DATA AGNOSTIC FILTER GATING FOR EFFICIENT DEEP NETWORKS

Hongyan Xu[1,5] *Xiu Su*[2] *Shan You*[3,4*] *Tao Huang*[3] *Fei Wang*[3] *Chen Qian*[3]
Changshui Zhang[3] Chang Xu[2] Dadong Wang[5] Arcot Sowmya[1]
[1]School of Computer Science and Engineering, University of New South Wales, Australia
[2]School of Computer Science, Faculty of Engineering, University of Sydney, Australia
[3]SenseTime Research
[4]Department of Automation, Tsinghua University,
Institute for Artificial Intelligence, Tsinghua University (THUAI),
Beijing National Research Center for Information Science and Technology (BNRist)
[5]Data61, The Commonwealth Scientific and Industrial Research Organisation (CSIRO)
hongyan.xu@unsw.edu.au, xisu5992@uni.sydney.edu.au, {youshan,wangfei,qianchen}@sensetime.com
zcs@mail.tsinghua.edu.cn, c.xu@sydney.edu.au,Dadong.wang@csiro.au,a.sowmya@unsw.edu.au

ABSTRACT

Filter pruning is essential for deploying a well-trained CNN model on edge computation devices with a target computation budget (*e.g.*, FLOPs). Current filter pruning methods mainly focus on leveraging feature maps to analyze the importance of filters, and prune those with less impact on the value of the CNN's loss function, thereby ignoring the variance of input batches to differences in sparse structure over the filters. In this paper, we propose a data-agnostic filter pruning method that uses an auxiliary network named Dagger module to induce pruning with the pre-trained weights as input. Besides, to help prune filters with a preset FLOPs constraint, we utilize an explicit FLOPs-aware regularisation mechanism to directly promote pruning filters toward the target FLOPs. Experimental results on CIFAR-10 and ImageNet datasets show that the proposed filter pruning method surpasses the state-of-the-art.

***Index Terms*—** Deep learning, Filter pruning, Model compression, Data agnostic, Dagger module, FLOPs-aware regularization.

1. INTRODUCTION

Recently, Artificial Intelligence (AI) engines with deep learning techniques have achieved remarkable success in various tasks [1, 2, 3]. However, aimed at the state-of-the-art accuracy performance, the conventionally trained CNN models usually have huge model sizes, and are unsuited for deployment on low-end computational devices. In this way, a natural problem goes that *besides the basic accuracy performance, how we can develop a ready-to-deploy model under a certain computation budget[4, 5], such as FLOPs*. Luckily, due to the development of model compression[6, 7, 8, 9], pruning has been an efficient way to acquire light models with existing heavy models.

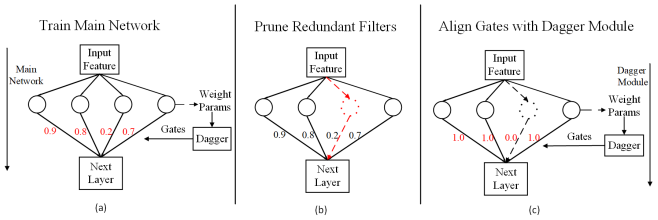

Fig. 1: Descriptive diagrams of our proposed method.

Currently, pruning methods can be divided into weight pruning and filter pruning. Of these, filter pruning is more competitive, as it can produce a lightweight model with a consistent network structure of pre-trained model and is friendly to current off-the-shelf deep learning frameworks. This paper refers to filters that are redundant for network performance as redundant filters. Filter pruning works by first finding and pruning the redundant filters, then retraining (fine-tuning) the pruned network to recover its performance.

For pruning redundant filters, many works propose leveraging the trained weights with their scale value for its importance. However, with the effect of batch normalization layer, the weights of architecture have no relative to its importance, *i.e.*, equal scaling of the filter weights will not impact the network results. Moreover, due to the data-agnostic property of deep learning, the importance of filters may not be linearly scaled to the value of weights. To solve these issues, we propose a plug-and-play module named dagger to mine the importance of different filters. To safely remove the dagger module after pruning without harm to the performance, we propose to leverage the binary gates for the remaining filters. We leverage the pre-trained weights as input for the dagger module, and thus we can acquire the pruned architecture very efficiently and achieve state-of-the-art performance results.

