Filter Pruning via Automatic Pruning Rate Search*

Qiming Sun[1], Shan Cao[1], and Zhixiang Chen[2]

[1] Shanghai University, Shanghai 200444, China
cshan@shu.edu.cn
[2] The University of Sheffield, Sheffield, S1 4DP, UK

Abstract. Model pruning is important for deploying models on devices with limited resources. However, the searching of optimal pruned model is still a significant challenge due to the large space to be exploited. In this paper, we propose an Automatic Pruning Rate Search(APRS) method to achieve automatic pruning. We reveal the connection between the model performance and Wasserstein distance to automatic searching optimal pruning rate. To reduce the search space, we quantify the sensitivity of each filter layer by layer and reveal the connection between model performance and Wasserstein distance. We introduce an end-to-end optimization method called Pareto plane to automatically search for the pruning rate to fit the overall size of the model. APRS can obtain more compact and efficient pruning models. To verify the effectiveness of our method, we conduct extensive experiments on ResNet, VGG and DenseNet, and the results show that our method outperforms the state-of-the-art methods under different parameter settings.

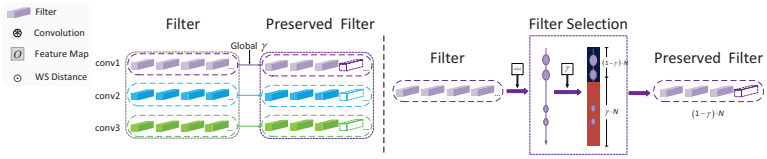
Keywords: Filter pruning · Pruning rate search · Pareto optimization.

1 Introduction

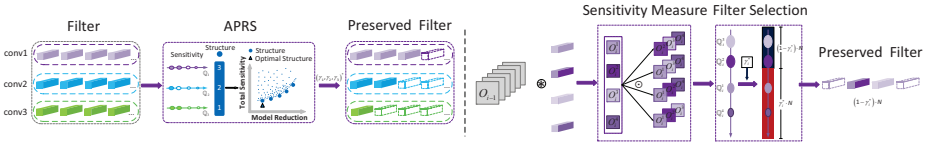
Convolutional neural networks (CNNs) have achieved great success in computer vision applications such as image classification [12,31,28], object detection [61,62], and segmentation [60]. However, most state-of-the-art CNNs are difficult to run in real-time on resource-limited embedded terminals due to high requirements on computing power and memory footprint. Reducing the computational cost and model size is an eternal and important requirement. To achieve this, prevalent technologies include network pruning [33,22,34,35,36,32], weight quantization [37,38,39,40,41], knowledge distillation [42,43,44,45,46], and low-rank decomposition [47,48,49].

CNN pruning can be viewed as a search problem in the pruning solution space. The solution space of pruning consists of the pruning rate of each layer, and each pruning solution corresponds to a sub-net or pruned candidate. In huge

* This work is supported by National Natural Science Foundation of China (No.61904101) and Shanghai Committee of Science and Technology of China (No.21ZR1422200).



(a) Pruning pipeline based on individual filter

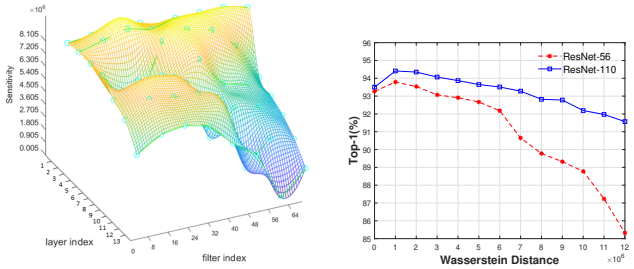


(b) Pruning pipieline based on feature maps before and after pruning

Fig. 1: **Two pipelines for pruning tasks. γ_i is the pruning rate of i-th layer. Fig.1(a) shows a traditional pipeline based on weight attribute sorting. The left and right are the result and process of manually setting the pruning rate. (b) is our pipeline based on the information difference of feature maps before and after pruning. (Left) Determining optimal pruned structure using automatic structure redistribution. (Right) Filter selection using the wasserstein distance.**

search space, the task of pruning is to get a more compact and efficient model to adapt to more application scenarios and devices. Filter pruning is dedicated to searching for filters that can satisfy the accuracy under the target model size. Its process has two steps, filter sorting and pruning rate selection. In the first step, measurement criteria are usually designed based on weights or output feature maps to evaluate the importance of filters, and eliminate the weights with less importance in units of filters. Filter sorting is mainly based on experience to propose custom metrics.

