BCNet: Searching for Network Width with Bilaterally Coupled Network

Xiu Su[1], Shan You[2,3]*, Fei Wang[2], Chen Qian[2], Changshui Zhang[3], Chang Xu[1]*
[1]School of Computer Science, Faculty of Engineering, The University of Sydney, Australia
[2]SenseTime Research
[3]Department of Automation, Tsinghua University,
Institute for Artificial Intelligence, Tsinghua University (THUAI),
Beijing National Research Center for Information Science and Technology (BNRist)
xisu5992@uni.sydney.edu.au, {youshan,wangfei,qianchen}@sensetime.com
zcs@mail.tsinghua.edu.cn, c.xu@sydney.edu.au

Abstract

Searching for a more compact network width recently serves as an effective way of channel pruning for the deployment of convolutional neural networks (CNNs) under hardware constraints. To fulfill the searching, a one-shot supernet is usually leveraged to efficiently evaluate the performance w.r.t. different network widths. However, current methods mainly follow a unilaterally augmented (UA) principle for the evaluation of each width, which induces the training unfairness of channels in supernet. In this paper, we introduce a new supernet called Bilaterally Coupled Network (BCNet) to address this issue. In BCNet, each channel is fairly trained and responsible for the same amount of network widths, thus each network width can be evaluated more accurately. Besides, we leverage a stochastic complementary strategy for training the BCNet, and propose a prior initial population sampling method to boost the performance of the evolutionary search. Extensive experiments on benchmark CIFAR-10 and ImageNet datasets indicate that our method can achieve state-of-the-art or competing performance over other baseline methods. Moreover, our method turns out to further boost the performance of NAS models by refining their network widths. For example, with the same FLOPs budget, our obtained EfficientNet-B0 achieves 77.36% Top-1 accuracy on ImageNet dataset, surpassing the performance of original setting by 0.48%.

1. Introduction

For practical deployment of convolutional neural networks (CNNs), it is important to consider different hardware budgets [12, 10, 11, 30], to name a few, floating point operations (FLOPs), latency, memory footprint and energy consumption. One way to simultaneously accommodate all these budgets is to prune the redundant channels of a model, so that a compact network width can be obtained. Typical channel pruning usually leverages a pre-trained network and implement the pruning in an end-to-end [21, 24, 32] or layer-by-layer [16, 36] manner. After pruning, the structure of the pre-trained model remains unchanged, so that the pruned network is friendly to off-the-shelf deep learning frameworks and can be further boosted by other techniques, such as quantization [12] and knowledge distillation [17, 41, 22].

Recently, [26] found the core of channel pruning is to learn a more compact *network width* instead of the retained weights. Other literature also uses number of channels/filters to indicate the network width. Thus follow-up work resorts to neural architecture search (NAS) [38, 40, 39, 1] or other automl techniques for directly searching for an optimal network width, such as MetaPruning [25], AutoSlim [43] and TAS [6]. In their methods, a one-shot supernet is usually leveraged for evaluation of different widths. Concretely, for the width c at a certain layer, we need to assign c channels (filters) in the layer and all layers follow the same way. Then all these assigned channels in the supernet specify a sub-network with the supernet. As a result, the performance of a network width refers to the accuracy of the specified sub-network with shared weights of supernet. For fair evaluation of different network widths, during the training of supernet, all network widths will be evenly sampled from the supernet and get optimized accordingly. For brevity, we use the name of layer width to indicate the width for a certain layer, while network width represents the set of widths for all layers.

In this way, how to specify the sub-network(s) for each network width matters for the performance evaluation. However, current methods [25, 43, 6] mainly follow a

*Corresponding authors.

unilaterally augmented (UA) principle for the evaluation of network widths in supernet. Suppose we count channels in a layer from the left to the right as Figure 1. To evaluate the width c, UA principle simply assigns the leftmost c channels to specify a sub-network for evaluation. In this way, channels within smaller width will also be used for evaluation of larger widths. Since we uniformly sample all widths during training the supernet, channels close to left side will be used more times than those close to the right side in the evaluation of widths as in Figure 1(a). For example, the leftmost channel will be used 6 times for evaluation while the rightmost channel is only used once. This causes *training unfairness* among the channels and their corresponding kernels. Left channels will be trained more than right ones. Nevertheless, this training unfairness will affect the accuracy of evaluation, and thus hampers the ability of supernet to rank over all network widths.

In this paper, we introduce a new supernet called Bilaterally Coupled Network (BCNet) to address the training and evaluation unfairness within UA principle. In BCNet, each channel is fairly trained and responsible for the same amount of widths. Specifically, both in training and evaluation, each width is determined symmetrically by the average performance of bilateral (*i.e.*, both left and right) channels. As shown in Figure 1(b), suppose a layer has 6 channels, then each channel of BCNet evenly corresponds to 7 layer widths from the left or right side. In this way, all channels will be trained equally; all widths are bilaterally coupled in BCNet and will be evaluated more fairly.

To encourage a rigorous training fairness over channels, we adopt a complementary training strategy for training BCNet as in Figure 2. As for the subsequent searching, since the evolutionary algorithm is empirically fairly sensitive to the initial population, we also propose a prior sampling method, which enables to generate a good and steady initial population instead of random initialization. Extensive experiments on the benchmark CIFAR-10 and ImageNet datasets show that our method outperforms the state-of-the-art methods under various FLOPs budget. For example, our searched EfficientNet-B0 achieves 74.9% Top-1 accuracy on ImageNet dataset with 192M FLOPs ($2\times$ acceleration).

2. Related Work

Channel pruning is an effective method to compress and accelerate an over-parameterized convolutional neural network, and thus enables the pruned network to accommodate various hardware computational budgets. Extensive studies are illustrated in the comprehensive survey [33]. Here, we summarize the typical approaches of channel pruning [21, 24, 16, 35] and network width search methods [25, 43, 6].

Channel pruning. Channel pruning is an prevalent method which aims to reduce redundant channels of an heavy model, and generally implemented by selecting significant channels [24, 16] or adding additional data-driven sparsity [21, 44, 37]. For example, CP [16] propose to construct a group Lasso to select unimportant channels. Slimming [24] impose a l_1 regularization on the scaling factors. DCP [44] propose to construct an additional discrimination-aware losses. Despite the achievements, these methods rely heavily on manually assigned pruning ratios or hyperparameter coefficients, which is complicated, time consuming and hardly to find Oracle solutions.