*Corresponding authors.

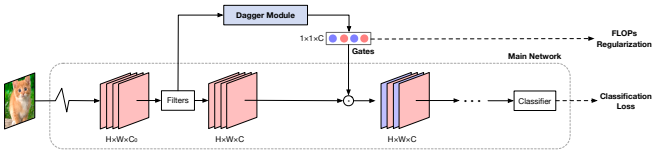

Fig. 2: Overall framework of our proposed method.

2. MODELING REDUNDANCY OF FILTERS WITH DAGGER MODULE

2.1. Binary gates

Denote the original network as $\mathcal{N}$ with pre-trained weights Ω^*. Suppose the network has L layers, and feature maps for each layer l are denoted as $F^l \in \mathbb{R}^{N\times n^l\times h^l\times w^l}$, where N is the batch size, n^l is the number of filters, and h^l and w^l are the height and width of the feature map. We assign a binary gate $g \in \{0,1\}$ to each filter, where $g = 1$ means that the corresponding filter should be retained and vice versa.

For feature map F^l the w.r.t. l-th layer, its valid filter number $\tilde{n}^l$ is exactly the amount of non-zero gates. Specifically, for the non-pruned pre-trained model, its gates are all ones, and $\tilde{n}^l = n^l$. The FLOPs of the pruned network is determined by the gates $\mathcal{G} = \{\boldsymbol{g}^l\}_{l=1}^L$, *i.e.*

$$FLOPs(\mathcal{G}) = \sum_{l=1}^{L} \mathcal{F}^l(\boldsymbol{g}^l), \tag{1}$$

where $\mathcal{F}^l(\cdot)$ is the FLOPs calculator w.r.t. the l-th layer. For example, if the l-th layer has n^l filters, then for a 1×1 convolutional layer, its FLOPs calculator $\mathcal{F}^l(\boldsymbol{g}^l)$ with gate $\boldsymbol{g}^l$ will be

$$\mathcal{F}^l(\boldsymbol{g}^l) = \left\|\boldsymbol{g}^{l-1}\right\|_0 \times h^l \times w^l \times \left\|\boldsymbol{g}^l\right\|_0. \tag{2}$$

As a result, we can formulate the number of filters as a mixed 0-1 binary optimization problem:

$$\begin{aligned} \min_{\mathcal{G},\Omega} \quad & \mathcal{L}(\mathcal{G},\Omega;\mathcal{D}_{tr},\Omega^*) \\ \text{s.t.} \quad & \text{FLOPs}(\mathcal{G}) \le C,\ \mathcal{G}\in\{0,1\}, \end{aligned} \tag{3}$$

where Ω is the weights of network $\mathcal{N}$ with pre-trained weights Ω^* and $\mathcal{D}_{tr}$ is the training dataset. Note that the hard constraint in Eq.(3) can amount to a version indicated by acceleration rate r, *i.e.*, $\text{FLOPs}(\mathcal{G}) \le C_0/r$, where C_0 is the overall FLOPs of the pre-trained model $\mathcal{N}(\Omega^*)$.

2.2. Modeling gates with the Dagger module

It is known that 0-1 optimization is an NP-hard problem. Therefore, we implement the filter pruning by considering a real-number gate in the interval $[0,1]$, as shown in Figure 1. We adopt a Dagger module with pre-trained weights to indicate the gates for filter pruning, *i.e.*, $\boldsymbol{g}^l(\mathcal{M}) = \mathcal{M}(\mathcal{W}^l;\theta^l)$.

To reduce the computation complexity of the Dagger module, we first merge the filter $\mathcal{W}^l$ by average pooling to fit the size of gates $\boldsymbol{g}^l$. The combined filters then pass