Pruning Rate Selection: After sorting the importance of filters, it is necessary to find the pruning rate to prune the pre-trained model. The choice of pruning rate in CNN has a huge impact on performance [10]. When a suitable pruning rate is used, the accuracy of the pruned model can even exceed the original model [65]. The choice of pruning rate can be divided into two ways: pre-defined and automatic search [1]. Pre-defined methods manually specify the clipping rate of each layer and determine the target structure in advance. Such methods usually remove parameters in each layer with fixed pruning rate [2,6], as shown on the left side of Fig. 1(a). Despite the target model size can be achieved, each layer retains a large amount of redundancy. The pre-defined methods cannot compare the pros and cons of different pruning rates, and it is difficult to prove the correctness of the pre-defined pruning rates. Moreover, the optimal pruning rate combination will change with the requirements of the model size, so the pre-defined method requires a lot of manual labor. Automatic search for pruning rates has begun to emerge, and more suitable pruning rates are automatically obtained through theoretical guidance, enabling end-to-end fine-tuning without



(a) Visualize the sensitivity of different layers on VGG-16.

(b) The effect of distance.

Fig. 2: Visualization of the pruning loss. (a) is the sensitivity of some filters in convolutional layers of VGG-16. (b) reveal the connection between the model performance and wasserstein distance for ResNet-56 and ResNet-110.

manual labor. A specific compression rate on a continuous space can be automatically generated using the DDPG algorithm in reinforcement learning [17]. Singh *et al* [7] models the pruning problem as a min-max optimization problem, and control the accuracy loss by adjusting the pruning rate. Automated deep compression [8] uses reinforcement learning to learn the optimal sparsity rate for each layer according to different requirements such as guaranteeing accuracy or limiting the amount of computation. However, in general, the current automatic methods do not quantitatively analyze the importance of each filter, and cannot maximize the performance improvement brought by the pruning rate.

Inspired by automatic pruning, the sensitivity analysis of each filter is introduced in our APRS to quantitatively prune the redundancy of the model , so as to search for a more suitable pruning rate. As shown in Fig. 1(b), we use Wasserstein distance to calculate the sensitivity and reveal that there is a strong correlation between the sensitivity of the pruned structure and the accuracy after fine-tune, this can help us quickly estimate the accuracy. Fig. 2(b) analyzes the relationship between the sub-network performance and sensitivity of ResNet-56 and ResNet-110. When the distance is within 7M, the sub-network accuracy exceeds the baseline, which means that the network can be more compact and efficient. The role of sensitivity can be explained in terms of the relative importance of the filters. We visualize the sensitivities of different layers and use the flatness of the sensitivity curves to characterize the relative contributions of the filters. As shown in Fig. 2(a), Layers with large curve fluctuations have high redundancy and vice versa. As the number of layers deepens, the redundancy increases greatly. APRS automatically assigns larger pruning rates to sensitive layers. The pruning rate depends on the number of filters retained in each layer. The filters retained under quantitative sensitivity analysis constituted a large number of sub-structure candidates. In this paper we attain a appropriate pruning rate by searching the sub-structure with Pareto optimization algorithm.

To sum up, our main contributions are three-fold:

1. Based on extensive statistical validation, we demonstrate that the impact of pruning can be effectively measured by Wasserstein distance. We were the first to introduce Wasserstein distance to predict model performance.
2. We propose an automatic search method to assign optimal pruning power without manual labor, which greatly saves the computational cost of automatic search for pruning rate.
3. APRS achieves state-of-the-art pruning performance compared to many more sophisticated methods. In the VGG-16 experiments, APRS can help us get 1.3% to 2.8% higher accuracy than the state-of-the-art algorithms. Meanwhile, it can greatly reduce the FLOPs of the model for computationally intensive devices.

2 Related Works

Different approaches have been proposed to obtain more compact, faster and more efficient networks, mainly including efficient neural network design [12,31,28,2], pruning [19,21,34,18,14,13], quantization [24,41,39], and distillation [44,45,46,43]. This paper briefly discusses related work on filter pruning and automated machine learning for pruning.

