a new supernet training metric with a distribution consistency constraint that considers both of the two likelihoods. It leads to two iterative sub-processes when training supernet, i.e., optimizing the supernet weights and inferring the local weight sharing states. This enables us to simultaneously supervise the supernet training and shrink the network parameter distribution gap.

However, the above optimizing process is intractable since the computational cost for the joint likelihood of pairwise networks is unaffordable and the solution space of local weight sharing states is too huge. Therefore, we innovatively propose a layer-wise optimization strategy with a clustering mechanism to avoid the computation for joint likelihood of all candidate architecture pairs and restrain the space of weight sharing states. The clustering algorithm is performed on network’s architecture in a self-supervised manner and we use cluster center’s weight sharing state to represent all weight sharing states of architectures in the same cluster. This greatly reduces the computational complexity and makes the whole optimization process feasible.

To summary, our main contributions are three-fold:

- We are the first to solve the weight sharing problem directly from the perspective of diminishing the distribution gap between the weights inherited from the supernet and the weights trained with stand-alone network. Such gap is believed to be the principal reason that impedes the one-shot NAS progress.
- We propose a novel joint training formula to iteratively update the supernet weights and topology, which facilitates a feasible optimization process.
- Our searched architectures deliver the new state-of-the-art performance on different benchmark datasets and search spaces.

2. Related Works

Neural architecture search has been successfully applied in image classification [21, 26, 27] and language tasks [27, 41]. Generally speaking, the NAS algorithms pre-define the search space for a network and exploit one search strategy, such as reinforcement learning [36], and evolutionary algorithms [1, 25, 29], to generate candidate architectures, which are then evaluated on the validation set. The searching strategy is then updated based on the validation results. The above processes are repeated several times until the condition of convergence is met.

In [41, 42], Zoph et al. first introduce NAS in classification and language modeling tasks, which search the optimal states of convolution layers, e.g., kernel size, stride. The works [41] and [42] have inspiring searching performance at the cost of high computation load, as they need to train large amounts of stand-alone models for architecture evaluation. To address these limitations, one-shot NAS algorithms are proposed, which exploit a supernet with shared weights to encode all subnetwork architectures. In such methods, the supernet can directly inherit weights from the supernet for performance evaluation, and the searching time can be greatly reduced. However, the weight sharing mechanism in the supernet training process will introduce weight coupling among different architectures, which greatly degenerates the searching performance. In addition, the nonuniform sampling of network architectures makes some of the subnetworks insufficiently trained. Large numbers of one-shot NAS algorithms are proposed to address the above-mentioned limitations [14, 18, 19]. One way to address the coupling problem is to narrow the search space [21, 42]. In [21, 27, 42], the network is assumed to consist of two kinds of building cells (i.e., the normal cell and the reduction cell), each of which contains some basic blocks (e.g., convolution, pooling). The algorithm only needs to search for the optimal structure for these two building cells, which are then stacked several times to generate the holistic net. Another idea to shrink the search space is to incorporate the sequential searching strategy [19, 21]. Li et al. [19] divide the search process into several stages, and progressively determine the network architecture from a bottom-up manner. Liu [21] simultaneously search the building cells (like [42]) and the ways that the cells are stacked. The search process starts from a
network with simple cells, and then sequentially searches for architectures with more complex cells. In recent years, a few algorithms introduce the distillation technology in one shot NAS algorithms [18, 26, 38]. Such methods train a high-performance network either offline [18, 38] or on-the-fly [26] to supervise the searching process for network architectures in the larger search space.

In most of the previous single-shot architecture search algorithms, the weight parameters are shared across all the sub-architectures with the same operator. The supernet topology (i.e., the parameter sharing states) is fixed during the model training process. While our method makes the first step towards simultaneously optimizing the supernet parameters and topology structure.

3. Neural Architecture Search with Distribution Consistency Constraint

Previous supernet based one-shot NAS algorithms simultaneously optimize the shared weights of different architectures without considering the inherent relations between them. This global weight sharing mechanism leads to the coupling problem among different sub-architectures, wherein the network weights of different sub-architectures influence each other and are usually insufficiently trained. The evaluation accuracy based on the stand-alone model is not always positively associated with the one-shot based evaluation, which is an open problem in the NAS field. We thus theoretically analyze this problem in the view of model parameter distribution.