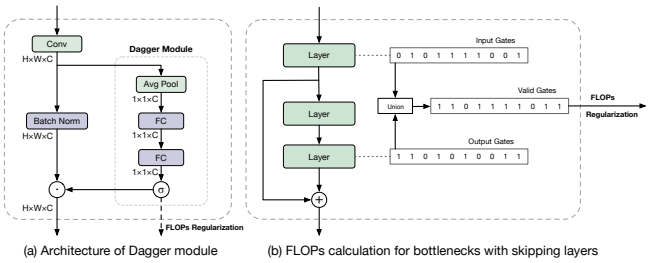

Fig. 3: Schematic diagram of Dagger module

through two fully connected (FC) layers and are activated via a sigmoid function $\sigma(\cdot)$, *i.e.*,

$$\boldsymbol{g}^l(\mathcal{M}) = \text{Sigmoid}(\text{FC}(\text{ReLU}(\text{FC}(\text{AvgPooling}(\mathcal{W}^l))))). \tag{4}$$

To better model a real gate, along with the main network we leverage an auxiliary network named the Dagger module to generate real-number gates with the help of trained weights Ω. The generated filter-wise binary gates are directly applied to output feature maps with dot products for identifying redundant filters. The framework is shown in Figure 2.

3. PRUNING WITH FLOPS-AWARE REGULARIZATION

With the gates generated by the Dagger module, Eq.(3) is relaxed into a continuous optimization problem. However, the hard constraint of FLOPs in Eq.(3) depends on the ℓ_0 norm of gates $\boldsymbol{g}^l$, which is computationally infeasible for optimization. So, we approximate it by adopting the surrogate ℓ_1 norm. For example, the FLOPs of gates of Eq.(2) can be estimated as $\mathcal{R}^l(\boldsymbol{g}^l;\mathcal{M}) = \left\|\boldsymbol{g}^{l-1}\right\|_1 \times h^l \times w^l \times \left\|\boldsymbol{g}^l\right\|_1$.

And the total FLOPs in Eq.(1) can also be indicated by

$$\mathcal{R}(\mathcal{G};\mathcal{M}) = \sum_{l=1}^{L} \mathcal{R}^l(\boldsymbol{g}^l;\mathcal{M}). \tag{5}$$

If the augmented network is initialized with all-one gates, $\mathcal{R}(\mathcal{G};\mathcal{M})$ will accurately estimate FLOPs since they are equal. $\mathcal{R}(\mathcal{G};\mathcal{M})$ can be regarded as a regularization for reducing the FLOPs of the pre-trained model. Moreover, since the regularization $\mathcal{R}(\mathcal{G};\mathcal{M})$ is continuous, it enables the optimization to resort to various gradient-based optimizers, such as stochastic gradient descent (SGD).

3.1. pruning filters under accurate estimation

The loss function defined in Eq.(3) cannot be directly optimized through gradient descent. So we reformulated Eq.(3) according to Lagrange multiplier and leverage Eq.(5) as the estimation of FLOPs, which can be formulated as:

$$\mathcal{L}_{all} = \mathcal{L}(\mathcal{G},\Omega,\mathcal{M};\mathcal{D}_{tr},\Omega^*) + \lambda\cdot\mathcal{R}(\mathcal{G};\mathcal{M}), \tag{6}$$

where $\lambda > 0$ is the coefficient of the Lagrange multiplier. In detail, the regularization $\lambda > 0$ is not always a good estimation due to the gap between ℓ_0 and ℓ_1 norm. Therefore,

Algorithm 1 Data agnostic filter gating for efficient deep networks

Input: A well-trained model $\mathcal{N}$ with weights Ω^*. Training dataset $\mathcal{D}_{tr}$. FLOPs budget C. All gates set $\mathcal{G}$.

initialize Dagger module $\mathcal{M}$ with Dagger weights Θ
one-gate set $\mathcal{A} = \mathcal{G}$, zero-gate set $\mathcal{B} = \emptyset$
while FLOPs$> C$ **do**
  align gates in one-gate set $\mathcal{A}$
  optimize the gates with fixed weights Ω^*
  obtain the smallest gates with ratio r as $\mathcal{Z}$
  update $\mathcal{B} = \mathcal{B} \bigcup \mathcal{Z}$ and $\mathcal{A} = \mathcal{A} - \mathcal{Z}$
  calculate the valid FLOPs via $\mathcal{A}$
  fine-tune the weights with fixed all gates in $\mathcal{A}$ being ones
end while

Output: retained gates $\mathcal{A}$

Eq.(6) is not suitable for learning gates in an end-to-end manner but should be leveraged under alternate iteration updates. Our proposed algorithm works iteratively, as in Algorithm 1.