CNN pruning is mainly divided into unstructured pruning [4,11,69], where arbitrary parameters can be removed, and structured pruning [6,32,36], where a set of weights is removed. Structured pruning is divided into filter pruning and channel pruning [64], etc. Filter pruning removes filters under certain metric. Since the original convolutional structure in filter prunnnig is still preserved, no dedicated hardware is required for implementation. [5] uses $l1$-norm as an indicator for judging the importance of filters. [10] uses the rank of the feature map of each layer to judge the importance of the filter, [16] uses the geometric median to prune the model, and delete the filter with a lower median. Some works judge whether the filter contributes [14] according to the importance of the output channel, and discard the unimportant filters. [15] prunes filters using Taylor expansion as a measure of pruning criteria. Channel Pruning regards the same output channel of different filters as a group, [13] proposes to prune channels through LASSO regression-based channel selection and least squares reconstruction. [6] uses the scaling factor γ in the BN layer to measure the importance of channels. Mitsuno *et al.* [25] analyzes the effect of channels of different convolutional layers on model performance.

AutoML Based Pruning. Traditional manual pruning methods rely heavily on expert experience to balance the amount of calculation, parameter size and accuracy, and usually cannot obtain the best pruning rate strategy. Liu *et al* [1] proposed that pruning can be solved by automated machine learning (AutoML), which determines hyperparameters through learning. After the emergence of AutoML, it was quickly combined with the network structure search method [66], and model pruning was performed by searching for the structure of the model. Most of the previous AutoML-based pruning methods are layer-by-layer pruning. The typical work [17] proposes to adaptively determine the compression rate of

Table 1: Comparison of APRS and other state-of-the-art methods on VGG16 with CIFAR10, "No FT" means no fine-tuning is required, "Layer-wise" stand for hierarchical reallocation. "Auto-P" means automatically assign optimal pruning substructure.

Method	No FT	Layer-wise	Auto-P	Param.↓(%)	Acc.(%)
L1 [5]	✗	✓	✗	64.0	93.40
EagleEye [23]	✓	✗	✓	-	-
Hrank [10]	✗	✓	✗	82.1	93.43
ABCPruner [22]	✗	✗	✓	88.7	93.08
AutoCompress [65]	✗	✓	✓	89.0	93.21
APRS(ours)	✓	✓	✓	**89.9**	**94.21**

each layer through reinforcement learning. In addition, A platform-aware neural network adaptation (NetAdapt) method [67] was proposed to sequentially obtain suitable pruning rate through progressive compression and short-term finetune. The work of [68] uses layer-by-layer greedy search to automate compression models. Currently, some AutoML based pruning methods pay attention to the compression rate of pruning. Table 1 shows the exploration of pruning rates by recent automatic methods. HRank [10] introduces manually predefined hierarchical pruning rates, but cannot search for more accurate pruning rates. EagleEye [23] proposes to find the relative optimal pruning rate, but it can only be selected from several candidate models. [22] supports automatic search of pruned structures, but cannot be pruned hierarchically and cannot achieve high compression rates. Due to the lack of sensitivity analysis and subsequent optimization of pruning schemes in previous AutoML based methods, it is difficult to find the optimal number of filters for each layer of convolution. Newer research shows that the essence of pruning is to determine the number of filters, not to choose the important filter/channel [1]. Cai *et al* [24] proposes to measure the sensitivity of each quantization scheme to obtain a quantization scheme with less loss. Inspired by [1,24], the APRS proposed in this paper considers the total sensitivity and regards the problem of automatic pruning as searching for an appropriate number of filters. Pareto optimization is applied to model formulation for the first time, which can accurately reduce parameters size or computational cost.