Network width search. Inspired by the development of NAS [1, 8, 20], network width search methods [25, 43, 6, 9, 14, 31] generally take a carefully designed one-shot supernet to rank the relative performance of different widths. For example, TAS [6] aims to search the optimal network width via a learnable continuous parameter distribution. MetaPruning [25] proposes to directly generate representative weights for different widths. AutoSlim [43] proposes to leverage a slimmable network to approximate the accuracy of different network widths. However, all these methods follow UA principle in assigning channels, which affects the fairness in evaluation. To accurately rank the performance of network widths, our proposed BCNet aims to assign the same opportunity for channels during training, thus ensures the evaluation fairness in searching optimal widths.

3. Channel Pruning as Network Width Search

Formally, suppose the target network to be pruned $\mathcal{N}$ has L layers, and each layer has l_i channels. Then channel pruning aims to identify redundant channels (indexed by $\mathcal{I}^i_{pruned}$) layer-wisely, *i.e.*,

$$\mathcal{I}^i_{pruned} \subset [1 : l_i], \tag{1}$$

where $[1 : l_i]$ is an index set for all integers in the range of 1 to l_i for i-th layer. However, [26] empirically finds that the absolute set of pruned channels $\mathcal{I}^i_{pruned}$ and their weights are not really necessary for the performance of pruned network, but the obtained width c_i actually matters, *i.e.*,

$$c_i = l_i - |\mathcal{I}^i_{pruned}|. \tag{2}$$

In this way, it is intuitive to directly search for the optimal network width to meet the given budgets.

Denote an arbitrary network width as $\boldsymbol{c} = (c_1, c_2, ..., c_L) \in \mathcal{C} = \bigotimes_{i=1}^{L}[1 : l_i]$, where $\bigotimes$ is the Cartesian product. Then the size of search space $\mathcal{C}$ amounts to $|\mathcal{C}| = \prod_{i=1}^{L} l_i$. However, this search space is fairly huge, *e.g.*, 10^{25} for $L = 25$ layers and $l_i = 10$ channels. To reduce the search space, current methods tend to search on a group level instead of channel-wise level. In specific, all channels at a layer is partitioned evenly into K groups, then we only need to consider K cases; there are just

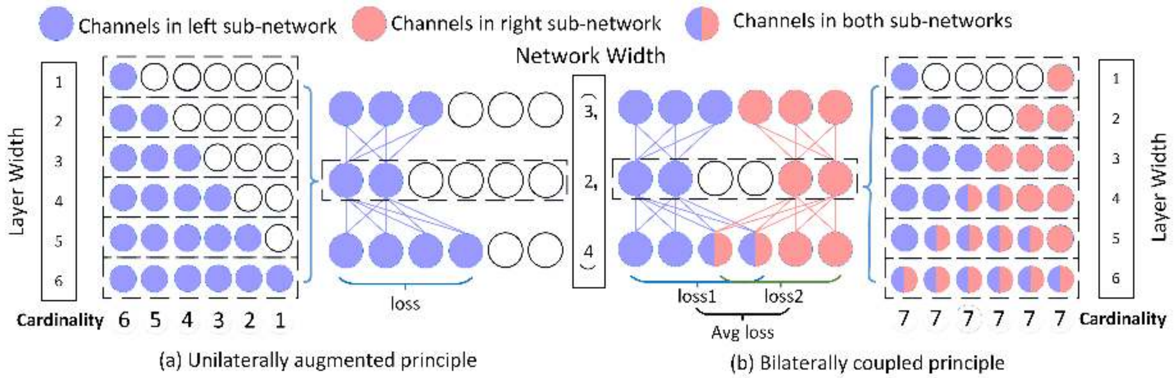

Figure 1. Comparison of unilaterally augmented (UA) principle and our proposed bilaterally coupled (BC) principle in supernet. In BC principle, each network width is indicated by two (left and right) paths, so that all channels get the same cardinality for evaluation different widths. However, in UA principle each width goes through one path, and training unfairness over channels and evaluation bias exist. Under uniform sampling strategy, for each channel the expectation of the times being evaluated is theoretically equal to the times being trained, since we simply sample each path and train it. For simplicity, we use *cardinality* to refer to the number of times that a channel is used for evaluation over all widths.

$(l_i/K) \cdot [1:K]$ layer widths for i-th layer. Therefore, the search space $\mathcal{C}$ is shrunk into $\mathcal{C}_K$ with size $|\mathcal{C}_K| = K^L$. In the following, we use both $\mathcal{C}$ and $\mathcal{C}_K$ seamlessly.

During searching, the target network is usually leveraged as a supernet $\mathcal{N}$, and different network widths $\boldsymbol{c}$ can be directly evaluated by sharing the same weights with the supernet. Then the width searching can be divided into two steps, i.e., supernet training, and searching with supernet. Usually, the original training dataset is split into two datasets, *i.e.*, training dataset $\mathcal{D}_{tr}$ and validation dataset $\mathcal{D}_{val}$. The weights $\mathcal{W}$ of the target supernet $\mathcal{N}$ is trained by uniformly sampling a width c and optimizing its corresponding subnetwork with weights with weights $w_c \subset W$,

$$W^* = \underset{\boldsymbol{w_c} \subset W}{\arg\min} \underset{\boldsymbol{c} \in U(\mathcal{C})}{\mathbb{E}} [\![\mathcal{L}_{tr}(\boldsymbol{w}_c; \mathcal{N}, \boldsymbol{c}, \mathcal{D}_{tr})]\!], \qquad (3)$$

where $\mathcal{L}_{train}$ is the training loss function, $U(\mathcal{C})$ is a uniform distribution of network widths, and $\mathbb{E}[\![\cdot]\!]$ is the expectation of random variables. Then the optimal network width $\boldsymbol{c}^*$ corresponds to the one with best performance on validation dataset, *e.g.* classification accuracy,

$$\begin{aligned} \boldsymbol{c}^* = & \underset{\boldsymbol{c} \in \mathcal{C}}{\arg\max} \ \text{Accuracy}(\boldsymbol{c}, \boldsymbol{w}_{\boldsymbol{c}}^*; W^*, \mathcal{N}, \mathcal{D}_{val}), \\ & \text{s.t. FLOPs}(\boldsymbol{c}) \leq F_b, \end{aligned} \qquad (4)$$

where F_b is a specified budget of FLOPs. Here we consider FLOPs rather than latency as the hardware constraint since we are not targeting any specific hardware device like EfficientNet [34] and other pruning baselines [44, 6, 14]. The searching of Eq.(4) can be fulfilled efficiently by various algorithms, such as random or evolutionary search [25]. Afterwards, the performance of the searched optimal width $\boldsymbol{c}^*$ is analyzed by training from scratch.

