Multi-Granularity Pruning for Model Acceleration on Mobile Devices

Tianli Zhao[1,2,3,4], Xi Sheryl Zhang[2,3], Wentao Zhu[5], Jiaxing Wang[6] Sen Yang[7], Ji Liu[8], and Jian Cheng[2,3⋆]

[1] School of Artificial Intelligence, University of Chinese Academy of Sciences
[2] Institute of Automation, Chinese Academy of Sciences
[3] AIRIA, [4] Maicro.ai, [5] Amazon Video Prime,
[6] JD.com, [7] Snap Inc., [8] Kwai Inc.

Abstract. For practical deep neural network design on mobile devices, it is essential to consider the constraints incurred by the computational resources and the inference latency in various applications. Among deep network acceleration approaches, pruning is a widely adopted practice to balance the computational resource consumption and the accuracy, where unimportant connections can be removed either channel-wisely or randomly with a minimal impact on model accuracy. The coarse-grained channel pruning instantly results in a significant latency reduction, while the fine-grained weight pruning is more flexible to retain accuracy. In this paper, we present a unified framework for the Joint Channel pruning and Weight pruning, named JCW, which achieves a better pruning proportion between channel and weight pruning. To fully optimize the trade-off between latency and accuracy, we further develop a tailored multi-objective evolutionary algorithm in the JCW framework, which enables one single round search to obtain the accurate candidate architectures for various deployment requirements. Extensive experiments demonstrate that the JCW achieves a better trade-off between the latency and accuracy against previous state-of-the-art pruning methods on the ImageNet classification dataset.

1 Introduction

Recently, deep learning has prevailed in many machine learning tasks. However, the substantial computational overhead limits its applications on resource-constrained platforms, e.g., mobile devices. To design a deep network deployable to the aforementioned platforms, it is necessary to consider the constraint incurred by the available computational resource and reduce the inference latency while maximizing the accuracy.

Pruning has been one of the predominant approaches to accelerating large deep neural networks. The pruning methods can be roughly divided into two categories, channel pruning which removes parameters in a channel-wise manner [20,60,23], and weight pruning which prunes parameters randomly [17,38].

⋆ Corresponding author.

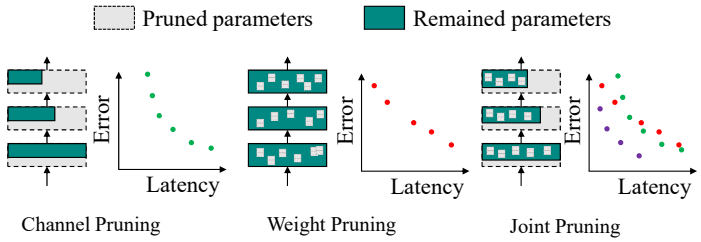

Fig. 1: Illustration of JCW (right), channel pruning (left) and weight pruning (middle). The JCW conducts joint channel and weight pruning, which achieves a better Pareto-frontier between the accuracy and latency with one single search.

The two mainstream pruning methods mainly focus on accelerating neural networks on one single dimension (e.g. either channel wisely or element wisely), while they may have different impacts on the latency and accuracy. For instance, the results in Table 1 show that the channel pruning method [15] offers better accuracy than the weight pruning method [10] when the inference latency is high. In contrast, the weight pruning [10] yields better accuracy than the channel pruning [15] under a low latency requirement. Inspired by this observation, this work attempts to unveil an important problem overlooked before, *is it possible to achieve a better latency-accuracy trade-off by designing a subtle network pruning method that enjoys benefits from both of the two pruning methods?*

However, it is non-trivial to determine the channel numbers and layer-wise weight sparsity in the joint channel and weight pruning because of the exponentially increasing decision space. Besides, simply applying channel width and weight sparsity search sequentially often leads to a sub-optimal solution because the two search steps are in fact entangled and it is difficult to determine a better balance between channel and weight pruning for a wide range of latency constraints.

To this end, we build an efficient model acceleration paradigm, named as JCW (**J**oint **C**hannel and **W**eight pruning) which accelerates the models by applying both channel and weight pruning jointly and finds the a better balance between channel and weight pruning automatically. Considering both the latency and accuracy, we formulate the model acceleration into a multi-objective optimization (MOO) problem. Specifically, given a predefined base model with L layers, denoting the number of channels of a certain compressed model for channel pruning by $C = \{c^{(l)}\}_{l=1}^{L}$, and weight sparsity for weight pruning by $S = \{s^{(l)}\}_{l=1}^{L}$, we search for a sequence of (C, S) lying on the Pareto-frontier between latency and accuracy:

$$(C, S)^* = \arg\min_{C,S} \{\mathcal{T}(C, S), \mathcal{E}(C, S)\}, \tag{1}$$

where $\mathcal{T}(C, S)$ and $\mathcal{E}(C, S)$ denote the latency and error rate of the compressed model, respectively. We further propose a uniform non-dominated sorting selection based on an enhanced evolutionary algorithm, NSGA-II [4], to generate

Method	Type	Latency	Accuracy
Fast	Weight Pruning	88.94 ms	72.0%
DMCP	Channel Pruning	82.95 ms	72.4%
Fast	Weight Pruning	35.89 ms	65.2%
DMCP	Channel Pruning	33.50 ms	62.7%

Table 1: Latency & accuracy of MobileNetV2 models accelerated by fast sparse convolution [10] and DMCP [15]. The best acceleration strategy differs under various latency budgets.

accurate candidate architectures with a wide range of latency in one single round search. To alleviate the search cost, we construct an accuracy predictor based on parameter sharing [42,16], and a latency predictor based on tri-linear interpolation.

We conduct extensive experiments to validate the effectiveness of JCW on the ImageNet dataset [5]. The JCW outperforms previous state-of-the-art model compression approaches by a large margin. Without loss of accuracy, the JCW yields 2.74×, 2.42× and 1.93× speedup over ResNet18 [19], MobileNetV1 [24] and MobileNetV2 [44], respectively.