3.1. A Probabilistic Explanation for Global Weight Sharing Supernet

Given the training data $D$ and a search space consisting of $N$ sampled candidate architectures $\alpha_1, ..., \alpha_N \in \mathcal{A}$, we use $p(W_{\alpha_i} | \alpha_i, D)$ to denote the parameter distribution of subnetwork $\alpha_i$ trained from scratch, and use $p(W_{\mathcal{A}} | \mathcal{A}, D)$ to denote the model parameter distribution of the supernet. Here, $W_{\alpha_i}$ is the model parameter of sub-architecture $\alpha_i$, and is a subset of $W_{\mathcal{A}}$. We argue that it is crucial to shrink the gap between $p(W_{\alpha_i} | \alpha_i, D)$ and $p(W_{\alpha_i} | \mathcal{A}, D)$ to ensure the monotonic correlation of evaluation accuracies of the one-shot and stand-alone models.

Based on the probability theory, $p(W_{\alpha_i} | \mathcal{A}, D)$ can be represented as

$$p(W_{\alpha_i} | \mathcal{A}, D) = \frac{p(W_{\alpha_i} | \alpha_i, D)p(\mathcal{A} | W_{\alpha_i}, \alpha_i, D)}{p(\mathcal{A} | \alpha_i, D)} \quad (1)$$

We assume that the model architectures $\alpha_1, ..., \alpha_N$ are independent, thus $p(\mathcal{A} | W_{\alpha_i}, \alpha_i, D)$ can be expanded as

$$p(\mathcal{A} | W_{\alpha_i}, \alpha_i, D) = \prod_{j=1}^{N} p(\alpha_j | W_{\alpha_i}, \alpha_i, D)$$

$$= \prod_{j=1}^{N} \frac{p(W_{\alpha_i} | \alpha_i, D)}{p(W_{\alpha_i} | \mathcal{A}, D)} \prod_{j=1}^{N} \frac{p(D | W_{\alpha_i}, \alpha_j)p(\alpha_j | W_{\alpha_i}, \alpha_i)}{p(D | W_{\alpha_i}, \alpha_i)}$$

$$= \frac{p(D | W_{\alpha_i}, \alpha_i)}{p(D | W_{\alpha_i}, \alpha_i)}.$$ (2)

The KL-divergence $D_{KL}(p(W_{\alpha_i} | \alpha_i, D), p(W_{\alpha_i} | \mathcal{A}, D))$ is adopted to measure the discordance between $p(W_{\alpha_i} | \alpha_i, D)$ and $p(W_{\alpha_i} | \mathcal{A}, D)$ as

$$D_{KL}(p(W_{\alpha_i} | \alpha_i, D), p(W_{\alpha_i} | \mathcal{A}, D)) = \int p(W_{\alpha_i} | \alpha_i, D) \log \frac{p(W_{\alpha_i} | \alpha_i, D)}{p(W_{\alpha_i} | \mathcal{A}, D)} dW_{\alpha_i}$$

$$= \int p(W_{\alpha_i} | \alpha_i, D) \log \frac{\prod_{j=1}^{N} p(D | W_{\alpha_i}, \alpha_j)p(\alpha_j | W_{\alpha_i}, \alpha_i)}{\prod_{j=1}^{N} p(D | W_{\alpha_i}, \alpha_i, \alpha_j)p(\alpha_j | W_{\alpha_i}, \alpha_i)} dW_{\alpha_i}$$

$$\propto \int p(W_{\alpha_i} | \alpha_i, D) \log \frac{\prod_{j=1}^{N} p(D | W_{\alpha_i}, \alpha_j)}{\prod_{j=1}^{N} p(D | W_{\alpha_i}, \alpha_i, \alpha_j)} dW_{\alpha_i}$$

$$= \int p(W_{\alpha_i} | \alpha_i, D) \log \frac{\prod_{j=1}^{N} p(D | W_{\alpha_i}, \alpha_j) \sum_{\alpha_i \neq \alpha_j}^{N} \prod_{j=1}^{N} p(D | W_{\alpha_i}, \alpha_j, \alpha_j) p(\alpha_j | W_{\alpha_i}, \alpha_i)}{\prod_{j=1}^{N} p(D | W_{\alpha_i}, \alpha_i, \alpha_j) p(\alpha_j | W_{\alpha_i}, \alpha_i)} dW_{\alpha_i}$$