3.2. Iterative optimization with FLOPs examination

In the following, we refer to the original network $\mathcal{N}$ as *main network*. As mentioned earlier, we implement the iterative update for the gates (Dagger module $\mathcal{M}$) and the weights of the main network $\mathcal{N}$, as detailed below.

3.2.1. Greedy pruning gates with fixed weights

The Dagger module generates gates for the main network and guides prune redundant filters with the main network fixed. And the main network provides classification evaluation for gates so that the retained gates can maintain the performance. The following objective can optimize the Dagger module:

$$\mathcal{L}_{\mathcal{M}} = \mathcal{L}(\mathcal{G}, \mathcal{M}; \mathcal{D}_{tr}, \Omega) + \lambda \cdot \mathcal{R}(\mathcal{G}; \mathcal{M}), \tag{7}$$

where Ω is fixed compared to Eq.(6); therefore, the gates can be optimized under the mutual supervision of classification loss and FLOPs-aware regularization.

Before optimizing the Dagger module and the gates, we need to fix the estimation gap of FLOPs regularization $\mathcal{R}(\mathcal{G}; \mathcal{M})$. We first retrain the Dagger module to enable its output gates to be 0.5. Then we add 0.5 to the output gates so that the values of gates are equal to 1. The advantages of these aligning gates are two-folds. First, after alignment, the gates are all ones; So $\mathcal{R}(\mathcal{G}; \mathcal{M})$ can accurately estimate FLOPs for further regularization. Second, aligning gates with 0 before the sigmoid activation corresponds to its maximum slopes, enhancing the impact of regularization $\mathcal{R}(\mathcal{G}; \mathcal{M})$ for optimizing gates. After the alignment, we can safely optimize Eq.(7) to obtain gates.

Table 1: Performance comparsion of MobileNetV2 and VGGNet on CIFAR-10.

Model	Groups	Methods	Flops	Params	Acc
MobileNetV2	200M	DCP[10]	218M	-	94.69%
		Uniform	207M	1.5M	94.57%
		Random	207M	-	94.20%
		Dagger	207M	1.9M	94.90%
	148M	MuffNet[11]	175M	-	94.71%
		Uniform	148M	1.1M	94.32%
		Random	148M	-	93.85%
		Dagger	148M	1.7M	94.85%
	88M	AutoSlim[12]	88M	1.5M	93.20%
		Uniform	88M	0.6M	94.32%
		Random	88M	-	93.85%
		Dagger	88M	1.1M	94.49%
		AutoSlim[12]	59M	0.7M	93.00%
VGGNet	200M	DCP[10]	199M	10.4M	94.16%
		SSS[13]	199M	5.0M	93.63%
		PFS[14]	199M	-	93.71%
		Uniform	199M	10.0M	93.45%
		Random	199M	-	93.02%
		Dagger	199M	6.0M	94.26%
	119M	CGNets[15]	117M	-	92.88%
		Uniform	119M	6.1M	93.03%
		Random	119M	-	92.22%
		Dagger	119M	2.7M	93.88%

3.2.2. Fine-tuning weights with fixed gates

After some gates are pruned, we need to fine-tune the weights of the main network with fixed gates. However, after the greedy pruning, those values of retained gates are no longer ones but in $(0, 1)$. Implementing fine-tuning weights will further worsen the coupled issue since the weights are trained from biased gates. Therefore, we set all retrained gates to ones and fine-tune the main network.

$$\mathcal{L}_{\Omega} = \mathcal{L}(\Omega; \mathcal{D}_{tr}, \mathcal{G}), \tag{8}$$

With Eq.(8), we compensate for the pruned filters by fine-tuning the retained filters.

3.3. Dealing with skipping layers

To construct the FLOPs-aware regularization $\mathcal{R}(\mathcal{G}; \mathcal{M})$, the FLOPs need to be calculated. For a bottleneck with skipping layers as Figure 3(b), each layer will have its gates $\boldsymbol{g}^l$. Denote the gates of input and output as $\boldsymbol{g}^{in}$ and $\boldsymbol{g}^{out}$. For a pre-trained model, the size of $\boldsymbol{g}^{in}$ and $\boldsymbol{g}^{out}$ is the same. The valid gates of $\boldsymbol{g}^{in}$ and $\boldsymbol{g}^{out}$ should be their union as Figure 3(b).

$$\boldsymbol{g} = \boldsymbol{g}^{in} \vee \boldsymbol{g}^{out} = 1 - (1 - \boldsymbol{g}^{in}) \cdot (1 - \boldsymbol{g}^{out}), \tag{9}$$

where $\vee$ is the union operation.