4. BCNet: Bilaterally Coupled Network

4.1. BCNet as a new supernet

As illustrated previously, for evaluation of width c at certain layer, unilaterally augmented (UA) principle assigns the leftmost c channels to indicate its performance as Figure 1(a). Hence all channels used for width c can be indexed by a set $\mathcal{I}_{UA}(c)$, *i.e.*,

$$\mathcal{I}_{UA}(c) = [1:c]. \qquad (5)$$

However, UA principle imposes an unfairness in updating channels (filters) for supernet. Channels with small index will be assigned to both small and large widths. Since different widths are sampled uniformly during the training of supernet, kernels for channels with smaller index thus get more training accordingly. To quantify this training unfairness, we can use the number of times that a channel is used for the evaluation of all widths to reflect its *training degree*, and we name it as *cardinality*. Suppose a layer has maximum l channels, then the cardinality for the c-th channel in UA principle is

$$\text{Card-UA}(c) = l - c + 1. \qquad (6)$$

In this way, the cardinality of all channels varies significantly and thus they get trained much differently, which introduces evaluation bias when we use the trained supernet to rank the performance over all widths.

To alleviate the evaluation bias over widths, our proposed BCNet serves as a new supernet which promote the fairness w.r.t. channels. As shown in Figure 1(b), in BCNet each width is simultaneously evaluated by the sub-networks corresponding to left and right channels. left and right channels. It can be seen as two identical networks $\mathcal{N}_l$ and $\mathcal{N}_r$ bilaterally coupled with each other, and use UA principle for evaluation but in a reversed order of counting channels.

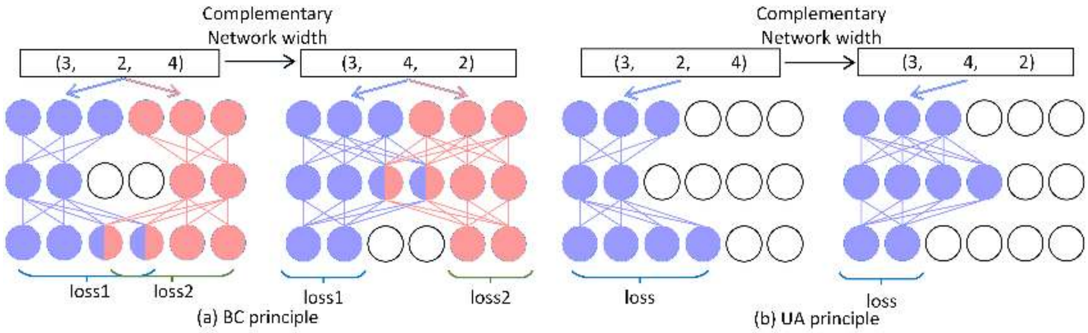

Figure 2. Illustration of the complementary of a network width for both our bilaterally coupled (BC) principle and the baseline unilaterally augmented (UA) principle. In BC principle, for any width $\boldsymbol{c}$, all channels will be trained evenly (2 times) by training $\boldsymbol{c}$ and its complementary together. However, this fairness can not be ensured in UA principle, but gets worse; some channels will be trained 2 times while others will be trained one or zero time.

In this way, all channels $\mathcal{I}_{BC}(c)$ used for evaluating width c in BCNet are indexed by

$$\mathcal{I}_{BC}(c) = \mathcal{I}^{l}_{UA}(c) \uplus \mathcal{I}^{r}_{UA}(c) \tag{7}$$
$$= [1:c] \uplus [(l-c+1):(l-c)], \tag{8}$$

where $\uplus$ means the merge of two lists with repeatable elements. In detail, left channels in $\mathcal{N}_l$ follow the same setting with UA principle as Eq.(5), while for right channels in $\mathcal{N}_r$, we count channels starting from right with $\mathcal{I}^{r}_{UA}(c) = [(l-c+1):(l-c)]$. As a result, the cardinality of each channel within BC principle is the sum from both two supernets $\mathcal{N}_l$ and $\mathcal{N}_r$. In detail, since channels count from the right side within $\mathcal{N}_r$, the cardinality for the c-th channel in left side corresponds to the cardinality of $l-c+1$-th channel in right side with Eq.(6). As a result, the cardinality for the c-th channel in BC principle is

$$\text{Card-BC}(c) = \text{Card-UA}(c) + \text{Card-UA}(l+1-c)$$
$$= (l-c+1) + (l+1-l-1+c) = l+1 \tag{9}$$

Therefore, the cardinality for each channel will always amounts to the same constant value (*i.e.*, 7 in Figure 1(b)) of widths, and irrelevant with the index of channels with BC principle, thus ensuring the fairness in terms of channel (filter) level, which promotes to fairly rank network widths with our BCNet.

4.2. Stochastic Complementary Training Strategy

To train the BCNet, we adopt stochastic training, *i.e.*, uniformly sampling a network width $\boldsymbol{c}$ from the search space $\mathcal{C}_K$, and training its corresponding channel (filters) $\mathcal{N}(W, \boldsymbol{c})$ using training data $\mathcal{D}_{tr}$ afterwards. Note that a single $\boldsymbol{c}$ has two paths in BCNet, during training, a training batch $\mathcal{B} \subset \mathcal{D}_{tr}$ is supposed to forward simultaneously through both $\mathcal{N}^*_l(W)$ and $\mathcal{N}^*_r(W)$. Then the training loss is the averaged loss of both paths, *i.e.*, for each batch $\mathcal{B}$

$$\mathcal{L}_{tr}(W, \boldsymbol{c}; \mathcal{B}) = \frac{1}{2} \cdot \left(\mathcal{L}_{tr}(\mathcal{N}_l; \boldsymbol{c}, \mathcal{B}) + \mathcal{L}_{tr}(\mathcal{N}_r; \boldsymbol{c}, \mathcal{B})\right). \tag{10}$$

Despite with our BCNet, channels are trained more evenly than other methods. However, it still can not ensure a rigorous fairness over channels. For example, if a layer has 3 channels and we sample 10 widths on this layer. Then results can come to that the first channel is sampled 4 times and the other two are sampled 3 times, respectively. The first channel thus still gets more training than the others, which ruins the training fairness.