where the local weight sharing mechanism is introduced and $\alpha_i \in \mathcal{A}$ indicates which parameter set the architecture $\alpha_i$ uses. Architectures $\alpha_i$ and $\alpha_j$ have shared weights if $\alpha_i = \alpha_j$, $p(\mathcal{A} | \alpha_i, D)$ and $p(\alpha_j | W_{\alpha_i}, \alpha_i)$ denote the priors of architecture set $\mathcal{A}$ and $\alpha_j$, which can be regarded as constant values. The probability $p(D | W_{\alpha_i}, \alpha_i, \alpha_j)$ denotes the likelihood probability given model parameter $W_{\alpha_i}$ and architectures $\alpha_i$ and $\alpha_j$, which can be further expanded as

$$p(D | W_{\alpha_i}, \alpha_i, \alpha_j) = \int p(W_{\alpha_j, \alpha_i} | \alpha_i, \alpha_j) p(\alpha_j | W_{\alpha_i}, \alpha_i) dW_{\alpha_j, \alpha_i} \quad (3)$$

where we use $W_{\alpha_j, \alpha_i}$ to denote model parameters of $\alpha_j$ eliminating those shared with $\alpha_i$, and $p(D | W_{\alpha_i}, W_{\alpha_j}, \alpha_i, \alpha_j)$ is the likelihood probability of architectures $\alpha_i$ and $\alpha_j$. From Eq. (3) and (4), it is easy to conclude that a larger probability $p(D | W_{\alpha_i}, W_{\alpha_j}, \alpha_i, \alpha_j)$ helps shrink the parameter distribution gap between the one-shot and stand-alone models.

From the view of network parameter distribution consistency, a feasible way to improve the search performance is to maximize $p(D | W_{\alpha_i}, W_{\alpha_j}, \alpha_i, \alpha_j)$ for any architecture pair $(\alpha_i, \alpha_j)$. When $\alpha_i$ and $\alpha_j$ have no shared parameters, $p(D | W_{\alpha_i}, \alpha_i, \alpha_j)$ can be rewritten as $p(D | W_{\alpha_i}, \alpha_i)$. Considering $p(D | W_{\alpha_i}, \alpha_i) \geq p(D | W_{\alpha_i}, \alpha_i, \alpha_j)$, we conclude
that avoiding parameter sharing is a possible way to improve the parameter distribution consistency.

In the previous one-shot NAS algorithms [14, 18, 19], the supernet is trained via solving the following optimization problem:

$$W_A^* = \arg \max_{W_A} \log p(W_A|A, D)$$
$$= \arg \max_{W_A} \log p(D|W_A, A) + \log p(W_A|A) - \log p(D|A),$$

where $p(W_A|A)$ and $p(D|A)$ can be regarded as constant values. In Eq. (5), the distribution gap between the one-shot and stand-alone models are not considered. Based on the previous derivations, we introduce another loss term to measure the distribution gaps between the one-shot and stand-alone models, and Eq. (5) can be reformulated as

$$W_A^*, o_A^* = \arg \max_{W_A, o_A} \log p(D|W_A, A, o_A)$$
$$+ \delta(o_{i_1}, o_{i_2}) \sum_{i,j=1}^N \log p(D|W_{i_1, i_2}, o_{i_1}, o_{i_2}, o_{i_3}).$$

The optimization process in Eq. (6) can be solved via the alternating direction method, i.e., optimizing the optimal local weights assignment $o_A$ with the fixed supernet weights, and vice versa. As far as we know, our method is the first attempt to jointly optimize the supernet topological structure (i.e., between which architectures the weights can be shared) and the supernet model parameters for one-shot NAS algorithms.

### 3.2. Supernet Training with Local Shared Weights

Based on the previous theoretical analysis, we divide the supernet training process into two interlaced stages, i.e., supernet topology generation and supernet weight update.