Our major contributions are summarized as follows,

- We build a general model acceleration framework by the joint channel and weight pruning, JCW, as shown in the right of Fig. 1, which finds a better balance between channel and weight pruning automatically and obtains a much better trade-off between model accuracy and latency on mobile devices. To our best knowledge, we are the first to investigate on achieving the balance between channel and weight pruning automatically for better model acceleration.
- We enhance the Pareto multi-objective optimization with a uniform non-dominated sorting selection. The enhanced search algorithm can find multiple accurate candidate architectures for various computational budgets through one single round search.
- Extensive experiments demonstrate the effectiveness of the joint channel pruning and weight pruning. The JCW outperforms previous state-of-the-art model compression and acceleration approaches by a large margin on the ImageNet classification dataset.

2 Methodology

2.1 Motivation

In this section, we review the two categories of existing network pruning methods – coarse-grained channel pruning and fine-grained weight pruning, and analyze their pros and cons, which give rise to our joint design method.

Channel pruning reduces the width of feature maps by pruning filters channel-wisely [15,48]. As a result, the original network is shrunk into a thinner one. The

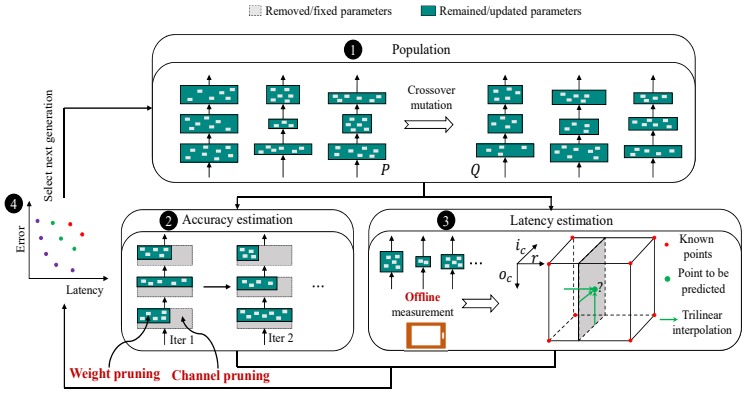

Fig. 2: Illustration of the framework of JCW.

channel pruning is well-structured and thus conventionally believed to be more convenient for model acceleration than random weight pruning [53,20]. However, a well-known drawback of channel pruning is its difficulty in attaining accuracy because of its strong structural constraints.

In random weight pruning, each individual element of parameters can be freely pruned [18,17]. It is more flexible and generally known to be able to get theoretically smaller models than channel pruning. More recently, Elsken *et al.* [10] made it more practical by arguing that despite of its irregular memory access, it can also be efficiently accelerated on mobile CPUs if implemented properly. However, the acceleration ratio achieved by pure weight pruning is still limited because the problem of irregular memory access still exists. For example, their method can only accelerate the computation by $\sim 3\times$ even when 90% of parameters are removed.

Based on our analysis, we present JCW, a unified framework that combines the advantages of both channel pruning and weight pruning for better model acceleration by applying the two jointly. JCW searches the number of channels and layer-wise weight sparsity jointly and automatically finds the balance between channel and weight pruning for a wide range of latency budgets, thus it is able to achieve a much better accuracy-latency trade-off.

2.2 Problem formulation

Formally, for some compressed model $\mathcal{A}$ with L layers, we denote the number of channels of each layer by $C_{\mathcal{A}} = \{c_{\mathcal{A}}^{(l)}\}_{l=1}^{L}$, and the weight sparsity[1] of each layer by $S_{\mathcal{A}} = \{s_{\mathcal{A}}^{(l)}\}_{l=1}^{L}$. In this way, each sub-network $\mathcal{A}$ can be represented by a pair of vectors: $\mathcal{A} = \{C_{\mathcal{A}}, S_{\mathcal{A}}\}$. Our goal is to accelerate the inference of networks by

[1] We use weight sparsity to denote the ratio of non-zero parameters of remaining channels across the whole paper.

applying channel pruning and weight pruning simultaneously, while at the same time minimizing the accuracy loss:

$$\mathcal{A}^* = \arg\min_{\mathcal{A}} \{\mathcal{T}(C_{\mathcal{A}}, S_{\mathcal{A}}), \mathcal{E}(C_{\mathcal{A}}, S_{\mathcal{A}})\}, \tag{2}$$

where $\mathcal{T}(C_{\mathcal{A}}, S_{\mathcal{A}})$ and $\mathcal{E}(C_{\mathcal{A}}, S_{\mathcal{A}})$ denote the inference latency and task specific error of the model, respectively. For simplicity, we will abbreviate them as $\mathcal{T}(\mathcal{A})$ and $\mathcal{E}(\mathcal{A})$ in the remaining of this paper under the clear context.

A crucial question for solving the problem in Eq. (2) is: *How to determine the number of channels and weight sparsity for each layer?* One alternative way is to first prune the channels of the original model in an automated way, then prune the weights of the channel-pruned model for further acceleration. However, this separated optimization may lead to a sub-optimal solution, because it is difficult to find the optimal balance between channel pruning and weight pruning by hand. Concretely speaking, the optimal architecture for channel pruning may be sub-optimal when further applying weight pruning for acceleration. Therefore, we instead optimize the number of channels and weight sparsity simultaneously in one single optimization run and determine the balance of acceleration between channel pruning and weight pruning automatically.

Another difficulty in solving the problem in Eq. (2) is that there are more than one objective (the latency and accuracy) to be optimized, yielding a multi-objective optimization (MOO) problem naturally. It is challenging to design one model that achieves the best values for both of the two objectives since these two objectives are generally conflict with each other. Therefore, the optimal solutions for the problem in Eq. (2) are not unique, and we need to find a sequence of models lying on the Pareto-frontier between the accuracy and latency[2]. We solve this problem based on the multi-objective evolutionary algorithm.

