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 manys, 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.

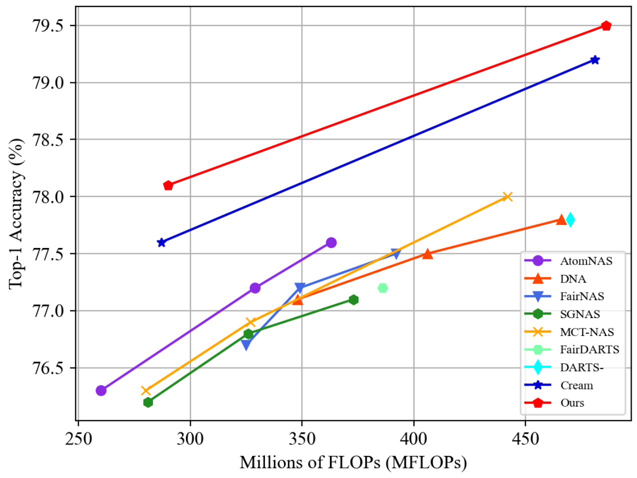

Figure 1. Comparison with state-of-the-art methods on ImageNet.

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

[*]Corresponding author.

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 subnetwork 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 one-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 α_i trained from scratch, and use $p(W_{\mathcal{A}}|\mathcal{A}, D)$ to denote the model parameter distribution of the supernet. Here, W_{α_i} is the model parameter of sub-architecture α_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

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

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

$$\begin{aligned} & p(\mathcal{A}|W_{\alpha_i}, \alpha_i, D) \\ =& \prod_{j=1}^{N} p(\alpha_j|W_{\alpha_i}, \alpha_i, D) \\ =& \frac{\prod_{j=1}^{N} p(D|W_{\alpha_i}, \alpha_i, \alpha_j)p(\alpha_j|W_{\alpha_i}, \alpha_i)}{p(D|W_{\alpha_i}, \alpha_i)} . \end{aligned} \qquad (2)$$

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

$$\begin{aligned} & \mathcal{D_{KL}}(p(W_{\alpha_i}|\alpha_i, D), p(W_{\alpha_i}|\mathcal{A}, D)) \\ =& \int p(W_{\alpha_i}|\alpha_i, D) \log \frac{p(W_{\alpha_i}|\alpha_i, D)}{p(W_{\alpha_i}|\mathcal{A}, D))} dW_{\alpha_i} \\ =& \int p(W_{\alpha_i}|\alpha_i, D) \log \frac{p(D|W_{\alpha_i}, \alpha_i)p(\mathcal{A}|\alpha_i, D)}{\prod_{j=1}^{N} p(D|W_{\alpha_i}, \alpha_i, \alpha_j)p(\alpha_j|W_{\alpha_i}, \alpha_i)} dW_{\alpha_i} \\ \propto& \int p(W_{\alpha_i}|\alpha_i, D) \log \frac{p(D|W_{\alpha_i}, \alpha_i)}{\prod_{j=1}^{N} p(D|W_{\alpha_i}, \alpha_i, \alpha_j)p(\alpha_j|W_{\alpha_i}, \alpha_i)} dW_{\alpha_i} \\ =& \int p(W_{\alpha_i}|\alpha_i, D) \log \frac{p(D|W_{\alpha_i}, \alpha_i)^{\sum_{j=1, o_{\alpha_i} \neq o_{\alpha_j}}^{N} - \mathbb{1}}}{\prod_{\substack{j=1 \\ o_{\alpha_i} = o_{\alpha_j}}}^{N} p(D|W_{\alpha_i}, \alpha_i, \alpha_j)p(\alpha_j|W_{\alpha_i}, \alpha_i)} dW_{\alpha_i} \end{aligned} \qquad (3)$$

where the local weight sharing mechanism is introduced and $o_{\alpha_i} \in o_{\mathcal{A}}$ indicates which parameter set the architecture α_i uses. Architectures α_i and α_j have shared weights if $o_{\alpha_i} = o_{\alpha_j}$. $p(\mathcal{A}|\alpha_i, D)$ and $p(\alpha_j|W_{\alpha_i}, \alpha_i)$ denote the priors of architecture set $\mathcal{A}$ and α_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_{α_i} and architectures α_i and α_j, which can be further expanded as