#### 3.2.1 Supernet Topology Generation

As described before, the optimal local weight assignment $o_A$ can be obtained by solving Eq. (6). A straightforward implementation is to assign a unique assignment id $o_{i_k}$ for each architecture. However, two architectures may only have similar structures in some layers, thus assigning a unique cluster id to the entire architecture is suboptimal. To make the local weight sharing mechanism more flexible, we propose the layer-wise architecture clustering algorithm, and subsequently optimize the local weight sharing states from the first layer to the last layer. We divide the potential network into $L$ parts (layers), with each of them containing $K$ candidate operators. We use $C_{l,k}$ to denote the $k$-th operator in the $l$-th layer. Different from the existing NAS algorithms, each operator in our work consists of $M$ candidate weight matrices which form a meta weight set $\mathcal{W}_{l,k} = \{w_{l,k}^1, w_{l,k}^2, \ldots, w_{l,k}^M\}$. Based on the above definitions, the weight parameter of the $l$-th layer in
\( \alpha_i \) is \( W_{\alpha_i,l} = W_{l,k}(\alpha_{l,i}) \), where \( \alpha_{l,i} \in \{1,2,...,M\} \) indicates which weight in \( W_{l,n} \) is exploited for architecture \( \alpha_l \). Suppose the local weight assignment states of the first \( l-1 \) layers have been obtained as \( o_{\mathcal{A}1:...:l-1} = \{o_{\mathcal{A}1:...:l-1}\} \), the optimal local weight assignment in the \( l \)-th layer can be obtained by solving
\[
\arg \max_{o_{\mathcal{A}l}} \log p(W_{\mathcal{A}1:...:l}) + \frac{\delta(o_{\mathcal{A}l}, o_{\mathcal{A}l}^*)}{(\text{7})}
\]
where \( \mathcal{W} = \{W_{l,n} \}_{l=1,n=1}^{L,N,K} \). Directly optimizing Eq. (7) by enumerating all possible \( o_{\mathcal{A}l}, i \in \{1,...,N\} \) is intractable. If the architectures are first divided into several clusters, Eq. (7) can be more easily optimized based on the few representative architectures in each cluster. For all the architectures containing \( C_{l,k} \), we first resort to the K-means clustering algorithm to divide the architectures into \( M \) clusters and then determine the weight assignment id for each cluster. Referring to [26], for any two architectures \( \alpha_i \) and \( \alpha_j \) with the same operate \( C_{l,k} \), we exploit the output \( \lambda_{i,j} \) of a meta network \( \mathcal{M}_{C_{l,k}} \) (detailed in the next section) to measure the matching degree between \( \alpha_i \) and \( \alpha_j \) in the \( l \)-th layer, where \( \lambda_{i,j} \) is computed as
\[
\alpha_{l-1}^\prime = \mathcal{M}_{C_{l,k}}(h_{l-1}^\alpha - h_{l-1}^{\alpha}) \text{, } \lambda_{i,j} = \mathcal{M}_{C_{l,k}}(h_{l-1}^\alpha - h_{l-1}^{\alpha'} \text{, if } \alpha_l \text{ and } \alpha_l' \text{ share } C_{l,k} \text{, else } 0}.
\]
and \( \lambda_{i,j} \) is
\[
\mathcal{L}(H_{x}, N(x, \alpha_i, W_{\alpha_i}^{t+1})) = \mathcal{L}(H_{x}, N(x, \alpha_i, W_{\alpha_i}^{t})) - \lambda_{i,j} \nabla_{W_{\alpha_j}} \mathcal{L}(H_{x}, N(x, \alpha_j, W_{\alpha_j}^{t}))),...\]
where \( \lambda_{i,j} \) is the learning rate for the meta network. In our implementation, the meta network is instantiated as one fully-connected layer with a sigmoid activation function. The supernet weights are updated for several epochs, and the training process continues to determine the weight assignment states for the \( l+1 \) layer. We formulate the overall training procedure in Algorithm 1.