2.3 Unified framework

Before going into the details, we first introduce the overview of the whole framework, which is illustrated in Fig. 2. The JCW works in an iterative way, it maintains a sequence of well-performing models $P = \{(C_i, S_i)\}_{i=1}^n$ with different number of channels and weight sparsity, here n is the population size. ❶ In each iteration, a new set of models $Q = \{(C_i^{new}, S_i^{new})\}_{i=1}^n$ are generated from P through crossover and mutation operators. Then, we estimate the accuracy (❷) and latency (❸) of all the models in $P \cup Q$. ❹ Based on the estimations, we select the models with various latency and relatively low error rates to form the next generation of P. The proposed components are integrated under the learning problem of Eq. (2), and the iteration continues until the qualified models are found. In the sequel, we instantiate different components of the framework, i.e. the accuracy estimator in Section 2.4, the latency estimator in Section 2.5, and the uniform non-dominated sorting selection of models in Section 2.6.

[2] In MOO, the Pareto-frontier is a set of solutions that for each solution, it is not possible to further improve some objectives without degrading other objectives.

2.4 Accuracy estimation

In JCW, the accuracy prediction is done efficiently by training a super-net with parameter sharing [16]. Parameter sharing has been widely used in previous one shot NAS methdos [16,3,59,45]. However, the accuracy predictor in JCW is different from theirs in that we support not only coarse-grained channel pruning but also fine-grained weight pruning. More importantly, these methods often train a super-net containing all the models in the whole search space, the training target can be formulated by:

$$\min_{w} \mathbb{E}_{(x,y)\sim\mathcal{D},\mathcal{A}\sim\Omega}[\mathcal{L}(x|y;W_{\mathcal{A}})], \tag{3}$$

where Ω is the search space (*e.g.* in the scenario of joint channel and weight pruning in this paper, the search space is all the models with different channel numbers and layer-wise sparsity), $\mathcal{D}$ is the training dataset, W is the parameter of the super-net, and $W_{\mathcal{A}}$ are a part of parameters from W to form the model with architectural configuration $\mathcal{A}$. In this case, the large number of models in the super-net are largely coupled with each other [48].

Considering that in our framework, we only need to determine the accuracy rank of a finite number of architectures (*i.e.* 2 times of the population size) each time, at the beginning of each evolutionary generation, we reconstruct the super-net to contain only models to be evaluated and retrain it to determine the accuracy rank. Formally, denote $\tilde{P} = P \cup Q = \{(C_i, S_i)\}_{i=1}^{2n}$ to be the sequence of models whose accuracy rank are to be estimated, we train the super-net with the following simplified target:

$$\min_{W} \mathbb{E}_{(x,y)\sim\mathcal{D},\mathcal{A}\sim\tilde{P}}[\mathcal{L}(x|y;W_{\mathcal{A}})], \tag{4}$$

and $W_{\mathcal{A}}$ is constructed by selecting the parameters with top channel indices and then applying norm-based weight pruning. Note that the number of models in $\tilde{P}$ is far less than the whole search space Ω. In this way, the coupling between different models in the super-net can be reduced.

2.5 Latency estimation

Previously, latency prediction is conducted by constructing a look up Table [55,49,50] or training an estimation model [54,2]. The former one is limited to a small number of candidates, while the latter one requires a large number of architecture-latency pairs to train the estimation model, which is laborious to collect. In contrast, in the JCW, we propose to estimate the latency of models with trilinear interpolation. The latency of one model can be represented by the summation of the latency of each layer:

$$\mathcal{T}(C,S) = \sum_{l=1}^{L} \mathcal{T}^{(l)}(C,S), \tag{5}$$

where $\mathcal{T}^{(l)}(C,S)$ is the latency of the l-th layer of the model represented by $\{C,S\}$. In the context of joint channel pruning and weight pruning, the latency of each layer depends on the number of input/output channels and the weight sparsity of that layer:

$$\mathcal{T}^{(l)}(C,S) = \hat{\mathcal{T}}^{(l)}(c^{(l-1)}, c^{(l)}, s^{(l)}). \tag{6}$$

For efficient latency estimation, we build a layer-wise latency predictor $\hat{\mathcal{T}}^{(l)}(\cdot)$ with trilinear interpolation. This is based on the observation that the latency is locally linear with respect to the layer width and weight sparsity[3]. Specifically, we denote $C_{max} = \{c^{(l)}_{max}\}_{l=1}^{L}$ as the maximum number of channels of each layer. For layer l with maximum input channels of $c^{(l-1)}_{max}$ and maximum output channels of $c^{(l)}_{max}$, we first measure the real runtime on the target device with channel width $0, \frac{1}{N}, \frac{2}{N}, \cdots, 1.0$ of both input and output channels, and weight sparsity of $0, \frac{1}{M}, \frac{2}{M}, \cdots, 1.0$, respectively, generating a 3-D array of architecture-latency samples. We denote $T^{(l)}_{ijk}$ to be the latency of the l^{th} layer's convolution with input, output channels and weight sparsity of $\frac{i}{N}c^{(l-1)}_{max}$, $\frac{j}{N}c^{(l)}_{max}$, and $\frac{k}{M}$, respectively, and define:

$$c^{(l)}_i = N\frac{c^{(l)}}{c^{(l)}_{max}} - i, \qquad s^{(l)}_i = Ms^{(l)} - i \tag{7}$$

to be the normalized array index. Given any input/output channels and weight sparsity, we can easily approximate the latency through trilinear interpolation of the 3-D array [4]:

$$\hat{\mathcal{T}}^{(l)}(c^{(l-1)}, c^{(l)}, s^{(l)}) = \sum_{i,j,k} \tau(c^{(l-1)}_i)\tau(c^{(l)}_j)\tau(s^{(l)}_k)T^{(l)}_{ijk}, \tag{8}$$

where:

$\tau(x) = \max(0, 1 - |x|)$.