3 Methodology

In this paper, our task is to automatically obtain a more compact and efficient network given a large pre-trained model, while minimizing accuracy reduction and minimizing computational cost and memory consumption. APRS will automatically find filter pruning rates for each layer, and possible pruning rate combinations $\gamma_1, \gamma_2, ..., \gamma_L$ for L layers. We focus on finding suitable structure vectors to search for sub-structures, and acquire the target pruned model after optimization in our method. As shown in Fig. 3, first we measure the Wasserstein distance

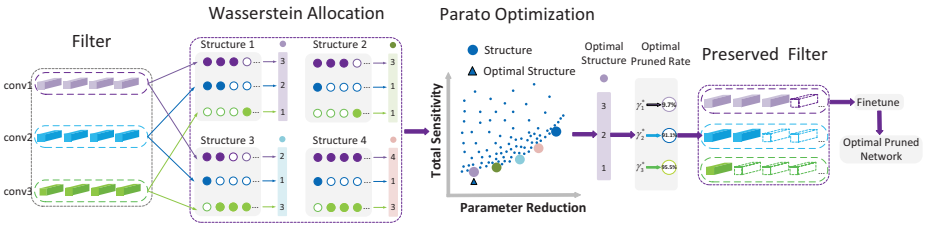


Fig. 3: Framework of our accurately filter pruning method (ASPR) for automatically optimizing pruning rates.

of all filters, which will guide us how to sort, called the sensitivity coefficient. According to the sensitivity coefficients, the unimportant filters in each layer can be roughly sorted and removed, which initially reduces the solution space in our APRS. The remaining filters are then reorganized across layers to construct as many the number of filter combinations(sub-structure vectors) as possible, and the total sensitivity of each sub-structure is calculated. All sub-structure vectors are sent into the Pareto plane, and its position on the Pareto plane can be determined according to the size and sensitivity of the sub-structure, so the sub-structure vector with the least perturbation can be intuitively locked under the constraint of model size. The sub-structure vector with the least sensitivity is output by the Pareto plane, and the pruning rate of each layer is fed back to the model to help us find the best sub-structure with minimal loss of accuracy. Finally, the sub-structure is fine-tuned end-to-end, and the fine-tuned model is used as the output of APRS. The raw optimization objective of pruning can be expressed as

$$\underset{\gamma_1,\gamma_2,\ldots,\gamma_L}{\arg\max}\ acc\left(M(\gamma_1,\gamma_2,...,\gamma_L;T_{train});T_{test}\right), \tag{1}$$

where γ_L is the pruning rate applied to the L-th layer, M is the pre-trained model with L layers. T_{train} and T_{test} represent the training dataset and test dataset respectively, acc is the model accuracy. The optimization goal is to achieve the highest accuracy acc on the test dataset. Given constraints such as targeted model size or FLOPs, a set of suitable pruning rates $\gamma_1, \gamma_2, ..., \gamma_L$ are automatically searched in APRS.

3.1 Notations

Assume a pre-trained CNN model consists of L convolutional layers, where $\mathbb{C}_l$ represents the l-th convolutional layer. The output channels (*i.e.*, feature maps) of the l-th layer, are denoted as $O_l = \left\{o_l^1, o_l^2, ..., o_l^{n_l}\right\} \quad \epsilon \quad \mathbb{R}^{n_l \times g \times h_l \times w_l}$, n_l represents the number of filters in $\mathbb{C}_l$, g is the size of input images. h_l and w_l are the height and width of the feature map, respectively. The parameter of the $\mathbb{C}_l$ layer can be described as a series of matries $W_{\mathbb{C}_l} = \left\{w_l^1, w_l^2, ..., w_l^{n_l}\right\} \quad \epsilon \quad \mathbb{R}^{n_l \times n_{l-1} \times a_l \times a_l}$, where the i-th filter of the l-th layer $w_l^i \quad \epsilon \quad \mathbb{R}^{n_{l-1} \times a_l \times a_l}$ can generate the i-th

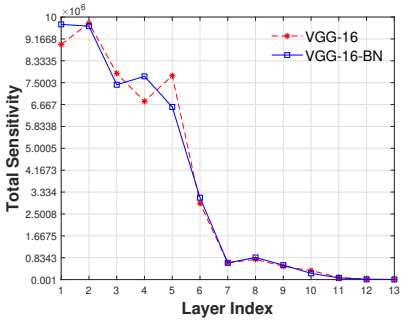


Fig. 4: Sensitivity of VGG-16 and its variant. In general, deeper convolutional layers have lower sensitivity.