After the supernet gets sufficiently trained, the network search process begins. We follow [14] to adopt the evolutionary search algorithm, where subnetworks within the supernet are selected and evaluated under the direction of the evolutionary controller. It is worth noting that the batch statistics of each subnetwork should be independent of the others during search. Thus, we recompute the batch statistics for each
\[
W_{f,k}(\alpha_{l,f}), \text{ where } k \text{ is the operator index of } \alpha_i \text{ in the } f \text{-th layer. Otherwise, } W_{f,k} \text{ is assigned with a random weight in } W_{f,k}. \text{ We use } H_v \text{ to denote the ground truth of batch data } x, \text{ and use } \mathcal{L}(.) \text{ to denote the loss function for model training. The variable } \gamma \text{ is the learning rate for the architecture } \alpha_i, \text{ and } \lambda_{i,j} = [\lambda_{i,j}^1, ..., \lambda_{i,j}^L] \text{ is output of the meta network, which controls the strength on how the gradient of architecture } \alpha_j \text{ influences the update process. By incorporating the third term in Eq. (8), we allow the model weights from different clusters to exchange gradient information, which boosts the training process of the supernet. It is worth noting that } \lambda_{i,j} \text{ is used to measure the matching degree between two architectures, based on the intuition that if gradients from two architectures can benefit the weight update process of each other, then these two architectures can be better matched. Using } W_{\alpha_i}^t \text{ to substitute } W_{\alpha_i} \text{, we obtain the following loss function}
\]

\[
\mathcal{L}(H_{x}, N(x, \alpha_i, W_{\alpha_i}^{t+1})) = \mathcal{L}(H_{x}, N(x, \alpha_i, W_{\alpha_i}^{t})) - \lambda_{i,j} \nabla_{W_{\alpha_j}} \mathcal{L}(H_{x}, N(x, \alpha_j, W_{\alpha_j}^{t}))),...\]

and \( \lambda_{i,j} \) is
\[
\lambda_{i,j} = \{ \mathcal{M}_{C_{l,k}}(h_{l-1}^\alpha - h_{l-1}^{\alpha'}) \text{, if } \alpha_l \text{ and } \alpha_l' \text{ share } C_{l,k} \text{, else } 0}.
\]

\[
\mathcal{L}(H_{x}, N(x, \alpha_i, W_{\alpha_i}^{t+1})) \text{ is a function with respect to } W_{\alpha_i} \text{, and } W_{\alpha_i} \text{ can be updated as}
\]

\[
W_{\alpha_i}^{t+1} = W_{\alpha_i}^t - \eta \nabla_{W_{\alpha_i}} \mathcal{L}(H_{x}, N(x, \alpha_i, W_{\alpha_i}^{t+1})),
\]

where \( \eta \) is the learning rate for the meta network. In our implementation, the meta network is instantiated as one fully-connected layer with a sigmoid activation function. The supernet weights are updated for several epochs, and the training process continues to determine the weight assignment states for the \( l+1 \) layer. We formulate the overall training procedure in Algorithm 1.
Algorithm 1: Supernet training with local weight sharing

Input: Supernet $\mathcal{N}$, search space $\mathcal{S}$, training data $D$, architecture number $N$, number of candidate states $M$, supernet layer number $L$, candidate operator number $K$, warmup epochs $T_w$, finetune epochs $T_f$, training epochs $T$, meta network update interval $\tau$, per-epoch iteration number $I$.

Output: The trained supernet model $\mathcal{N}$ with optimal weight assignment $\omega_A$

1: Initialize $\mathcal{N}$ wherein each operator $\mathcal{C}_{l,k}$ includes $M$ candidate weights
2: Randomly sample $A = \{\alpha_1, \alpha_2, ... , \alpha_N\}$, where $A \subset \mathcal{S}$
3: Train $(\mathcal{N}, D, T_w)$
4: for $l = 1: L$ do
5:     for $t = 1: (T \times I)$ do
6:         Randomly select $k \in [1, K]$
7:         Randomly sample $\alpha_i, \alpha_j$ sharing $\mathcal{C}_{l,k}$
8:         Update $W_{\alpha_i}$ according to Eq. 8
9:         if $t \mod \tau = 0$ then
10:            Update $W_{\mathcal{A} \mathcal{C}_{l,k}}$ according to Eq. 12
11:     end if
12: end for
13: for $k = 1: K$ do
14: for $\alpha_i, \alpha_j \in A$ do
15:     if $\mathcal{C}_{l,k} \in \alpha_i$ and $\mathcal{C}_{l,k} \in \alpha_j$ then
16:         Compute $\lambda_{l,j}^{i,j}$ between $\alpha_i$ and $\alpha_j$
17:     end if
18: end for
19: end for
20: Classify $A$ into $M$ clusters based on $\lambda_{l,j}^{i,j}$
21: Obtain $\alpha_i$ for each $\alpha_i \in A$ via solving Eq. 7
22: Train $(\mathcal{N}, D, T_f)$
23: end for
24: Train $(\mathcal{N}, D, T_f)$

candidate subnetwork on a subset of training dataset before evaluation. Finally, we get the performance of subnetworks ranked by the weight inherited from the supernet.