$$\begin{aligned} & p(D|W_{\alpha_i}, \alpha_i, \alpha_j) \\ =& \int p(W_{\alpha_j - \alpha_i})p(D|W_{\alpha_i}, W_{\alpha_j}, \alpha_i, \alpha_j)dW_{\alpha_j - \alpha_i}, \end{aligned} \qquad (4)$$

where we use $W_{\alpha_j - \alpha_i}$ to denote model parameters of α_j eliminating those shared with α_i, and $p(D|W_{\alpha_i}, W_{\alpha_j}, \alpha_i, \alpha_j)$ is the likelihood probability of architectures α_i and α_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 (α_i, α_j). When α_i and α_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

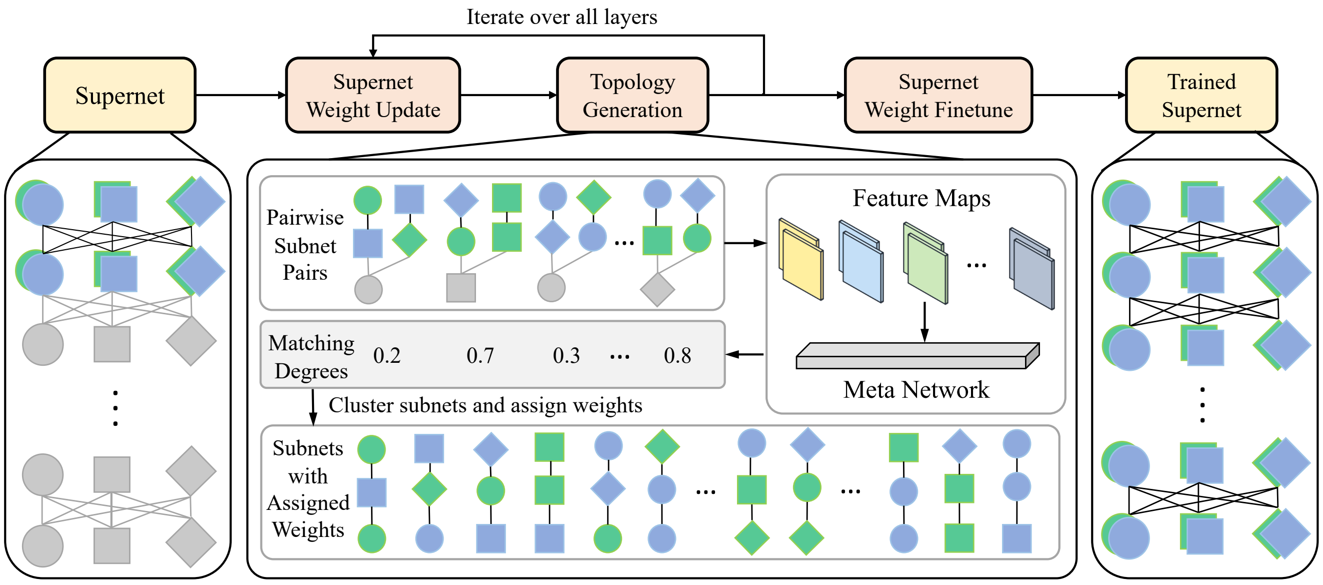

Figure 2. Supernet topology generation. The left side of the figure illustrates a toy supernet where the first 2 layers have been clustered and assigned weights. The circle, square, and rhombus denote different operators in each layer. The blue and green colors denote different weight candidates for each operator. To generate the supernet topology for the 3rd layer, we first sample the subnet pairs which share the same operators and feed the corresponding feature maps to the meta network. The meta network then predicts the matching degrees between arbitrary two subnets. A higher matching degree indicates the two subnets could share the same operator weights in this layer and vice versa. Based on the matching degrees, we are finally able to cluster the subnets and assign the weight id by optimizing Eq. 7. The above process is iterated over all layers.