To solve this issue, we propose to leverage a complementary training strategy, *i.e.*, after sampling a network width $\boldsymbol{c}$, both $\boldsymbol{c}$ and its complementary $\bar{\boldsymbol{c}}$ get trained. For example, suppose a width $\boldsymbol{c} = (3, 2, 4)$ with maximum 6 channels per layer, then its complementary amounts to $\bar{\boldsymbol{c}} = (3, 4, 2)$ as Figure 2. The training loss for the BCNet is thus

$$\mathcal{L}_{tr}(W; \mathcal{D}_{tr}, \mathcal{N}) = \mathop{\mathbb{E}}_{\boldsymbol{c} \in U(\mathcal{C})} [\![\mathcal{L}_{tr}(W, \boldsymbol{c}; \mathcal{D}_{tr}) + \mathcal{L}_{tr}(W, \bar{\boldsymbol{c}}; \mathcal{D}_{tr})]\!]. \tag{11}$$

In this way, when we sample a width $\boldsymbol{c}$, we can always ensure all channels are evenly trained, and expect a more fair comparison over all widths based on the trained BCNet. Note that this complementary strategy only works for our BCNet, and fails in the typical unilateral augmented (UA) principle [43, 25, 6], which even worsens the bias as shown in Figure 2(b).

4.3. Evolutionary Search with Prior Initial Population Sampling

After the BCNet $\mathcal{N}^*$ is trained with weights W^*, we can evaluate each width by examining its performance (*e.g.*, classification accuracy) on the validation dataset $\mathcal{D}_{val}$ as Eq.(4). Besides, similar to the training of BCNet, the performance of a width $\boldsymbol{c}$ is indicated by the averaged accuracy of its left and right paths. Moreover, to boost the searching performance, we leverage the multi-objective NSGA-II [4] algorithm for evolutionary search, and hard FLOPs constraint can be thus well integrated. In generally, evolutionary search is prone to the initial population before the sequential mutation and crossover process. In this way, we

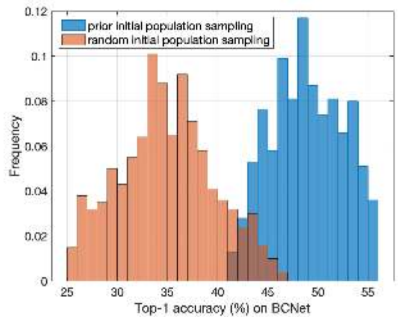

Figure 3. Histogram of Top-1 accuracy of searched widths on BCNet by evolutionary searching method with our prior or random initial population sampling w.r.t. ResNet50 (2G FLOPs) on ImageNet dataset.

propose a Prior Initial Population Sampling method to allocate a promising initial population, which is expected to contribute to the evolutionary searching performance.

Concretely, suppose the population size is P, and we hope the sampled initial population have high performance in order to generate competing generations during search. Note that during training of BCNet, we have also sampled various widths, whose quality can be reflected by the training loss. In this way, we can record the top m (*e.g.*, $m = 100$) widths $\{\boldsymbol{c}^{(i)}\}_{i=1}^m$ with smallest training loss $\{\ell^{(i)}\}_{i=1}^m$ as priors for good network widths. However, even the group size for every layer is set to 10, the search space of MobileNetV2 is as large as 10^{25}, which is too large to search good widths within limited training epochs. Thus we aim to learn layer-wise discrete sampling distributions $\mathcal{P}(l, c_i)$ to perform stochastic sampling a width $\boldsymbol{c} = (c_1, ., c_l, ., c_L)$, where $\mathcal{P}(l, c_i)$ indicates the probability of sampling width c_l at l-th layer subject to $\sum_i \mathcal{P}(l, c_i) = 1$.

Note that these m prior network widths actually can reflect the preference over some widths for each layer. For example, if at a layer l, a width c_l exists in these m prior widths with smaller training loss, then the sampling probability $\mathcal{P}(l, c_i)$ should be large as well. In this way, we can measure the *potential error* $\mathcal{E}(l, c_i)$ of sampling c_l width at l-th layer by recording the averaged training loss of all m widths going through it,*i.e.*,

$$\mathcal{E}(l, c_i) = \frac{1}{\sum_{j=1}^m \mathbf{1}\{\boldsymbol{c}_l^{(j)} = i\}} \cdot \sum_{j=1}^m \ell^{(j)} \cdot \mathbf{1}\{\boldsymbol{c}_l^{(j)} = i\}, \quad (12)$$

where $\mathbf{1}\{\cdot\}$ is the indicator function. Then the objective is to sample with minimum expected potential errors, *i.e.*,

$$\min_{\mathcal{P}} \sum_l \sum_i \mathcal{P}(l, c_i) \cdot \mathcal{E}(l, c_i), \text{ s.t. } \sum_i \mathcal{P}(l, c_i) = 1, \\ \mathcal{P}(l, c_i) \geq 0, \forall\, l = 1, ..., L. \quad (13)$$

In addition, we also need to deal with the hard FLOPs constraint in the initial population. Since the FLOPs of a layer depends on the channels of its input and output, we can limit the expected FLOPs of the sampled network width, *i.e.*,

$$\sum_l \sum_{(i,j)} \mathcal{P}(l, c_i) \cdot F(l, c_i, c_j) \cdot \mathcal{P}(l+1, c_j) \leq F_b, \quad (14)$$

where $F(l, c_i, c_j)$ is the FLOPs of l-th layer with c_i input channels and c_j output channels, which can be pre-calculated and stored in a looking-up table. Then Eq.(14) is integrated as an additional constraint for the problem Eq.(13). The overall problem is a QCQP (Quadratically constrained quadratic programming), which can be efficiently solved by many off-the-shelf solvers, such as CVXPY [5], GH [28]. As Figure 3 shows, our proposed sampling method can significantly boost evolutionary search by providing better initial populations. On average, the performance of our searched widths are much better than those obtained by random initial population, which proves the effectiveness of our proposed sampling method.

5. Experimental Results

In this section, we conduct extensive experiments on the ImageNet and CIFAR-10 datasets to validate the effectiveness of our algorithm. For all structures, we search on the reduced space $\mathcal{C}_K$ with default $K = 20$. Note that most pruning methods do not report their results by incorporating the knowledge distillation (KD) [17, 7] improvement in retraining except for MobileNetV2. Thus in our method, except for MobileNetV2, we also do not include KD in final retraining. Detailed experimental settings are elaborated in supplementary materials.