Table 2: Performance comparison of pruned ResNet-50 and MobileNetV2 on ImageNet dataset.

Model	Methods	Flops	Params	Acc
ResNet50	SFP[18]	2.4G	-	74.6%
	FPGM[19]	2.4G	-	75.6%
	LEGR[20]	2.4G	-	75.7%
	PFS[14]	2.0G	-	75.6%
	MetaPruning[21]	2.0G	-	75.4%
	Uniform	2.0G	10.2M	74.1%
	Random	2.0G	-	73.2%
	Dagger	2.0G	11.7M	**76.1%**
MobileNetV2	MetaPruning[21]	140M	-	68.2%
	LEGR[20]	150M	-	69.4%
	Uniform	140M	2.7M	67.6%
	Random	140M	-	67.1%
	Dagger	140M	2.84M	**69.5%**
	MetaPruning[21]	105M	-	65.0%
	Uniform	106M	1.5M	64.1%
	Random	106M	-	63.5%
	Dagger	106M	2.46M	**67.2%**

4. EXPERIMENTAL RESULTS

4.1. Configuration and settings

Training. We optimize the Dagger module and main network for 400 (100) iterations with a batch size of 320 (64) for the ImageNet (CIFAR-10) dataset. The pruning rate per update is set to 0.6%, and the balance parameter λ is set to 8 for all networks. Once the FLOPs budget is achieved, we will finetune the pruned weights with the learning rate initialized to 0.01. Besides, we also cover two vanilla baselines. The first one is *Uniform*, *i.e.*, shrinking the width of a network by the fixed rate to meet the requirement of FLOPs budget. The second one is a variant of the random set of filters within each layer, denoted as *Random*, which is implemented ten times and reports the average performance. Concretely, we randomly adjust the number of filters within Uniform in a certain range to meet the FLOPs budget.

4.2. Experiments on CIFAR-10 dataset

Networks. We conduct filter pruning on VGGNet [16] and MobileNetV2 [17]. VGGNet (MobileNetV2) has 20M (2.2M) parameters and 399M (297M) FLOPs with an error rate of 6.01% (5.53%).

Results. As shown in Table 1, our method achieves the best accuracy w.r.t. different FLOPs on the CIFAR-10 dataset. In detail, for VGGNet, our pruned 50% FLOPs VGGNet outperforms the DCP, Slimming, and PFS by 0.26%, 0.45%, and 0.54%, respectively, and even surpass the pre-trained model by 0.26%. Besides, compared with the Uniform and Random baselines, our pruned VGGNet-19 can increase the accuracy by more than 0.80%. The performance of our 207M MobileNetV2 is better than the DCP and pre-trained model by 0.44% and 0.22%, respectively, which proves that our method can achieve promising results even with small budgets.

Table 3: Efficiency of pruned ResNet-50 and MobileNetV2 on ImageNet Dataset w.r.t. different pruning ratios.

Model	Pruning ratios	Params	FLOPs	Time cost(h)
ResNet-50	30%	16.5M	2.9G	1.5×8
	50%	11.7M	2.0G	2.5×8
	70%	9.2M	1.2G	3.4×8
MobileNetV2	30%	3.19M	210M	1.0×8
	50%	2.96M	150M	1.4×8
	70%	2.54M	90M	1.8×8

4.3. Experiments on ImageNet dataset

Results of ResNet-50. The pretrained ResNet-50 has 25.5M parameters and 4.1G FLOPs with 76.6% Top-1 accuracy. As shown in Table 2, our algorithm outperforms the SFP [18], FPGM [19] and LEGR [20] by 1.5%, 0.5%, and 0.4%, respectively, while our pruned ResNet-50 has even smaller FLOPs (by 0.4G). Besides, our pruning network has a 0.5% increase in accracy compared to PFS[14].