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:

$$\begin{aligned} {W_{\mathcal{A}}}^* &= \arg\max_{W_{\mathcal{A}}} \log p(W_{\mathcal{A}}|\mathcal{A}, D) \\ &= \arg\max_{W_{\mathcal{A}}} \log p(D|W_{\mathcal{A}}, \mathcal{A}) \\ &\quad + \log p(W_{\mathcal{A}}|\mathcal{A}) - \log p(D|\mathcal{A}), \end{aligned} \tag{5}$$

where $p(W_{\mathcal{A}}|\mathcal{A})$ and $p(D|\mathcal{A})$ 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

$$\begin{aligned} {W_{\mathcal{A}}}^*, {o_{\mathcal{A}}}^* = \arg\max_{W_{\mathcal{A}}, \boldsymbol{o_{\mathcal{A}}}} & \log p(D|W_{\mathcal{A}}, \mathcal{A}, o_{\mathcal{A}}) \\ + \delta(o_{\alpha_i}, o_{\alpha_j}) & \sum_{i,j=1}^{N} \log p(D|W_{\alpha_i}, W_{\alpha_j}, \alpha_i, \alpha_j). \end{aligned} \tag{6}$$

The optimization process in Eq. (6) can be solved via the alternating direction method, *i.e.*, optimizing the optimal local weights assignment $o_{\mathcal{A}}$ 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 $o_{\mathcal{A}}$ can be obtained by solving Eq. (6). A straightforward implementation is to assign a unique assignment id o_{α_i} 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 $\mathcal{C}_{l,k}$ 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 $\mathcal{W}_{l,k} = \left\{w_{l,k}^1, w_{l,k}^2, ..., w_{l,k}^M\right\}$. Based on the above definitions, the weight parameter of the l-th layer in

α_i is $W_{\alpha_i,l} = \mathcal{W}_{l,k}(o_{\alpha_i^l})$, where $o_{\alpha_i^l} \in \{1, 2, ..., M\}$ indicates which weight in $\mathcal{W}_{l,n}$ is exploited for architecture α_i. Suppose the local weight assignment states of the first $l-1$ layers have been obtained as $o_{\mathcal{A}^{1,..,l-1}} = \{o_{\alpha_1^{l-1},...,\alpha_N^{l-1}}\}$, the optimal local weight assignment in the l-th layer can be obtained by solving

$$\begin{aligned} o^*_{\mathcal{A}^l} =& \arg\max_{\boldsymbol{o}_{\mathcal{A}^l}} \log p(D|\mathcal{W}, o_{\mathcal{A}^{1,...,l}}) \\ &+ \delta(o_{\alpha_i^l}, o_{\alpha_j^l}) \sum_{i,j=1}^{N} \log p(D|\mathcal{W}, \alpha_i, \alpha_j, o_{\mathcal{A}^{1,...,l-1}}), \end{aligned} \tag{7}$$

where $\mathcal{W} = \{\mathcal{W}_{l,n}\}_{l=1,n=1}^{l=L,n=K}$. Directly optimizing Eq. (7) by enumerating all possible $o_{\alpha_i^l}, i \in \{1, ..., N\}$ 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 $\mathcal{C}_{l,k}$, 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 α_i and α_j with the same operate $\mathcal{C}_{l,k}$, we exploit the output $\lambda^l_{i,j}$ of a meta network $\mathcal{M}_{\mathcal{C}_{l,k}}$ (detailed in the next section) to measure the matching degree between α_i and α_j in the l-th layer, where $\lambda^l_{i,j} = \mathcal{M}_{\mathcal{C}_{l,k}}(h^{l-1}_{\alpha_i} - h^{l-1}_{\alpha_j})$. Here, $h^{l-1}_{\alpha_i} = \mathcal{N}(x, \alpha_i^{1,...,l-1}, W_{\alpha_j^{1,...,l-1}})$ denotes the output feature map of the $l-1$-th layer for architecture α_i. We compute the matching degrees between arbitrary two architectures containing operator $\mathcal{C}_{l.k}$, and obtain a matching degree matrix, based on which the architectures are classified into M clusters via the K-means clustering algorithm. Then, we randomly sample several representative architectures from each cluster and obtain the weight assignments for these architectures via solving Eq. (7). Based on the representative architectures in each cluster, we obtain the optimal assignment id for all the candidate architectures. Figure 2 presents a toy model on how our method sequentially determines the local weight sharing states.