Comparison methods. We include multiple competing pruning, network width search methods and NAS models for comparison, such as DMCP [9], TAS [6], AutoSlim [43], MetaPruning [25], AMC [14], DCP [44], LEGR [3], CP [16], AutoPruner [27], SSS [21], EfficientNet-B0 [34] and ProxylessNAS [1]. Moreover, we also consider two vanilla baselines. *Uniform*: we shrink the width of each layer with a fixed factor to meet FLOPs budget. *Random*: we randomly sample 20 networks under FLOPs constraint, and train them by 50 epochs, then we continue training the one with the highest performance and report its final result.

5.1. Results on ImageNet Dataset

ImageNet dataset contains 1.28M training images and 50K validation images from 1K classes. In specific, we report the accuracy on the validation dataset as [24, 43], and the original model takes as the supernet while for the 1.0$\times$ FLOPs of all models, the supernet refers to a 1.5$\times$ FLOPs of original model by uniform width scaling. To verify the performance on both heavy and light models, we search on the ResNet50 and MobileNetV2 with different FLOPs

Table 1. Performance comparison of ResNet50 and MobileNetV2 on ImageNet. Methods with "*" denotes that the results are reported with knowledge distillation.

ResNet50					
FLOPs level	Methods	FLOPs	Parameters	Top-1	Top-5
3G	AutoSlim* [43]	3.0G	23.1M	76.0%	-
	MetaPruning [25]	3.0G	-	76.2%	-
	LEGR [3]	3.0G	-	76.2%	-
	Uniform	3.0G	19.1M	75.9%	93.0%
	Random	3.0G	-	75.2%	92.5%
	BCNet	3.0G	22.6M	**77.3%**	93.7%
2G	SSS [21]	2.8G	-	74.2%	91.9%
	GBN [42]	2.4G	31.83M	76.2%	92.8%
	SFP [13]	2.4G	-	74.6%	92.1%
	LEGR [3]	2.4G	-	75.7%	92.7%
	FPGM [15]	2.4G	-	75.6%	92.6%
	TAS* [6]	2.3G	-	76.2%	93.1%
	DMCP [9]	2.2G	-	76.2%	-
	MetaPruning [25]	2.0G	-	75.4%	-
	AutoSlim* [43]	2.0G	20.6M	75.6%	-
	Uniform	2.0G	13.3M	75.1%	92.7%
	Random	2.0G	-	74.6%	92.2%
	BCNet	2.0G	18.4M	**76.9%**	93.3%
1G	AutoPruner [27]	1.4G	-	73.1%	91.3%
	MetaPruning [25]	1.0G	-	73.4%	-
	AutoSlim* [43]	1.0G	13.3M	74.0%	-
	Uniform	1.0G	6.9M	73.1%	91.8%
	Random	1.0G	-	72.2%	91.4%
	BCNet	1.0G	12M	**75.2%**	92.6%
	AutoSlim* [43]	570M	7.4M	72.2%	-
	Uniform	570M	6.9M	71.6%	90.6%
	Random	570M	-	69.4%	90.3%
	BCNet	570M	12.0M	**73.2%**	91.1%

MobileNetV2					
FLOPs level	Methods	FLOPs	Parameters	Top-1	Top-5
305M (1.5×)	AutoSlim* [43]	305M	5.7M	74.2%	-
	Uniform	305M	3.6M	72.7%	90.7%
	Random	305M	-	71.8%	90.2%
	BCNet	305M	4.8M	**73.9%**	91.5%
	BCNet*	305M	4.8M	**74.7%**	92.2%
200M	Uniform	217M	2.7M	71.6%	89.9%
	Random	217M	-	71.1%	89.6%
	BCNet	217M	3.0M	**72.5%**	90.6%
	BCNet*	217M	3.0M	**73.5%**	91.3%
	MetaPruning [25]	217M	-	71.2%	-
	LEGR [3]	210M	-	71.4%	-
	AMC [14]	211M	2.3M	70.8%	-
	AutoSlim* [43]	207M	4.1M	73.0%	-
	Uniform	207M	2.7M	71.2%	89.6%
	Random	207M	-	70.5%	89.2%
	BCNet	207M	3.1M	**72.3%**	90.4%
	BCNet*	207M	3.1M	**73.4%**	91.2%
100M	MetaPruning [25]	105M	-	65.0%	-
	Uniform	105M	1.5M	65.1%	89.6%
	Random	105M	-	63.9%	89.2%
	BCNet	105M	2.3M	**68.0%**	89.1%
	BCNet*	105M	2.3M	**69.0%**	89.9%
	MuffNet [2]	50M	-	50.3%	-
	MetaPruning [25]	43M	-	58.3%	-
	Uniform	50M	0.9M	59.7%	82.0%
	Random	50M	-	57.4%	81.2%
	BCNet	50M	1.6M	**62.7%**	83.7%
	BCNet*	50M	1.6M	**63.8%**	84.6%

Table 2. Searching results of EfficientNet-B0 and ProxylessNAS on ImageNet dataset.

EfficientNet-B0				
Groups	Methods	Param	Top-1	Top-5
385M	Uniform	5.3M	76.88%	92.64%
	Random	5.1M	76.37%	92.25%
	BCNet	6.9M	**77.36%**	93.17%
192M	Uniform	2.7M	74.26%	92.24%
	Random	2.9M	73.82%	91.86%
	BCNet	3.8M	**74.92%**	92.06%
ProxylessNAS				
Groups	Methods	Param	Top-1	Top-5
320M	Uniform	4.1M	74.62%	91.78%
	Random	4.3M	74.16%	91.23%
	BCNet	5.4M	**75.07%**	91.97%
160M	Uniform	2.2M	71.16%	89.49%
	Random	2.5M	70.89%	89.12%
	BCNet	2.9M	**71.87%**	89.96%

budgets. In our experiment, the original ResNet50 (MobileNetV2) has 25.5M (3.5M) parameters and 4.1G (300M) FLOPs with 77.5% (72.6%) Top-1 accuracy, respectively.