In practice, we find that $M = 10, N = 8$ is sufficient for approximating latency with high efficiency and accuracy as illustrated in Fig. 5. Compared to other latency estimation methods [55,2] requiring tens of thousands of architecture-latency pairs, the proposed trilinear interpolation-based latency predictor can be efficiently constructed with as less as 700 data points.

2.6 Uniform non-dominated sorting selection

An important role of individual selection is to search for well-performed models while keeping model diversity in terms of latency. However, this cannot be fulfilled by the standard selection scheme used in previous multi-objective evolutionary algorithms [4] because in JCW, the accuracy estimation with parameter

[3] Please refer to the Appendix for more results about this observation.

[4] More detailed derivation is given in Appendix.

sharing is not as accurate as latency estimation. As a result, the evolver will put too much emphasis on the latency minimization, and large models will be gradually and incorrectly removed during evolution as shown in the left of Fig. 7.

To handle this obstacle, we propose a uniform non-dominated sorting selection to generate diverse architectures of various latency. We first uniformly sample N points from the interval $[T_{\min}, T_{\max}]$:

$$T_i = T_{\min} + i \times \frac{T_{\max} - T_{\min}}{N-1}, i = 0, 1, 2, \cdots, N-1, \tag{9}$$

where $T_{\min}$, $T_{\max}$ are the minimal and maximal latency, respectively. For each T_i, we sort the individuals in the merged population $P \cup Q$ with objectives $\{|T_i - \mathcal{T}|, \mathcal{E}\}$ with non-dominated sort [4], where $\mathcal{T}$ and $\mathcal{E}$ are latency and error rate of the network architecture.

Let $F_i = \{F_i^{(s)}\}_{s=0}^{n_i-1}$ be frontier stages after sorting architectures with objectives $\{|T_i - \mathcal{T}|, \mathcal{E}\}$. We select candidates stage by stage in the order $F_0^{(0)}, F_1^{(0)}, \cdots$ until the number of selected candidates reaches the evolutionary population size. When we select individuals from F_i, architectures with latency close to T_i and relatively low error rate will be selected. In the right of Fig. 7, it demonstrates that our uniform non-dominated sorting selection generates diverse architectures of various latency.

3 Experimental Results

To validate the efficiency of our joint channel and weight pruning (JCW), we conduct extensive experiments, including ablation studies, based on ResNet18 [19], MobileNetV1 [24], and MobileNetV2 [44] on the ImageNet classification dataset [5].

3.1 Implementation details

We use a similar technique as fast sparse ConvNets [10] for efficient computation of sparse convolution, with two slight improvements. (i) For weight pruning, we group the parameters along the output channels and remove the parameters group-wisely. Specifically, four parameters at the same location of adjacent output channels are grouped together, and parameters in the same group are removed or retained simultaneously. The grouping strategy is beneficial for efficient data reuse [10]. In all of our experiments, the group size is set to 4. (ii) We extend their computation algorithm to support not only matrix multiplication but also regular convolution. We implement an efficient algorithm for the computation of sparse convolution and utilize it to measure the latency of our searched sparse models.

For evolutionary search, we set the population size to 64 and the number of search steps to 128. We sample a subset of the ImageNet dataset for the supernet training. Specifically, we randomly sample 100 classes from ImageNet, 500 images per class to construct the train set, and randomly sample 50 images from the rest images per class to construct the validation set. We train the

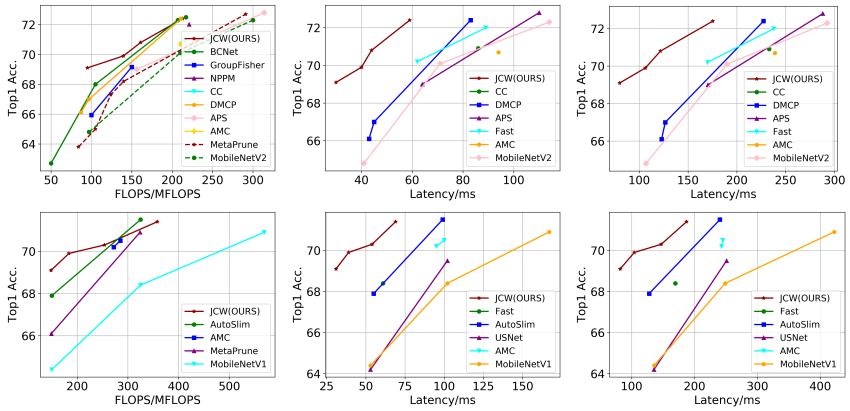

Fig. 3: Comparison of JCW with state of the art model compression/acceleration methods. Top: the results on MobileNetV2. Bottom: the results on MobileNetV1. Left: the comparison of FLOPS. Middle: the comparison of latency on one single Cortex-A72 CPU. Right: the comparison on one single Cortex-A53 CPU.

supernet for 30 epochs with batch size of 256, where the first 10 epochs are used for parameter sharing warming up. The learning rate and weight decay are set to 0.1 and 0.00004 in all the experiments, respectively. For each model in supernet, the batch normalization (BN) statistics are recalculated with $1,280$ images.

After we complete the search stage, the generated architectures are re-trained on the whole training set and validated on the validation set of the ImageNet dataset. We train the compressed models with ADMM, details about the hyper-parameters of different models are given in Appendix.

If not specified, the latency of all the models are measured on one single ARM Cortex-A72 CPU. The latency of models for channel pruning methods is measured with TFLite [14], which is a commonly used mobile-oriented deep learning inference framework for dense deep neural networks. The latency of models with weight sparsity is measured with our implemented high-performance sparse convolution algorithm. We run each model for 20 times and report the average of the runtime.