4. Experiments

In this section, we first present the implementation details of our experiments, including the dataset, search space, and training details. Then we compare our method with state-of-the-art algorithms on both the ImageNet [10] and the NAS-Bench-201 [13] datasets. At last, we conduct extensive ablation studies to verify the effectiveness of each component of the proposed algorithm.

4.1. Implementation Details

Dataset. We adopt the ImageNet dataset [10] as one of our benchmarks. The original training set of ImageNet is randomly split into two sets: 50000 images for validation (50 images for each class exactly) and the rest for training. The original validation set is used for testing, on which all the evaluation results are reported. Besides standard search spaces, we also benchmark the proposed method on the NAS-Bench-201 [13]. NAS-Bench-201 consists of 15,625 architectures in a reduced DARTS-like search space, where it has 4 internal nodes and 5 operations per node.

Search Space. As with the recent works [2, 5, 15, 18, 26, 32], we perform architecture search over the search space consisting of mobile inverted bottleneck MBConv and squeeze-excitation modules to ensure a fair comparison. There are 6 basic operators, including MBConv with kernel sizes of 3, 5 and expansion rates of 4, 5, 6. The space contains about $7.58 \times 10^{19}$ architecture candidates in total.

Supernet. We train the supernet for 150 epochs using SGD optimizer with momentum 0.9 and weight decay 4e-5. The learning rate is set to 0.5 with a linear annealing. Besides, we set $N = 10,000$, $M = 2$, $L = 16$, $K = 6$, $T_w = 10$, $T_f = 44$, $T = 3$, and $\tau = 20$. We use 8 Nvidia Tesla V100 GPUs with a batch size of 1,024 for the supernet training.

Search. We follow [14] to employ the evolutionary algorithm to search the well-performed subnetworks within the randomly initialized subnetwork set $\mathcal{A}$. Before evaluation, the batch statistics for each subnetwork is recomputed based on 200 batches of training data. We set the max iteration as 20 and the population size as 50. The mutation and crossover are performed on the Top-10 best-performing architectures for each iteration.

Retrain. Similar to the training of EfficientNet [32], our selected architecture is retrained for 500 epochs on ImageNet using RMSProp optimizer with momentum 0.9 and decay 0.9. The learning rate is set to 0.064 with a warm-up in the first 3 epochs and a cosine annealing. The dropout ratio is 0.2 and the weight decay is 1e-5. During training, AutoAugment [9] policy and exponential moving average are adopted. The model is retrained using 16 Nvidia Tesla V100 GPUs with a batch size of 2,048.

4.2. Comparisons with the State-of-the-arts

ImageNet. The quantitative comparisons with the state-of-the-arts on the ImageNet dataset are presented in Table 1 and Figure 1. We conduct experiments under two different constraints ($\text{Flops} < 350M$ and $\text{Flops} < 500M$ respectively). As can be seen, our method consistently outperforms the recent SOTA algorithms with comparable flops and training costs. In particular, the smaller model (namely Ours-S) searched under a Flops constraint of 350M achieves 78.1% Top-1 classification accuracy, which even outperforms other methods with larger Flops (e.g., 345M and 465M). Besides, with the Flops constraint of 500M, the larger model Ours-L also shows superiority over other methods. All the superior results demonstrate the effectiveness of our method.