As shown in Table 1, our BCNet achieves the highest accuracy on ResNet50 and MobileNetV2 w.r.t. different FLOPs, which indicates the superiority of our BCNet to other pruning methods. For example, our 3G FLOPs ResNet50 decreases only 0.2% Top-1 accuracy compared to the original model, which exceeds AutoSlim [43] and MetaPruning [25] by 1.3% and 1.1%. While for MobileNetV2, our 207M MobileNetV2 exceeds the state-of-the-art AutoSlim, MetaPruning by 0.4%, 1.1%, respectively. In addition, our BCNet even surpasses other algorithms more on tiny MobileNetV2 (105M) with 68% Top-1 accuracy and exceeds MetaPruning by 3.0%.

To further demonstrate the effectiveness of our BCNet on highly efficient models, we conduct searching on the NAS-based models EfficientNet-B0 and ProxylessNAS. The original EfficientNet-B0 (ProxylessNAS) has 5.3M (4.1M) parameters and 385M (320M) FLOPs with 76.88% (74.62%) Top-1 accuracy, respectively. As shown in Table 2, although the increase of performance is not as signicant as in Table 1, our method can still boost the NAS-based models by more than 0.4% on Top-1 accuracy.

5.2. Results on CIFAR-10 Dataset

We also examine the performance of MobileNetV2 and VGGNet on the moderate CIFAR-10 dataset, which has 50K training and 10K testing images with size 32×32 of 10 categories. Our original VGGNet (MobileNetV2) has 20M (2.2M) parameters and 399M (297M) FLOPs with accuracy of 93.99% (94.81%).

As shown in Table 3, our BCNet still enjoys great advantages in various FLOPs levels. For instance, our 200M MobileNetV2 can achieve 95.44% accuracy, which even outperforms the original model by 0.63%. Moreover, even with super tiny size (28M), our BCNet can still have 94.02% accuracy, which surpasses the state-of-the-art AutoSlim[43] by 2.0%. As for VGGNet, our BCNet is capable of outperforming those competing channel pruning methods DCP [44] and Slimming [24] by 0.20% and 0.56% with 2× acceleration rate.

5.3. Ablation Studies

Effect of BCNet as a supernet. To validate the effectiveness of our proposed supernet BCNet, we search the ResNet50, MobileNetV2, EfficientNet-B0 and Proxyless-

Table 3. Performance comparison of MobileNetV2 and VGGNet on CIFAR-10.

Groups	Methods	FLOPs	Params	accuracy
MobileNetV2				
200M	DCP [44]	218M	-	94.69%
	Uniform	200M	1.5M	94.57%
	Random	200M	-	94.20%
	BCNet	200M	1.5M	**95.44%**
146M	MuffNet [2]	175M	-	94.71%
	Uniform	146M	1.1M	94.32%
	Random	146M	-	93.85%
	BCNet	146M	1.2M	**95.42%**
44M	AutoSlim [43]	88M	1.5M	93.20%
	AutoSlim [43]	59M	0.7M	93.00%
	MuffNet [2]	45M	-	93.12%
	Uniform	44M	0.3M	92.88%
	Random	44M	-	92.31%
	BCNet	44M	0.4M	**94.42%**
28M	AutoSlim [43]	28M	0.3M	92.00%
	Uniform	28M	0.2M	92.37%
	Random	28M	-	91.69%
	BCNet	28M	0.2M	**94.02%**
VGGNet				
Groups	Methods	FLOPs	Params	accuracy
200M	Sliming [24]	199M	10.4M	93.80%
	DCP [44]	199M	10.4M	94.16%
	Uniform	199M	10.0M	93.45%
	Random	199M	-	93.02%
100M+	**BCNet**	197M	3.1M	**94.36%**
	Uniform	185M	9.3M	93.30%
	Random	185M	-	92.71%
	BCNet	185M	6.7M	**94.14%**
	CP [16]	156M	7.7M	93.67%
	Multi-loss [18]	140M	5.5M	94.05%
	Uniform	138M	6.8M	93.14%
	Random	138M	-	92.17%
	BCNet	138M	3.3M	**94.09%**
77M	CGNets [19]	91.8M	-	92.88%
	Uniform	77.0M	3.9M	92.38%
	Random	77.0M	-	91.72%
	BCNet	77.0M	1.2M	**93.53%**
	CGNet [19]	61.4M	-	92.41%

NAS on ImageNet dataset with 2× acceleration. Our default baseline supernet is that adopted by AutoSlim [43], which follows unilateral augmented principle to evaluate a network width. As the results in Table 4 shows, under the greedy search, only using our BCNet evaluation mechanism (second line) can enjoy a gain of 0.27% to 0.66% Top-1 accuracy. When searching with evolutionary algorithms, the gain still reaches at 0.28% to 0.35% Top-1 accuracy on various models. These exactly indicates using BCNet as supernet could boost the evaluation and searching performance. As for the complementary training strategy, we can see that it enables to boost our BCNet by improving the MobileNetV2 (ResNet50) from 69.92% (76.41%) to 70.04% (76.56%) on Top-1 accuracy. Note that greedy search without BCNet supernet amounts to AutoSlim, we can further indicate the superiority of our method to AutoSlim with achieved Top-1 accuracy 70.20% (76.90%) vs 69.52% (75.94%) on MobileNetV2 (ResNet50).

Effect of search space. We adopt the grouped search space $\mathcal{C}_K$ to reduce its complexity. To investigate the effect of search space, we searched VGGNet on CIFAR-10 dataset and MobileNetV2 on ImageNet dataset with various group size K. As in Figure 4(b), our method achieves the best performance in most cases around our default value $K = 20$. In addition, we noticed that when the group size K is small,

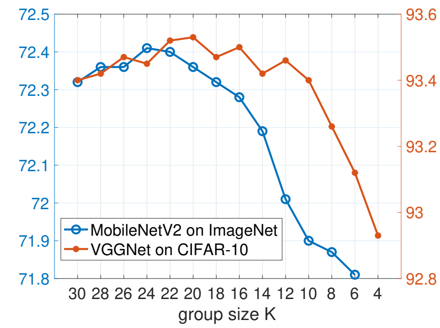

Figure 4. Accuracy performance of the searched network with different group size K of the search space.

the performance of searched network will increase with K growing larger. This is because group size K determines the size of search space $\mathcal{C}_K$, and larger K induces larger search space. In this way, the obtained network width will be closer to the Oracle optimal width, with higher accuracy achieved accordingly. In addition, the performance tends to be stable when the group size lies in $[14 : 22]$ but decreases afterwards, which implies the searching space might be too large for searching an optimal width.