3.2 Comparison with state-of-the-art methods

We compare JCW with multiple state-of-the-art model compression and acceleration methods, including BCNet [45], NPPM [13], GroupFisher [34], CC [1], DMCP [15], APS [48], AMC [23], AutoSlim [57], USNet [58], and Fast [10] in terms of model FLOPS, accuracy, and latency on multiple types of devices. The main results are shown in Figure 3 and Table 2, and more detailed results are shown in Appendix. The middle of the column shows the comparison in terms of accuracy and latency on a single ARM Cortex-A72 CPU. We see

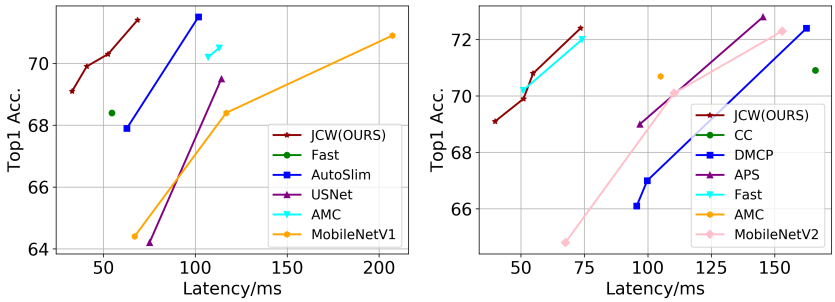

Fig. 4: Comparison in terms of accuracy and latency on 4× Cortex-A53 CPUs.

from the figure that JCW outperforms all the baseline methods by a large margin. This proves that JCW achieves a better latency-accuracy trade-off.

Method	Latency A72	Latency A53	Latency A53×4	Acc@1
Uniform 1×	537ms	1167ms	402ms	69.8%
DMCP	341ms	893ms	291ms	69.7%
APS	363ms	921ms	322ms	70.2%
JCW(OURS)	**194ms**	**518ms**	**158ms**	**69.7%**
	224ms	**694ms**	**179ms**	**70.2%**

Table 2: Experimental results on the Resnet18 model.

Generalization to other deployment platforms. The latency of models can be largely depended on deployment platforms. Thus a common problem about JCW may be: Whether it can generalize well to other deployment platforms? To resolve this, we directly deploy the searched models on Cortex-A53 CPUs. The latency-accuracy comparisons are shown in the right column of the Figure 3, we see from the figure that, although not targeting Cortex-A53 during the search, JCW still achieves excellent latency-accuracy trade-off on this platform, outperforming all the other methods by a large margin. In Figure 4, we further show the comparison of accuracy and latency on multi-device platforms. The latency is further measured on 4× Cortex-A53 CPUs. JCW still outperforms all the other baselines in most cases, except on MobileNetV2 under the latency constraint of 50 ms, where JCW performs very similar to Fast [10].

Generalization to other metrics. Besides latency, FLOPs is also an important evaluation index when deploying deep learning models to mobile devices. We thus further compare JCW with other methods in terms of FLOPs and accuracy in the left column of Figure 3. We see that although targeting latency, JCW still achieves competitive trade-off between FLOPs and accuracy. This is not surprising because JCW includes flexible weight pruning.

To sum up, the proposed JCW can achieve excellent trade-off between latency and accuracy, and generalizes well to other deployment platforms and metrics.

3.3 Ablation study

Accuracy of latency estimation. To evaluate the accuracy of the proposed latency estimation, we compare the real latency and predicted latency of 100

randomly generated MobileNetV1 models with various number of channels and weight sparsity. Specifically, we run each model for 20 times, calculate the average runtime and compare it with the predicted runtime. Results are shown in the left of Fig. 5. From the figure, we can observe that with trilinear interpolation, the predicted latency is highly correlated to the real latency of deep networks. The right of Figure 5 shows the latency and the number of arithmetic operations of different deep networks. We can see that the number of arithmetic operations (MFLOPS) is positively correlated with latency generally, while the latency does not monotonically increase with the number of operations (MFLOPS). This is mainly because that the model's latency on real hardware platform can be impacted by both computation intensity and other factors such as memory access time. This phenomenon motivates us to design deep networks based on latency instead of FLOPs for model pruning. When deploying a deep network into a practical hardware, we consider the latency of runtime, not the FLOPs.

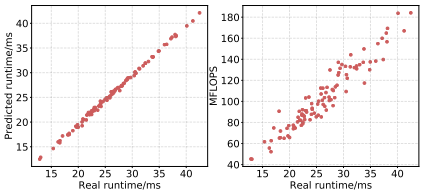

Fig. 5: Left: the real runtime & the predicted runtime with proposed trilinear interpolation. Right: the real runtime & FLOPs of models.

Correlation of accuracy predictor. In JCW, the accuracy predictor is designed to support both channel and weight pruning, which has not been studied before. Recall in Sec. 2.4 that in each training, the super-net will be re-initialized and only contains a finite number of architectures. We further evaluate the correlation between predicted accuracy and true accuracy. It is time consuming to train a number of models on the large-scale ImageNet dataset, so we evaluate it on the CIFAR-10 dataset. We train a super-net containing 128 ResNet20 models with various channel widths and weight sparsity, and all the hyper-parameters are the same as described in Sec. 3.1. Fig. 6 shows a high correlation between predicted and true accuracy.

Method	CP	WP	Optim.	Latency	Accuracy
WSO	✗	✓	-	161.72 ms	68.45%
				196.82 ms	69.54%
CWO	✓	✗	-	161.14 ms	66.78%
				205.03 ms	67.75%
SCW	✓	✓	Seq.	197.62 ms	69.29%
				221.65 ms	69.58%
JCW	✓	✓	**Joint**	**160.37 ms**	**69.16%**
				196.44 ms	**69.90%**
				223.73 ms	**70.19%**

Table 3: Comparisons among different variants of JCW for accelerating ResNet18 on ImageNet. JCW consistently outperforms all of its variants.

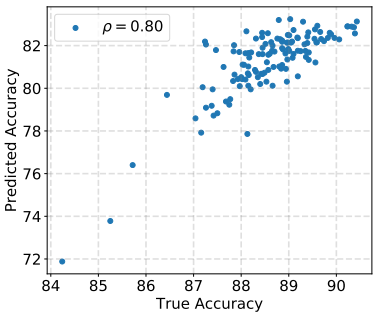

Fig. 6: Correlation between predicted accuracy and true accuracy of ResNet20 models on CIFAR-10 dataset.