3.2.2 Supernet Weight Update

When the local weight sharing states of the previous l layers are determined, we update the supernet weight via the stochastic gradient descent algorithm. For each training batch x, two architectures α_i and α_j with at least one shared operator are sampled. Then the model parameter of α_i in the f-th layer can be updated as

$$\begin{aligned} W^{t+1}_{\alpha_i} =& W^t_{\alpha_i} - \gamma \nabla_{W_{\alpha_i}} \mathcal{L}(\mathbf{H}_x, \mathcal{N}(x, \alpha_i, W^t_{\alpha_i})) \\ &- \lambda_{i,j} \nabla_{W_{\alpha_j}} \mathcal{L}(\mathbf{H}_x, \mathcal{N}(x, \alpha_j, W^t_{\alpha_j})), \end{aligned} \tag{8}$$

where the gradient from two architectures are considered. During model update, when the layer index $f \leq l$, the model parameter $W_{\alpha_i,f}$ in the f-th layer is assigned to with $\mathcal{W}_{f,k}(o_{\alpha_{i,f}})$, where k is the operator index of α_i in the f-th layer. Otherwise, $W_{\alpha_i,f}$ is assigned with a random weight in $\mathcal{W}_{f,k}$. We use $\mathbf{H}_x$ to denote the ground truth of batch data x, and use $\mathcal{L}(.)$ to denote the loss function for model training. The variable γ is the learning rate for the architecture α_i, and $\lambda_{i,j} = [\lambda^1_{i,j}, ..., \lambda^2_{i,j}]$ is output of the meta network, which controls the strength on how the gradient of architecture α_j 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 $\lambda^f_{i,j}$ 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 $W^{t+1}_{\alpha_i}$ to substitute $W^t_{\alpha_i}$, we obtain the following loss function

$$\begin{aligned} &\mathcal{L}(\mathbf{H}_x, \mathcal{N}(x, \alpha_i, W^{t+1}_{\alpha_i})) \\ =& \mathcal{L}(\mathbf{H}_x, \mathcal{N}(x, \alpha_i, W^t_{\alpha_i} - \gamma \nabla_{W_{\alpha_i}} \mathcal{L}(\mathbf{H}_x, \mathcal{N}(x, \alpha_i, W^t_{\alpha_i})) \\ &- \lambda_{i,j} \nabla_{W_{\alpha_j}} \mathcal{L}(\mathbf{H}_x, \mathcal{N}(x, \alpha_j, W^t_{\alpha_j})))), \end{aligned} \tag{9}$$

where $\lambda_{i,j} \nabla_{W_{\alpha_i}} \mathcal{L}(\mathbf{H}_x, \mathcal{N}(x, \alpha_j, W^t_{\alpha_j}))$ is computed as

$$\begin{aligned} &\lambda_{i,j} \nabla_{W_{\alpha_i}} \mathcal{L}(\mathbf{H}_x, \mathcal{N}(x, \alpha_j, W^t_{\alpha_j})) \\ =& [\lambda^1_{i,j} \nabla_{W_{\alpha_i^1}} \mathcal{L}(\mathbf{H}_x, \mathcal{N}(x, \alpha_j, W^t_{\alpha_j^1})), ..., \\ &\lambda^L_{i,j} \nabla_{W_{\alpha_i^L}} \mathcal{L}(\mathbf{H}_x, \mathcal{N}(x, \alpha_j, W^t_{\alpha_j^L}))]^\top \end{aligned} \tag{10}$$

and $\lambda^l_{i,j}$ is

$$\lambda^l_{i,j} = \begin{cases} \mathcal{M}_{\mathcal{C}_{l,k}}(h^{l-1}_{\alpha_i} - h^{l-1}_{\alpha_j}, W_{\mathcal{M}_{\mathcal{C}_{l,k}}}), & \alpha^l_i \text{ and } \alpha^l_j \text{ share } \mathcal{C}_{l,k} \\ 0, & \text{else} \end{cases}. \tag{11}$$