5.4. Visualization and Interpretation of Results

For intuitively understanding, we visualize our searched networks with various FLOPs in Figure 5. Moreover, for clarity we show the retained ratio of layer widths compared to that of the original models. Note that for MobileNetV2, ResNet50, EfficientNet-B0 and ProxylessNAS with skipping or depthwise layers, we merge these layers which are required to have the same width. More visualization results can refer to the supplementary materials.

From Figure 5, we can see that on the whole, with decreasing FLOPs, layer width nearer the input tends to be reduced. However, the last layer is more likely to be retained. This might result from that the last layer is more sensitive to the classification performance, thus it is safely kept when the FLOPs is reduced. In the sequel, we will illustrate more elaborate observations w.r.t. each network, and present some intuitions accordingly.

ResNet50 on ImageNet. We found that when the network is pruned with a large FLOPs budget (*e.g.*, 3G or 2G), width of the first 1×1 convolutional layer (*e.g.*, 2nd and 5th layer in Figure 5) of each block in ResNet50 is preferentially reduced, which means 1×1 convolution may contribute less to classification performance. However, when FLOPs drops to a fairly small value (*e.g.*, 570M), channel number of 3×3 convolution (*e.g.*, 3rd and 6th layer in Figure 5) will decrease dramatically while that of 1×1 convolution increases instead. This implies that the network will be forced to use more 1×1 convolutions instead of 3×3 convolutions to extract information from feature maps. In addition, this observation also indicates that evolutionary algorithm is more effective than greedy search w.r.t. small

Table 4. Performance of searched MobileNetV2 (150M FLOPs), ResNet50 (2G FLOPs), EfficientNet-B0 and ProxylessNAS on ImageNet dataset with different supernet and searching methods.

evaluator		searching			models							
BCNet supernet	complementary training	greedy search	evolutionary random	evolutionary prior	MobileNetV2 Top-1	MobileNetV2 Top-5	ResNet50 Top-1	ResNet50 Top-5	EfficientNet-B0 Top-1	EfficientNet-B0 Top-5	ProxylessNAS Top-1	ProxylessNAS Top-5
		✓			69.52%	88.91%	75.64%	92.90%	74.02%	91.58%	70.97%	89.43%
✓		✓			69.87%	88.99%	76.30%	93.16%	74.39%	91.66%	71.24%	89.57%
✓	✓	✓			**69.91%**	89.02%	**76.42%**	93.19%	**74.51%**	91.78%	**71.33%**	89.62%
			✓		69.64%	88.85%	76.12%	92.95%	74.35%	91.54%	71.13%	89.49%
✓			✓		69.92%	88.91%	76.41%	93.12%	74.63%	91.93%	71.48%	89.69%
✓	✓		✓		70.04%	89.02%	76.56%	93.21%	74.73%	91.85%	71.62%	89.73%
✓	✓			✓	**70.20%**	89.10%	**76.90%**	93.30%	**74.92%**	92.06%	**71.87%**	89.96%

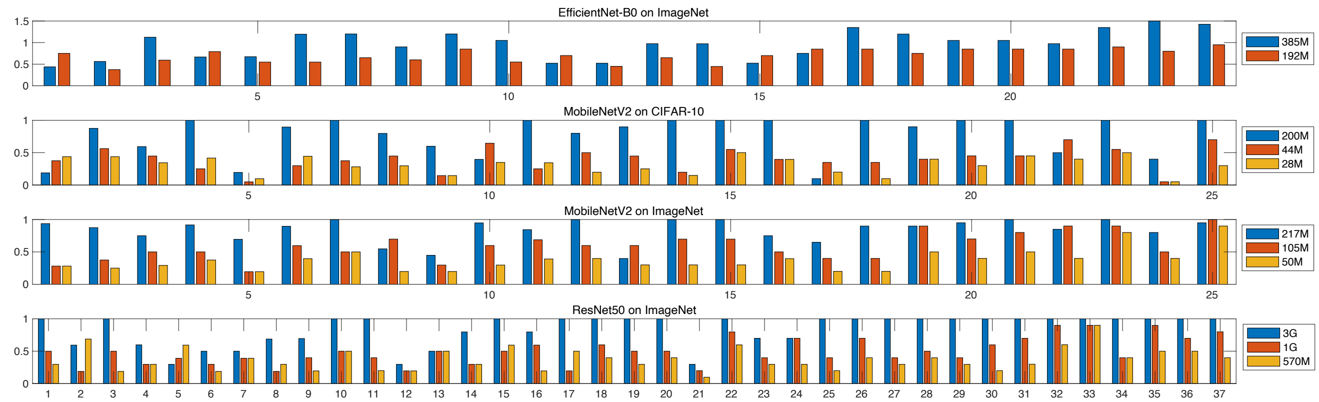

Figure 5. Visualization of searched networks w.r.t. different FLOPs. The vertical axis means the ratio of retained channel number compared to that of original networks at each layer.

FLOPs since evolutionary algorithm can always maintain the original search space. Nevertheless, greedy algorithm will greedily prune out more 1×1 convolutions at the beginning, which can not be recovered for small FLOPs budget.

MobileNetV2 on ImageNet and CIFAR-10. Different from ResNet50, widths of MobileNetV2 decrease more evenly with the reduction of FLOPs. This may be due to the limitation of depthwise convolutions, which requires the output channel number of first 1×1 convolution and the second 3×3 convolution to be the same in MobileNetV2 blocks. Compared from pruning on ImageNet, widths closer to the input layer are more easily to be clipped on CIFAR-10 dataset. This may be because the input of CIFAR-10 is 32×32, which do not need as many widths as ImageNet in the last layer. In addition, when FLOPs is reduced to a fairly small value (*e.g.*, 28M, 44M, and 50M for MobileNetV2), unlike pruning on ImageNet, the width of the last layer of MobileNetV2 on CIFAR-10 decreases rapidly. The reason for this phenomenon may be that MobileNetV2 on ImageNet is forced to classify 1000 categories, while it only needs to deal with 10-way classification on CIFAR-10. Then the width of the last layer on ImageNet tends to be retained, but gets decreased rapidly on CIFAR-10. More visualizations about MobileNetV2 are analyzed in the supplementary material.

EfficientNet-B0 on ImageNet. EfficientNet-B0 shares similar block structure with MobileNetV2. However, the width of EfficientNet-B0 varies more evenly than MobileNetV2, which may be due to its width setting is more better since it is determined by NAS. In detail, compared to the original setting of EfficientNet-B0, for the searched $1\times$ FLOPs network, the channels of adjacent blocks show opposite fluctuations (*e.g.*, channels of 1,3,5 blocks increase while channels in 2,4,6 blocks decrease). This may mean that the fluctuations of widths are conducive to the performance of searched network structure.