channel o_l^i. a_l refers to the kernel size. In our APRS, for the l-th layer, We get pruned output feature maps $O_l^1, O_l^2, ..., O_l^{n_l}$, where $O_l^1 = \{0, o_l^2, ..., o_l^{n_l}\}$. For convenience, we assign zero to the activation value corresponding to the pruned channel. We compute the difference $\mathbb{Q}_l^i$ between the original output feature map and the output feature map after pruning the i-th filter. So we get the difference set: $\mathbb{Q}_l = \{\mathbb{Q}_l^1, \mathbb{Q}_l^2, ..., \mathbb{Q}_l^{n_l}\}$ where $\mathbb{Q}_l^i$ is the distance after removing the i-th filter o_l^i of the l-th layer. We prune filters with sequential traversal, so the number of distances is equal to the number of filters. In this paper, we call $\mathbb{Q}_l^i$ as sensitivity and use the Wasserstein distance to calculate it: $\mathbb{Q}_l^i = was\{O_l, O_l^i\}$.

3.2 Sensitivity Measurement

Wasserstein distance is used to measure the difference between two distributions, specifically, which describes the minimum cost required to transform from one distribution to the other.

$$\begin{aligned} was\left(O_1, O_2\right) &= \inf_{\gamma(x,y)\in\mathbb{Z}} \sum_{x,y} \|x-y\| \\ &= \inf_{\gamma\sim\mathbb{Z}(O_1,O_2)} E_{(x,y)\sim\gamma}\|x-y\|, \\ s.t.\ x \in O_1, y &\in O_2 \end{aligned} \tag{2}$$

Intuitively, $E_{(x,y)\sim\gamma}\|x-y\|$ can be understood as the consumption required to move O_1 to O_2 under the path planning of γ. $\mathbb{Z}$ is the set of all possible joint distributions that combine the O_1 and O_2 distributions. The Wasserstein distance is the minimum cost under the optimal path planning. Compared with using Kullback–Leibler(KL) divergence as the post-quantization sensitivity [24] in parameter quantization, the advantage of Wesserstein distance is that even if the support sets of the two distributions do not overlap or overlap very little, they can still reflect the distance of the two distributions. Since the activation values are relatively concentrated, to a certain extent, it is more friendly to use Wesserstein distance to calculate the distance.

So the importance of a single filter is obtained, so that we can flexibly choose the filters we need to delete according to the objective requirements, such as pre-trained model based on VGG-16 with CIFAR-10, if we need to get a 2.6M model, we can find a set of the most suitable filter pruning schemes. In this case, the accuracy of the model is up to 94.5%, which is theoretically the best. Assuming that the sensitivity measured by Wasserstein is:

$$\begin{aligned} \mathbb{Q}_i\left(\gamma_i\right) &= \frac{1}{N} \sum_{j=1}^{N_{dist}} was\left(M\left(n_i, O_i\right), M^{'}\left(n_i * \gamma_i, O_i^{'}\right)\right) \\ s.t.\ & n_i * \gamma_i \leqslant \mathbb{I}_{size} \end{aligned} \tag{3}$$

where n_i is the number of filters for the i-th layer, and $\mathbb{I}_{size}$ refers to the limit of model size. The output of the sub-net is close to the output of the pre-trained model when $\mathbb{Q}_i(\gamma_i)$ is small, so it is relatively insensitive when pruning $n_i * \gamma_i$ filters for i-th layer, and vice versa.

3.3 Pareto Optimization

Pareto optimization is a scientific research method that has recently been used in mixed-precision quantization [24,26] to find the most suitable solution. The implementation principle of Pareto optimization is as follows. Given target model size $\mathbb{I}_{size}$, we measure the total sensitivity of pruned model for each combination of pruning rate. We choose the pruning rate corresponding to the smallest disturbance in Fig. 5. In detail, we solve the following optimization problem:

$$\arg\min_{\gamma_1,\gamma_2,\ldots,\gamma_L} \mathbb{Q}_{sum} = \sum_{i=1}^{L} \sum_{n=1}^{n_i} \mathbb{Q}_i\left(\gamma_i\right) \ \ s.t. \ \sum_{i=1}^{L} n_i * \gamma_i \leqslant \mathbb{I}_{size} \tag{4}$$

Note that we verify the sensitivity of different filters independently (since pruning multiple channels and pruning these channels separately are very close in sensitivity). Using dynamic programming methods, we can get optimal solutions in different sets. Benefiting from the Pareto optimization algorithm, APRS can fulfill a batter balance between model size and sensitivity. Given a target model size $\mathbb{I}_{size}$, the pruning combination candidate with the smallest overall sensitivity is search as the optimal pruning scheme.