$\mathcal{L}(\mathbf{H}_x, \mathcal{N}(x, \alpha_i, W^{t+1}_{\alpha_i}))$ is a function with respect to $W_{\mathcal{M}_{\mathcal{C}_{l,k}}}$, and $W_{\mathcal{M}_{\mathcal{C}_{l,k}}}$ can be updated as

$$W^{t+1}_{\mathcal{M}_{\mathcal{C}_{l,k}}} = W^t_{\mathcal{M}_{\mathcal{C}_{l,k}}} - \eta \nabla_{W_{\mathcal{M}_{\mathcal{C}_{l,k}}}} \mathcal{L}(\mathbf{H}_x, \mathcal{N}(x, \alpha_i, W^{t+1}_{\alpha_i})), \tag{12}$$

where η 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 statics of each subnetwork should be independent of the others during search. Thus, we recompute the batch statics for each

Algorithm 1 Supernet training with local weight sharing

Input: Supernet $\mathcal{N}$, search space $\mathbb{S}$, 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 τ, per-epoch iteration number I.

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

```
1:  Initialize $\mathcal{N}$ wherein each operator $\mathcal{C}_{l,k}$ includes $M$ candidate weights
2:  Randomly sample $\mathcal{A} = \{\alpha_1, \alpha_2, ...\alpha_N\}$, where $\mathcal{A} \subset \mathbb{S}$
3:  Train ($\mathcal{N}$, $D$, $T_w$)
4:  for $l = 1 : L$ do
5:      for $t = 1 : (T \times I)$ do
6:          Randomly sample $k \in [1, K]$
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 $W_{\mathcal{M}_{\mathcal{C}_{l,k}}}$ according to Eq. 12
11:         end if
12:     end for
13:     for $k = 1 : K$ do
14:         for $\alpha_i, \alpha_j \in \mathcal{A}$ do
15:             if $\mathcal{C}_{l,k} \in \alpha_i$ and $\mathcal{C}_{l,k} \in \alpha_j$ then
16:                 Compute $\lambda^l_{i,j}$ between $\alpha_i$ and $\alpha_j$
17:             end if
18:         end for
19:     end for
20:     Classify $\mathcal{A}$ into $M$ clusters based on $\lambda^l_{i,j}$
21:     Obtain $o_{\alpha^l_i}$ for each $\alpha_i \in \mathcal{A}$ via solving Eq. 7
22:     Train ($\mathcal{N}$, $D$, $T$)
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 7.58×10^{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 statics 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 (Flops $<$ 350M and 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-

	Methods	Top-1 (%)	Top-5 (%)	Flops (M)	Supernet train (GPU days)	Search cost (GPU days)
200-350M Flops	MobileNetV2 [28]	72.0	91.0	300	-	-
	MobileNetV3$_{L1.0}$ [15]	75.2	-	219	$\approx$ 3000	-
	OFA [2]	76.9	-	230	53	2
	AtomNAS-A+ [24]	76.3	93.0	260	20.5	-
	AKD [23]	73.0	92.2	300	-	1000
	SPOS [14]	74.7	-	328	12	<1
	GreedyNAS-C [37]	76.2	92.5	284	7	<1
	DNA-A [18]	77.1	93.3	348	24	0.6
	Cream-S [26]	77.6	93.3	287	12	0.02
	FairNAS-B [7]	75.1	-	345	12	<1
	SGNAS-A [17]	76.2	-	281	12	<1
	MCT-NAS-C [30]	76.3	92.6	280	12	<1
	Ours-S	**78.1**	**93.8**	290	16	<1
350-500M Flops	EfficientNet-B0 [32]	76.3	93.2	390	$\approx$ 3000	-
	ProxylessNAS [3]	75.1	-	465	15	-
	MnasNet-92 [31]	74.8	92.1	388	-	-
	AtomNAS-C+ [24]	77.6	93.6	363	20.5	-
	GreedyNAS-A [37]	77.1	93.3	366	7	<1
	MixNet-M [33]	77.0	93.3	360	$\approx$ 3000	-
	DNA-C [18]	77.8	93.7	466	24	0.6
	SCARLET-A [6]	76.9	93.4	365	10	12
	DSNAS [16]	74.3	91.9	324	-	-
	FairDARTS-C [8]	77.2	93.5	386	3	-
	Cream-M [26]	79.2	94.2	481	12	0.02
	FairNAS-A [7]	75.3	-	388	12	<1
	DARTS- [5]	77.8	93.9	470	4.5	-
	SGNAS-C [17]	77.1	-	373	12	<1
	MCT-NAS-A [30]	78.0	93.9	442	12	<1
	Ours-L	**79.5**	**94.5**	486	19	<1

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 (*Gloabl 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