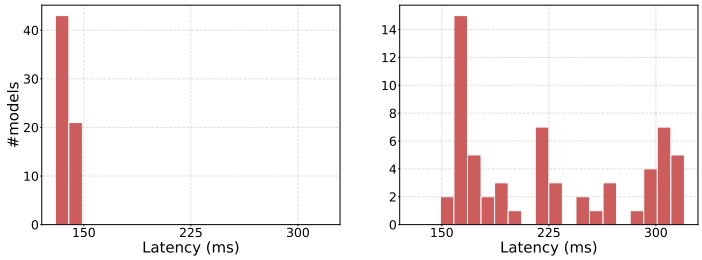

Fig. 7: Left: latency distribution after 50 evolutionary steps with standard non-dominated sorting selection. Right: latency distribution after 50 evolutionary steps with uniform non-dominated sorting selection. The searches are conducted with ResNet18 on ImageNet. The proposed selection generates models of various latency, while the standard selection gives priority to models of low latency.

Effect of joint channel and weight pruning.

The core idea of JCW is to apply joint channel and weight pruning jointly for better latency-accuracy trade-off. To prove its effectiveness, we compare JCW with three of its variants, *i.e.* (i) WSO which only searches for the layer-wise weight sparsity, while the number of channels for each layer remains the maximum value; (ii) CWO which only searches for the number of channels while keeps the full parameters of remaining channels; (iii) SCW which first searches for number of channels and then searches for weight sparsity with fixed channel widths.

Table 3 shows that our JCW outperforms all the other variants, and CWO performs the worst. In particular, under the latency of $\sim 160ms$, the accuracy of JCW is 0.61% and 2.38% higher than WSO and CWO, respectively. Under the latency of $\sim 196ms$, the JCW achieves 0.46% and 0.61% higher accuracy than WSO and SCW, respectively. Moreover, we see from the table that SCW, which simply applies channel width and weight sparsity search sequentially, does not achieve a better accuracy-latency trade-off than JCW. This is reasonable because simply applying channel and weight pruning search sequentially will lead to a sub-optimal balance between channel and weight pruning. In contrast, JCW optimizes channel width and weight sparsity jointly in one single search and tries to find the optimal balance between channel and weight pruning, thus achieving the best accuracy-latency trade-off.

Effect of uniform non-dominated sorting selection. Considering the search efficiency, we predict the accuracy of models with different architectures using parameter sharing at the cost of inaccurate accuracy prediction. Because of inaccurate model accuracy prediction, the evolver tends to focus on minimizing the other objective, the latency. For instance, let $\mathcal{A}$, $\mathcal{B}$ be two architectures in the combined population to be selected. We assume that $\mathcal{T}(\mathcal{A}) < \mathcal{T}(\mathcal{B})$, $\mathcal{E}(\mathcal{A}) > \mathcal{E}(\mathcal{B})$. Here, $\mathcal{T}$ and $\mathcal{E}$ are real latency and error rate of architectures, respectively. In terms of multi-objective optimization, there is no priority relation between $\mathcal{A}$ and $\mathcal{B}$. In other words, both architectures $\mathcal{A}$ and $\mathcal{B}$ should be

selected in the new population with an equal chance. If the accuracy estimation is inaccurate, it is likely that the predicted error rate of $\mathcal{A}$ is smaller than $\mathcal{B}$. In this case, the standard non-dominated sorting selection may remove the architecture $\mathcal{B}$ from the population incorrectly. This motivates us to develop the uniform non-dominated sort scheme, which explicitly selects architectures with a wide range of latency. The uniform non-dominated sorting selection generates diverse architectures of various latency.

To further validate the effectiveness of the proposed selection method, we show the latency distribution of models searched with the above two different selection methods in Fig. 7. The experiments are conducted with ResNet18 on the ImageNet dataset. From the left of Fig. 7, we can observe that the latency of models searched with the original selection scheme are small after 50 evolutionary steps. In contrast, from the right of Fig. 7 we can observe that with the proposed uniform non-dominated sorting selection, models with relatively large latency are also preserved during search. So we conclude that the enhanced evolutionary algorithm with the proposed uniform non-dominated sorting selection is able to generate diverse models with various latency.

4 Related Works

Pruning has long been one of the primary techniques for network compression and acceleration [18,17,32,20]. These methods remove unimportant parameters from the original network and optimize the remaining parts of the networks to retain accuracy. According to the granularity of pruning, these methods can be categorized into fine-grained weight pruning and coarse-grained filter pruning. In weight pruning, parameters are removed in weight-level [28,18,17,7,12]. Weight pruning is flexible to achieve theoretically smaller models and can also be efficiently accelerated on mobile CPUs thanks to the recent work of fast sparse convolution [10]. In contrast, channel pruning compresses networks by removing parameters at the filter level. Most of the early channel pruning methods are based on an filter importance scoring scheme, *e.g.*, the filter norm [31,21], the percentage of zero-activation [26], the reconstruction error of outputs [20,37,6], the increase of loss after pruning [39,40,41,34], the geometric properties of filters [22,27]. Besides, sparse regularization-based methods [53,43] have also been intensively explored. Channel pruning methods are well-structured, thus it can be directly accelerated without extra implementation efforts.