NAS-Bench-201. The comparisons with the state-of-the-
<table>
<thead>
<tr>
<th>Methods</th>
<th>Top-1 (%)</th>
<th>Top-5 (%)</th>
<th>Flops (M)</th>
<th>Supernet train (GPU days)</th>
<th>Search cost (GPU days)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNetV2 [28]</td>
<td>72.0</td>
<td>91.0</td>
<td>300</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MobileNetV3 [15]</td>
<td>75.2</td>
<td>-</td>
<td>219</td>
<td>≈ 3000</td>
<td>-</td>
</tr>
<tr>
<td>OFA [2]</td>
<td>76.9</td>
<td>-</td>
<td>230</td>
<td>53</td>
<td>2</td>
</tr>
<tr>
<td>AtomNAS-A+ [24]</td>
<td>76.3</td>
<td>93.0</td>
<td>260</td>
<td>20.5</td>
<td>-</td>
</tr>
<tr>
<td>AKD [23]</td>
<td>73.0</td>
<td>92.2</td>
<td>300</td>
<td>-</td>
<td>1000</td>
</tr>
<tr>
<td>SPOS [14]</td>
<td>74.7</td>
<td>-</td>
<td>328</td>
<td>12</td>
<td>&lt;1</td>
</tr>
<tr>
<td>GreedyNAS-C [37]</td>
<td>76.2</td>
<td>92.5</td>
<td>284</td>
<td>7</td>
<td>&lt;1</td>
</tr>
<tr>
<td>DNA-A [18]</td>
<td>77.1</td>
<td>93.3</td>
<td>348</td>
<td>24</td>
<td>0.6</td>
</tr>
<tr>
<td>Cream-S [26]</td>
<td>77.6</td>
<td>93.3</td>
<td>287</td>
<td>12</td>
<td>0.02</td>
</tr>
<tr>
<td>FairNAS-B [7]</td>
<td>75.1</td>
<td>-</td>
<td>345</td>
<td>12</td>
<td>&lt;1</td>
</tr>
<tr>
<td>SGNAS-A [17]</td>
<td>76.2</td>
<td>-</td>
<td>281</td>
<td>12</td>
<td>&lt;1</td>
</tr>
<tr>
<td>MCT-NAS-C [30]</td>
<td>76.3</td>
<td>92.6</td>
<td>280</td>
<td>12</td>
<td>&lt;1</td>
</tr>
<tr>
<td>Ours-S</td>
<td>78.1</td>
<td>93.8</td>
<td>290</td>
<td>16</td>
<td>&lt;1</td>
</tr>
</tbody>
</table>

Table 1. Comparisons with the state-of-the-arts on the ImageNet dataset.

arts on NAS-Bench-201 are given in Table 2. All algorithms adopt the training and validation set of CIFAR-10 for architecture search and use the NAS-bench-201 API to query the ground-truth performance of searched architectures on three datasets. Our results are averaged on 4 runs of searching. As can be observed, our method outperforms the state-of-the-arts on all three datasets and our best results approach the optimal performance. The superior performance verifies the effectiveness of our algorithm.

### 4.3. Ablation Studies

In this section, extensive ablation studies are conducted to demonstrate the effectiveness of each component of our method.

**Weight sharing strategies.** We validate the effectiveness of the local weight sharing mechanism by comparing three variant implementations, which are respectively the baseline method (*Global Sharing*) that utilizes the global weight sharing supernet, the implementation with randomly assigned local weight sharing states (*Random Assign*) and our implementation. The comparison results can be referred to Table 3, where our method improves the second best implementation by a relative gain of 0.9% in terms of the Top-1 classification accuracy on the ImageNet dataset. Considering both the distribution consistent constraint and the matching degrees output by the meta network, our algorithm is able to train more distribution consistent supernet.

**Number of candidate weight states.** In our method, each operator consists of $M$ candidate weight states. We conduct experiments to analyze how $M$ influences the searching performance. As is shown in Table 4, the method achieves comparable results when $M$ is 2 and 3 respectively. Given a network with $L$ layers, there are altogether $M^L$ local weight assignment states, which provides enough flexibility for supernet training. In our implementation, we set $M$ as 2 in all of our experiments for training efficiency.