Methods	Cost (hours)	CIFAR-10		CIFAR-100		ImageNet16-120	
		valid	test	valid	test	valid	test
DARTS^{1st} [22]	3.2	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
DARTS^{2nd} [22]	10.2	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
SETN [11]	9.5	84.04±0.28	87.64±0.00	58.86±0.06	59.05±0.24	33.06±0.02	32.52±0.21
GDAS [12]	8.7	89.89±0.08	93.61±0.09	71.34±0.04	70.70±0.30	41.59±1.33	41.71±0.98
FairNAS [7]	2.7	90.07±0.57	93.23±0.18	70.94±0.94	71.00±1.46	41.90±1.00	42.19±0.31
SGNAS [17]	2.5	90.18±0.31	93.53±0.12	70.28±1.20	70.31±1.09	44.65±2.32	44.98±2.10
DARTS- [5]	3.2	91.03±0.44	93.80±0.40	71.36±1.51	71.53±1.51	44.87±1.46	45.12±0.82
Ours	3.9	**91.50±0.07**	**94.29±0.07**	**73.03±0.21**	**73.02±0.16**	**46.17±0.36**	**46.41±0.14**
Ours-best	3.9	91.53	94.22	73.13	73.17	46.32	46.48
optimal	-	91.61	94.37	73.49	73.51	46.77	47.31

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

Training strategy	Top-1(%)	Top-5(%)	Flops(M)
Global-Sharing	77.2	93.2	291
Random-Assign	77.4	93.3	289
Ours	78.1	93.8	290

Table 3. Ablation study on different weight sharing strategies.

M	Top-1(%)	Top-5(%)	Flops(M)
1	77.2	93.2	291
2	78.1	93.8	290
3	77.9	93.8	292

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

N	Top-1(%)	Top-5(%)	Flops(M)
5,000	77.7	93.5	285
10,000	78.1	93.8	290
20,000	78.0	93.9	294

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

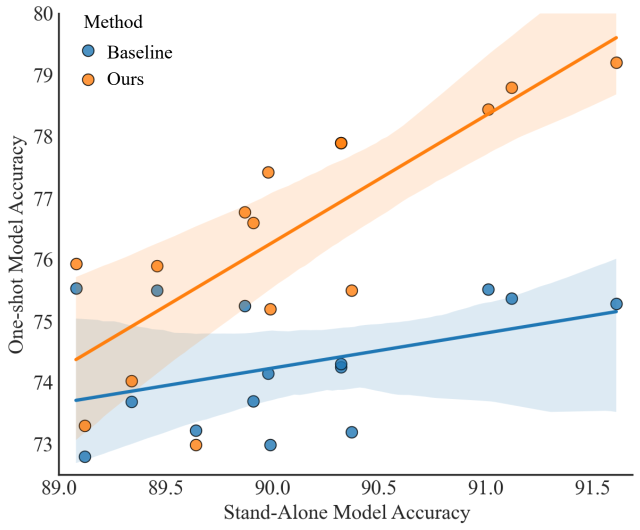

Figure 3. Comparison of ranking effectiveness for our method and the global weight-sharing baseline.

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

[1] Peter J Angeline, Gregory M Saunders, and Jordan B Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE transactions on Neural Networks*, 5(1):54–65, 1994.

[2] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *ICLR*, 2020.

[3] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019.

[4] Yukang Chen, Tong Yang, Xiangyu Zhang, Gaofeng Meng, Xinyu Xiao, and Jian Sun. Detnas: Backbone search for object detection. *arXiv preprint arXiv:1903.10979*, 2019.

[5] Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. Darts-: Robustly stepping out of performance collapse without indicators. In *ICLR*, 2021.

[6] Xiangxiang Chu, Bo Zhang, Qingyuan Li, Ruijun Xu, and Xudong Li. Scarlet-nas: bridging the gap between stability and scalability in weight-sharing neural architecture search. In *ICCV workshop*, pages 317–325, 2021.

[7] Xiangxiang Chu, Bo Zhang, and Ruijun Xu. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. In *ICCV*, pages 12239–12248, 2021.

[8] Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair darts: Eliminating unfair advantages in differentiable architecture search. In *ECCV*, pages 465–480, 2020.

[9] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *CVPR*, pages 113–123, 2019.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.

[11] Xuanyi Dong and Yi Yang. One-shot neural architecture search via self-evaluated template network. In *ICCV*, pages 3681–3690, 2019.

[12] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *CVPR*, pages 1761–1770, 2019.

[13] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *ICLR*, 2020.

[14] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *ECCV*, pages 544–560, 2020.

[15] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *ICCV*, pages 1314–1324, 2019.

[16] Shoukang Hu, Sirui Xie, Hehui Zheng, Chunxiao Liu, Jianping Shi, Xunying Liu, and Dahua Lin. Dsnas: Direct neural architecture search without parameter retraining. In *CVPR*, pages 12084–12092, 2020.