6. Conclusion

In this paper, we introduce a new supernet called BCNet to address the training unfairness and corresponding evaluation bias for searching optimal network width. In our BCNet, each channel is fairly trained and responsible for the same amount of widths. Besides, we leverage a stochastic complementary strategy for the training of BCNet, and propose a prior initial population sampling method to boost the evolutionary search. Extensive experiments have been implemented on benchmark ImageNet and CIFAR-10 datasets to show the superiority of our proposed method to other state-of-the-art channel pruning/network width search methods.

Acknowledgments

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

References

[1] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[2] Hesen Chen, Ming Lin, Xiuyu Sun, Qian Qi, Hao Li, and Rong Jin. Muffnet: Multi-layer feature federation for mobile deep learning. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[3] Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. Legr: Filter pruning via learned global ranking. *arXiv preprint arXiv:1904.12368*, 2019.

[4] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[5] Steven Diamond and Stephen Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *The Journal of Machine Learning Research*, 17(1):2909–2913, 2016.

[6] Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. In *Advances in Neural Information Processing Systems*, pages 759–770, 2019.

[7] Shangchen Du, Shan You, Xiaojie Li, Jianlong Wu, Fei Wang, Chen Qian, and Changshui Zhang. Agree to disagree: Adaptive ensemble knowledge distillation in gradient space. *Advances in Neural Information Processing Systems*, 33, 2020.

[8] Jianyuan Guo, Kai Han, Yunhe Wang, Chao Zhang, Zhaohui Yang, Han Wu, Xinghao Chen, and Chang Xu. Hit-detector: Hierarchical trinity architecture search for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11405–11414, 2020.

[9] Shaopeng Guo, Yujie Wang, Quanquan Li, and Junjie Yan. Dmcp: Differentiable markov channel pruning for neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1539–1547, 2020.

[10] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1580–1589, 2020.

[11] Kai Han, Yunhe Wang, Qiulin Zhang, Wei Zhang, Chunjing Xu, and Tong Zhang. Model rubik's cube: Twisting resolution, depth and width for tinynets. *arXiv preprint arXiv:2010.14819*, 2020.

[12] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[13] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018.

[14] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[15] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2019.

[16] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[18] Yiming Hu, Siyang Sun, Jianquan Li, Jiagang Zhu, Xingang Wang, and Qingyi Gu. Multi-loss-aware channel pruning of deep networks. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 889–893. IEEE, 2019.

[19] Weizhe Hua, Yuan Zhou, Christopher M De Sa, Zhiru Zhang, and G Edward Suh. Channel gating neural networks. In *Advances in Neural Information Processing Systems*, pages 1884–1894, 2019.

[20] Tao Huang, Shan You, Yibo Yang, Zhuozhuo Tu, Fei Wang, Chen Qian, and Changshui Zhang. Explicitly learning topology for differentiable neural architecture search. *arXiv preprint arXiv:2011.09300*, 2020.

[21] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 304–320, 2018.

[22] Shumin Kong, Tianyu Guo, Shan You, and Chang Xu. Learning student networks with few data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4469–4476, 2020.

[23] Tsungyi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. pages 936–944, 2017.

[24] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017.

[25] Zechun Liu, Haoyuan Mu, Xiangyu Zhang, Zichao Guo, Xin Yang, Kwang-Ting Cheng, and Jian Sun. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3296–3305, 2019.

[26] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[27] Jian-Hao Luo and Jianxin Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *arXiv preprint arXiv:1805.08941*, 2018.

[28] Jaehyun Park and Stephen Boyd. General heuristics for nonconvex quadratically constrained quadratic programming. *arXiv preprint arXiv:1703.07870*, 2017.

[29] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: towards real-time object detection with region proposal networks. 2015:91–99, 2015.

[30] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[31] Xiu Su, Shan You, Tao Huang, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. Locally free weight sharing for network width search. *arXiv preprint arXiv:2102.05258*, 2021.

[32] Xiu Su, Shan You, Tao Huang, Hongyan Xu, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. Data agnostic filter gating for efficient deep networks. *arXiv preprint arXiv:2010.15041*, 2020.

[33] Vivienne Sze, Yu Hsin Chen, Tien Ju Yang, and Joel S. Emer. Efficient processing of deep neural networks. *Synthesis Lectures on Computer Architecture*, 15(2):1–341, 2020.

[34] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. pages 6105–6114, 2019.

[35] Yehui Tang, Yunhe Wang, Yixing Xu, Dacheng Tao, Chunjing Xu, Chao Xu, and Chang Xu. Scop: Scientific control for reliable neural network pruning. *arXiv preprint arXiv:2010.10732*, 2020.

[36] Yehui Tang, Shan You, Chang Xu, Jin Han, Chen Qian, Boxin Shi, Chao Xu, and Changshui Zhang. Reborn filters: Pruning convolutional neural networks with limited data. In *AAAI*, pages 5972–5980, 2020.

[37] Yehui Tang, Shan You, Chang Xu, Boxin Shi, and Chao Xu. Bringing giant neural networks down to earth with unlabeled data. *arXiv preprint arXiv:1907.06065*, 2019.

[38] Yibo Yang, Hongyang Li, Shan You, Fei Wang, Chen Qian, and Zhouchen Lin. Ista-nas: Efficient and consistent neural architecture search by sparse coding. *arXiv preprint arXiv:2010.06176*, 2020.

[39] Yibo Yang, Shan You, Hongyang Li, Fei Wang, Chen Qian, and Zhouchen Lin. Towards improving the consistency, efficiency, and flexibility of differentiable neural architecture search. *arXiv preprint arXiv:2101.11342*, 2021.

[40] Shan You, Tao Huang, Mingmin Yang, Fei Wang, Chen Qian, and Changshui Zhang. Greedynas: Towards fast one-shot nas with greedy supernet. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1999–2008, 2020.

[41] Shan You, Chang Xu, Chao Xu, and Dacheng Tao. Learning from multiple teacher networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1285–1294, 2017.

[42] Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 2130–2141, 2019.

[43] Jiahui Yu and Thomas Huang. Autoslim: Towards one-shot architecture search for channel numbers. *arXiv preprint arXiv:1903.11728*, 8, 2019.

[44] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems*, pages 875–886, 2018.