Apart from the aforementioned pruning methods, many recently emerging pruning methods formulate the network pruning as an architecture search, taking benefits from the automated process in composing architectures to avoid the labor-prohibitive model design. He *et al.* [23] propose to determine the number of channels with reinforcement learning and outperforms human-designed pruning methods. Lin *et al.* [33] search for the channel numbers with population-based algorithm. Wang *et al.* [52] model the problem of channel number search as structural redundancy reduction. Gao *et al.* [13], Wang *et al.* [51] train parameterized accuracy predictors to guide the pruning process. Liu *et al.* [35] train a

meta network to predict the weights of compressed models, then conduct evolutionary search for pruning. There is also a vast body of work utilizing the parameter sharing technique to train a supernet for accuracy evaluation and conduct the pruning with evolutionary search [16,3,45], greedy slimming [57], or reinforcement learning [48]. Besides, differentiable channel number search approaches [9,15] have also been investigated.

Besides the above static pruning methods, there is also a vast body of work pruning networks dynamically. These methods decide online inference which connections should be pruned according to the input image [29,47,30]. Dynamic pruning is a promising method to reduce the average computational complexity over a large number of testing samples, while it is not convenient for deployment on mobile devices for real-time applications because its computation complexity depends on the input image and is unpredictable. Our method lies in the family of static pruning.

Efficient model design often involves multiple objectives, *e.g.*, the accuracy, the latency, the model size, *etc.* In this perspective, it is more desired to search for a sequence of Pareto optimal models. Many works have been proposed to deal with multi-objective model design. Hsu *et al.* [25], Tan *et al.* [46] integrate multiple objectives into one correlated reward function and conduct the search with reinforcement learning. However, they need trial and error to design the form and related hyper-parameters for the correlated reward function, which is prohibitively laborious. Some recent works [8,36,11,56] search for Pareto optimal architectures directly with evolutionary algorithm and a selection criterion based on non-dominated sorting [4]. Our work focuses on model pruning, which is orthogonal to these general NAS methods.

To our best knowledge, the JCW is the first work investigating the essential part of pruning: is it possible to absorb both benefits of channel and weight pruning and achieve a better accuracy-latency trade-off by applying the two jointly? We conduct extensive experiments and ablation studies, which demonstrate that the joint channel and weight pruning achieves a better accuracy-latency Pareto frontier than previous pruning approaches.

5 Conclusion

In this work, we propose a joint channel and weight pruning, named JCW, which achieves a better pruning proportion between channel and weight pruning. We further construct a multi-objective optimization considering both model accuracy and inference latency in the JCW, which can be solved by a tailored Pareto-optimization evolutionary algorithm. Extensive experiments demonstrate that the effectiveness of each component of JCW.

Acknowledgements This work was supported in part by the National Key Research and Development Program of China under Grant 2021ZD0201504, in part by the Strategic Priority Research Program of Chinese Academy of Sciences under Grant XDA27040300.

References

1. Towards compact cnns via collaborative compression. In: IEEE Conference on Computer Vision and Pattern Recognition, (CVPR)
2. Berman, M., Pishchulin, L., Xu, N., B.Blaschko, M., Medioni, G.: Aows: Adaptive and optimal network width search with latency constraints. In: 2020 IEEE Conference on Computer Vision and Pattern Recognition, (CVPR) (2020)
3. Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S.: Once for all: Train one network and specialize it for efficient deployment. In: International Conference on Learning Representations (2020)
4. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation **6**(2), 182–197 (2002). https://doi.org/10.1109/4235.996017
5. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. Ieee (2009)
6. Ding, X., Ding, G., Zhou, X., Guo, Y., Liu, J., Han, J.: Approximated oracle filter pruning for destructive cnn width optimization. In: IEEE Conference on Machine Learning, (ICML) (2019)
7. Ding, X., Ding, G., Zhou, X., Guo, Y., Liu, J., Han, J.: Global sparse momentum sgd for pruning very deep neural networks. In: Advances in Neural Information Processing Systems, (NeurIPS) (2019)
8. Dong, J.D., Cheng, A.C., Juan, D.C., Wei, W., Sun, M.: Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 517–531 (2018)
9. Dong, X., Yang, Y.: Network pruning via transformable architecture search. In: Advances in Neural Information Processing Systems. pp. 760–771 (2019)
10. Elsen, E., Dukhan, M., Gale, T., Simonyan, K.: Fast sparse convnets. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (June 2020)
11. Elsken, T., Metzen, J.H., Hutter, F.: Efficient multi-objective neural architecture search via lamarckian evolution. In: International Conference on Learning Representations, (ICLR) (2019)
12. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: International Conference on Learning Representations (ICLR) (2019)
13. Gao, S., Huang, F., Cai, W., Huang, H.: Network pruning via performance maximization. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2021)
14. Google-Inc.: Machine learning for mobile devices: Tenworflow lite. `https://www.tensorflow.org/lite` (2020)
15. Guo, S., Wang, Y., Li, Q., Yan, J.: Dmcp: Differentiable markov channel pruning for neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 1539–1547 (2020)
16. Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., Sun, J.: Single path one-shot neural architecture search with uniform sampling. In: European Conference on Computer Vision. pp. 544–560. Springer (2020)
17. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In: International Conference on Learning Representations (ICLR) (2016)

18. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural networks. In: Advances in Neural Information Processing Systems, (NIPS) (2015)
19. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015)
20. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: 2017 IEEE International Conference on Computer Vision (ICCV) (2017)
21. He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y.: Soft filter pruning for accelerating deep convolutional neural networks. In: Proceedings of International Joint Conference on Artificial Intelligence, (IJCAI) (2018)
22. He, Y., Liu, P., Wang, Z., Hu, Z., Yang, Y.: Filter pruning via geometric median for deep convolutional neural networks acceleration. In: IEEE Conference on Computer Vision and Pattern Recognition, (CVPR) (2019)
23. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S.: Amc: Automl for model compression and acceleration on mobile devices. In: in Proceedings of the European Conference on Computer Vision (ECCV) (September 2018)
24. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. CoRR **abs/1704.04861** (2017), `http://arxiv.org/abs/1704.04861`
25. Hsu, C.H., Chang, S.H., Liang, J.H., Chou, H.P., Liu, C.H., Chang, S.C., Pan, J.Y., Chen, Y.T., Wei, W., Juan, D.C.: Monas: Multi-objective neural architecture search using reinforcement learning. arXiv preprint arXiv:1806.10332 (2018)
26. Hu, H., Peng, R., Tai, Y.W., Tang, C.K.: Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. arXiv:1607.03250 (2016)
27. Joo, D., Yi, E., Baek, S., Kim, J.: Linearly replaceable filters for deep network channel pruning. In: The 34th AAAI Conference on Artificial Intelligence, (AAAI) (2021)
28. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Advances in neural information processing systems. pp. 598–605 (1990)
29. Li, C., Wang, G., Wang, B., Liang, X., Li, Z., Chang, X.: Dynamic slimmable network. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 8607–8617 (June 2021)
30. Li, F., Li, G., He, X., Cheng, J.: Dynamic dual gating neural networks. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). pp. 5330–5339 (October 2021)
31. Li, H., Kadav, A., Durdanovic, I.: Pruning filters for efficient convnets. In: Internation Conference on Learning Representation, (ICLR) (2017)
32. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710 (2016)
33. Lin, M., Ji, R., Zhang, Y., Zhang, B., Wu, Y., Tian, Y.: Channel pruning via automatic structure search. In: Proceedings of International Joint Conference on Artificial Intelligence, (IJCAI) (2020)
34. Liu, L., Zhang, S., Kuang, Z., Zhou, A., Xue, J., Wang, X., Chen, Y., Yang, W., Liao, Q., Zhang, W.: Group fisher pruning for practical network compression. In: International Conference on Machine Learning, (ICML) (2021)
35. Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K.T., Sun, J.: Metapruning: Meta learning for automatic neural network channel pruning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 3296–3305 (2019)

36. Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., Banzhaf, W.: Nsga-net: neural architecture search using multi-objective genetic algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 419–427 (2019)
37. Luo, J., Zhang, H., Zhou, H., Xie, C., Wu, J., Lin, W.: Thinet: Pruning cnn filters for a thinner net. IEEE Transactions on Pattern Analysis and Machine Intelligence, (TPAMI) (2018)
38. Molchanov, D., Ashukha, A., Vetrov, D.: Variational dropout sparsifies deep neural networks. In: in Proceedings of the International Conference on Machine Learning (2017)
39. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient inference. In: International Conference on Learning Representations, (ICLR) (2017)
40. Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J.: Importance estimation for neural network pruning. In: IEEE Conference on Computer Vision and Pattern Recognition, (CVPR) (2019)
41. Peng, H., Wu, J., Chen, S., Huang, J.: Collaborative channel pruning for deep neural networks. In: International Conference on Machine Learning, (ICML) (2019)
42. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. In: International Conference on Machine Learning (2018)
43. Ruan, X., Liu, Y., Li, B., Yuan, C., Hu, W.: Dpfps: Dynamic and progressive filter pruning for compressing convolutional neural networks from scratch. In: The 34th AAAI Conference on Artificial Intelligence, (AAAI) (2021)
44. Sandler, M., Howard, A.G., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
45. Su, X., You, S., Wang, F., Qian, C., Zhang, C., Xu, C.: Bcnet: Searching for network width with bilaterally coupled network. In: IEEE Conference on Computer Vision and Pattern Recognition, (CVPR) (2021)
46. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2820–2828 (2019)
47. Tang, Y., Wang, Y., Xu, Y., Deng, Y., Xu, C., Tao, D., Xu, C.: Manifold regularized dynamic network pruning. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 5018–5028 (June 2021)
48. Wang, J., Bai, H., Wu, J., Shi, X., Huang, J., King, I., Lyu, M., Cheng, J.: Revisiting parameter sharing for automatic neural channel number search. Advances in Neural Information Processing Systems **33** (2020)
49. Wang, K., Liu, Z., Lin, Y., Lin, J., Han, S.: Haq: Hardware-aware automated quantization with mixed precision. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
50. Wang, T., Wang, K., Cai, H., Lin, J., Liu, Z., Wang, H., Lin, Y., Han, S.: Apq: Joint search for network architecture, pruning and quantization policy. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 2078–2087 (2020)
51. Wang, W., Chen, M., Zhao, S., Chen, L., Chen, L., Hu, J., Liu, H., Cai, D., He, X., Liu, W.: Accelerate cnns from three dimensions: A comprehensive pruning framework. In: International Conference on Machine Learning, (ICML) (2021)

52. Wang, Z., Li, C., Wang, X.: Convolutional neural network pruning with structural redundancy reduction. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2021)
53. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: in Advances in Neural Information Processing Systems (NeurIPS) (2016)
54. Yang, H., Zhu, Y., Liu, J.: Ecc: Platform-independent energy-constrained deep neural network compression via a bilinear regression model. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2019)
55. Yang, T., Howard, A.G., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., Adam, H.: Netadapt: Platform-aware neural network adaption for mobile applications. In: European Conference on Computer Vision, (ECCV) (2018)
56. Yang, Z., Wang, Y., Chen, X., Shi, B., Xu, C., Xu, C., Tian, Q., Xu, C.: Cars: Continuous evolution for efficient neural architecture search. In: IEEE Conference on Computer Vision and Pattern Recognition, (CVPR) (2020)
57. Yu, J., Huang, T.S.: Autoslim: Towards one-shot architecture search for channel numbers. CoRR **abs/1903.11728** (2019), `http://arxiv.org/abs/1903.11728`
58. Yu, J., Huang, T.S.: Universally slimmable networks and improved training techniques. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 1803–1811 (2019)
59. Yu, J., Jin, P., Liu, H., Bender, G., Kindermans, P.J., Tan, M., Huang, T., Song, X., Pang, R., Le, Q.: Bignas: Scaling up neural architecture search with big single-stage models. In: European Conference on Computer Vision, (ECCV) (2020)
60. Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., Huang, J., Zhu, J.: Discrimination-aware channel pruning for deep neural networks. In: in Advances in Neural Information Processing Systems (NeurIPS) (2018)