Results of MobileNetV2. The pre-trained MobileNetV2 has 3.5M parameters, and 300M FLOPs with 68.2% Top-1 accuracy. We prune the network under two different FLOPs budgets (140M and 106M). As shown in Table 2, by pruning the MobileNetV2 to 140M FLOPs, our pruned MobileNetV2 outperforms the pre-trained MobileNetV2 by 1.3%. Our method also leads to a 2.4% and 1.9% increase in Top-1 accuracy compared with the baseline of Random and Uniform for FLOPs 140M (106M). Moreover, with the same FLOPs budget, our method can surpass MetaPruning by a large margin of 1.3% for FLOPs 140M (106M).

4.4. Efficiency of Dagger in pruning filters

To investigate the efficiency of our method in pruning filters, we report the time cost on pruning w.r.t. different pruning ratios in Table 3. All experiments are implemented with 8 NVIDIA 1080 Ti GPUs.

As in Table 3, our method can quickly get the desired model size with the pre-trained model. We optimize the Dagger module and main network for 400 (100) iterations with the batch size of 320 (64) and pruning ratio of 0.6% for ImageNet (CIFAR-10) dataset in each update. Therefore, taking 50% FLOPs budget as an example, we only need to go through about 8 (10) epochs for ImageNet (CIFAR-10) dataset.

5. CONCLUSION

In this paper, we leverage the Dagger module with pre-trained weights to model the gates for filter pruning. Since our Dagger module is indirectly related to input batches, we can safely remove the Dagger module after pruning. Besides, we assign a binary gate for each filter to indicate whether the filter should be retrained or pruned, which helps generate dataset-related gates. Moreover, we involve an explicit FLOPs regularization to guide the pruning of redundant filters. Extensive experiments on CIFAR-10 and ImageNet datasets show the superiority of our method to other pruning methods.

6. REFERENCES

[1] Fei Wang, Liren Chen, Cheng Li, Shiyao Huang, Yanjie Chen, Chen Qian, and Chen Change Loy, "The devil of face recognition is in the noise," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 765–780.

[2] 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.

[3] Hongyan Xu, Xiu Su, Yi Wang, Huaiyu Cai, Kerang Cui, and Xiaodong Chen, "Automatic bridge crack detection using a convolutional neural network," *Applied Sciences*, vol. 9, no. 14, pp. 2867, 2019.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[5] Xiu Su, Shan You, Mingkai Zheng, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu, "K-shot nas: Learnable weight-sharing for nas with k-shot supernets," in *International Conference on Machine Learning*. PMLR, 2021, pp. 9880–9890.

[6] Xiu Su, Shan You, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu, "Bcnet: Searching for network width with bilaterally coupled network," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 2175–2184.

[7] 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.

[8] Xiu Su, Shan You, Jiyang Xie, Mingkai Zheng, Fei Wang, Chen Qian, Changshui Zhang, Xiaogang Wang, and Chang Xu, "Vision transformer architecture search," *arXiv preprint arXiv:2106.13700*, 2021.

[9] Xiu Su, Tao Huang, Yanxi Li, Shan You, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu, "Prioritized architecture sampling with monto-carlo tree search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 10968–10977.

[10] 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*, 2018, pp. 875–886.

[11] 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/CVF International Conference on Computer Vision Workshops*, 2019, pp. 0–0.

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

[13] Zehao Huang and Naiyan Wang, "Data-driven sparse structure selection for deep neural networks," in *The European Conference on Computer Vision (ECCV)*, September 2018.

[14] Yulong Wang, Xiaolu Zhang, Lingxi Xie, Jun Zhou, Hang Su, Bo Zhang, and Xiaolin Hu, "Pruning from scratch," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020, vol. 34, pp. 12273–12280.

[15] Weizhe Hua, Christopher De Sa, Zhiru Zhang, and G Edward Suh, "Channel gating neural networks," *arXiv: Learning*, 2018.

[16] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[17] 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*, 2018, pp. 4510–4520.

[18] 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.

[19] 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*, 2019, pp. 4340–4349.

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

[21] 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/CVF International Conference on Computer Vision*, 2019, pp. 3296–3305.