**Number of candidate architectures.** At last, we conduct ablation experiments to analyze the robustness of our method in terms of the candidate architecture number $N$. In our supernet training process, we set the initial architecture number $N = 10,000$. However, it is possible that the best-performing architecture is not included in the initial
<table>
<thead>
<tr>
<th>Methods</th>
<th>Cost (hours)</th>
<th>CIFAR-10 valid</th>
<th>CIFAR-100 valid</th>
<th>ImageNet16-120 valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>DARTS</td>
<td>3.2</td>
<td>39.77 ± 0.00</td>
<td>54.30 ± 0.00</td>
<td>15.03 ± 0.00</td>
</tr>
<tr>
<td>DARTS</td>
<td>10.2</td>
<td>39.77 ± 0.00</td>
<td>54.30 ± 0.00</td>
<td>15.03 ± 0.00</td>
</tr>
<tr>
<td>SETN</td>
<td>9.5</td>
<td>84.04 ± 0.28</td>
<td>87.64 ± 0.00</td>
<td>58.86 ± 0.06</td>
</tr>
<tr>
<td>GDAS</td>
<td>8.7</td>
<td>89.89 ± 0.08</td>
<td>93.61 ± 0.09</td>
<td>71.34 ± 0.04</td>
</tr>
<tr>
<td>FairNAS</td>
<td>2.7</td>
<td>90.07 ± 0.57</td>
<td>93.23 ± 0.18</td>
<td>70.94 ± 0.94</td>
</tr>
<tr>
<td>SGNAS</td>
<td>2.5</td>
<td>90.18 ± 0.31</td>
<td>93.53 ± 0.12</td>
<td>70.28 ± 1.20</td>
</tr>
<tr>
<td>DARTS</td>
<td>3.2</td>
<td>91.03 ± 0.44</td>
<td>93.80 ± 0.40</td>
<td>71.56 ± 1.51</td>
</tr>
<tr>
<td>Ours</td>
<td>3.9</td>
<td><strong>91.50 ± 0.07</strong></td>
<td><strong>94.29 ± 0.07</strong></td>
<td><strong>73.03 ± 0.21</strong></td>
</tr>
<tr>
<td>optimal</td>
<td>-</td>
<td>91.53</td>
<td>94.22</td>
<td>73.13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Training strategy</th>
<th>Top-1(%)</th>
<th>Top-5(%)</th>
<th>Flops(M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global-Sharing</td>
<td>77.2</td>
<td>93.2</td>
<td>291</td>
</tr>
<tr>
<td>Random-Assign</td>
<td>77.4</td>
<td>93.3</td>
<td>289</td>
</tr>
<tr>
<td>Ours</td>
<td>78.1</td>
<td>93.8</td>
<td>290</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>Top-1(%)</th>
<th>Top-5(%)</th>
<th>Flops(M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,000</td>
<td>77.7</td>
<td>93.5</td>
<td>285</td>
</tr>
<tr>
<td>10,000</td>
<td>78.1</td>
<td>93.8</td>
<td>290</td>
</tr>
<tr>
<td>20,000</td>
<td>78.0</td>
<td>93.9</td>
<td>294</td>
</tr>
</tbody>
</table>

Table 2. Comparisons with the state-of-the-arts on NAS-Bench-201. 1st: first-order, 2nd: second-order.

Table 3. Ablation study on different weight sharing strategies.

Table 4. Ablation study on the number of candidate weight states.

Table 5. Ablation study on the number of candidate architectures.

candidate pool. Thus, in this experiment, we set the candidate number to 5,000, 10,000 and 20,000 respectively. The evaluation results of searched architectures are illustrated in Table 5. As can be seen, increasing the number of candidate architectures to 20,000 does not bring further performance improvement, thus we get the conclusion that setting N = 10,000 brings a good trade-off between training efficiency and overall performance.

**Model ranking.** To further verify the effectiveness of our NAS method, we compared the model ranking abilities between our method and the global weight-sharing baseline by visualizing the relationship between performance of the stand-alone models and the models with inherited weights. Both our method and baseline are trained on NAS-Bench-201 for 150 epochs. We randomly sample 15 subnets from the search space and query their ground-truth performance using the NAS-Bench-201 API. The comparison results are shown in Figure 3. Each sampled model corresponds to 2 points in the figure, representing the correlation between the true performance and predicted performance of the two methods. As can be observed, our method ranks the subnets more precisely, which explains the reason why our method achieves better final results.

**5. Conclusion**

In this paper, we propose a novel distribution consistent neural architecture search algorithm to avoid the possible weight coupling problem. We analyze the reason why weight sharing in a supernet leads to inferior performance, and introduce a distribution consistency constraint as well as the local weight sharing mechanism in the supernet training process. Specifically, a two-stage optimization formula is derived to iteratively optimize the supernet topology and the network model parameters, which tries to figure out the optimal local weight sharing states sequentially. We conduct large amounts of experiments on different benchmark datasets and search space. The superior results validate the effectiveness of the proposed method.

**Acknowledgements.** This work is partially supported by the National Natural Science Foundation of China under Grant no. 62106149.
References