[17] Sian-Yao Huang and Wei-Ta Chu. Searching by generating: Flexible and efficient one-shot nas with architecture generator. In *CVPR*, 2021.

[18] Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. Block-wisely supervised neural architecture search with knowledge distillation. In *CVPR*, pages 1989–1998, 2020.

[19] Xiang Li, Chen Lin, Chuming Li, Ming Sun, Wei Wu, Junjie Yan, and Wanli Ouyang. Improving one-shot nas by suppressing the posterior fading. In *CVPR*, pages 13836–13845, 2020.

[20] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *CVPR*, pages 82–92, 2019.

[21] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, pages 19–34, 2018.

[22] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *ICLR*, 2019.

[23] Yu Liu, Xuhui Jia, Mingxing Tan, Raviteja Vemulapalli, Yukun Zhu, Bradley Green, and Xiaogang Wang. Search to distill: Pearls are everywhere but not the eyes. In *CVPR*, pages 7539–7548, 2020.

[24] Jieru Mei, Yingwei Li, Xiaochen Lian, Xiaojie Jin, Linjie Yang, Alan Yuille, and Jianchao Yang. Atomnas: Fine-grained end-to-end neural architecture search. In *ICLR*, 2020.

[25] Geoffrey F Miller, Peter M Todd, and Shailesh U Hegde. Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pages 379–384, 1989.

[26] Houwen Peng, Hao Du, Hongyuan Yu, QI LI, Jing Liao, and Jianlong Fu. Cream of the crop: Distilling prioritized paths for one-shot neural architecture search. *NeurIPS*, 33, 2020.

[27] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *ICML*, pages 4095–4104, 2018.

[28] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, pages 4510–4520, 2018.

[29] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[30] Xiu Su, Tao Huang, Yanxi Li, Shan You, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. Prioritized architecture sampling with monto-carlo tree search. In *CVPR*, pages 10968–10977, 2021.

[31] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, pages 2820–2828, 2019.

[32] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *ICML*, pages 6105–6114, 2019.

[33] Mingxing Tan and Quoc V Le. Mixconv: Mixed depthwise convolutional kernels. In *BMVC*, 2019.

[34] Jiaxing Wang, Haoli Bai, Jiaxiang Wu, Xupeng Shi, Junzhou Huang, Irwin King, Michael Lyu, and Jian Cheng. Revisiting parameter sharing for automatic neural channel number search. *NeurIPS*, 2020.

[35] Ning Wang, Yang Gao, Hao Chen, Peng Wang, Zhi Tian, Chunhua Shen, and Yanning Zhang. Nas-fcos: Fast neural architecture search for object detection. In *CVPR*, pages 11943–11951, 2020.

[36] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[37] Shan You, Tao Huang, Mingmin Yang, Fei Wang, Chen Qian, and Changshui Zhang. Greedynas: Towards fast one-shot nas with greedy supernet. In *CVPR*, pages 1999–2008, 2020.

[38] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. In *ECCV*, pages 702–717, 2020.

[39] Xiong Zhang, Hongmin Xu, Hong Mo, Jianchao Tan, Cheng Yang, Lei Wang, and Wenqi Ren. Dcnas: Densely connected neural architecture search for semantic image segmentation. *arXiv preprint arXiv:2003.11883*, 2020.

[40] Yiyang Zhao, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo. Few-shot neural architecture search. In *ICML*, pages 12707–12718, 2021.

[41] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[42] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, pages 8697–8710, 2018.