Suppose our final solution space is $\mathbb{P} = \prod_{l=1}^{L} n_l$. Here, the nominal goal is to maximize the sub-nets performance on the validation set by solving the following optimization problem: $\arg\min_{\gamma_1,\gamma_2,\ldots,\gamma_L} \mathbb{Q}_{sum}$. The computational cost of the Pareto plane is reduced from $\mathrm{O}(N\Pi_{i=1}^{L} n_i)$ to $\mathrm{O}(N\sum_{i=1}^{L} n_i)$ by calculating the sensitivity of each layer separately. That is to say, we compute N total sensitivities $\mathbb{Q}_{sum}$ for all N different pruning choices, while the computational complexity of each sub-nets is $\mathrm{O}(\sum_{i=1}^{L} n_i)$. The dynamic programming algorithm traverses sensitivity caused by each filter and obtains all the pruned sub-nets at the filter level, so theoretically can find the best pruning rate configuration among the set. Our experiments show that the final subnet achieves state-of-the-art accuracy with a smaller amount of parameters compared to the original pretrained model.

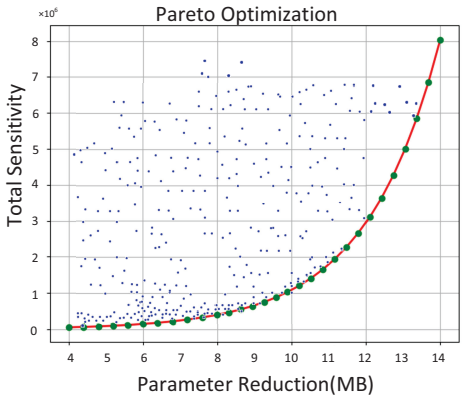


Fig. 5: The Pareto plane of VGG-16 on CIFAR-10. Each point shows a sub-structure vector setting. The x-axis is the parameter size of each configuration, and the y-axis is the total sensitivity. Then Pareto plane method chooses a sub-structure vector configuration with minimum perturbation given the model size.

APRS can also be used as a plug-and-play module, which can be combined with other filter pruning methods to bring a lot of improvement without changing the original algorithm. Further results may be found in supplementary material.

4 Experiments

4.1 Experimental Settings

Datasets and CNNs. In order to get a rigorous pruning result, we conducted experiments on the classic classification datasets CIFAR-10 [27] and ImageNet [70]. Mainstream CNN models was introduced into our ASPR experiments, including VGG-16 [31], Resnet-50/110 with residual module [28], and DenseNet-40 with a dense block [29], which is relatively simple and easy to implement in hardware. For each sub-net, we calculate the Wasserstein distance and use this as a theoretical guide for pruning rate search.

Evaluating Indicator. We adopt general evaluation indicator, *i.e.*, the number of weights and overall FLOPs (refers Float Points Operations) in the convolutional layer, to describe model size and floating-point operations of the sub-net. We adopt the variation of top-1 accuracy to compare model performance. To evaluate performance on specific tasks, we furnish top-1 and top-5 accuracy of pruned models and the pruning rate (denoted as PR) on CIFAR-10 and ImageNet.

Configurations. The implementation environment of APRS is pytorch [30]. The batch size, learning rate and momentum are set to 128, 0.01 and 0.9, respectively. We train on two NVIDIA Tesla P40 GPUs. It is worth noting that

Table 2: Comparison results on VGG-16 for the CIFAR-10 dataset. Acc(%)↓ refers Top-1 accuracy decrease.

Model	Top-1(%)	Top-5(%)	Acc(%)↓	Parameters	FLOPs
VGG-16	93.87	99.47	0	14.98M	313.73M
L1 [5]	93.40	99.40	0.47	5.4M	206.00M
SFP [54]	92.08	-	0.28	5.4M	113.23M
Zhao *et al.* [51]	93.18	99.12	0.63	3.92M	190.00M
GAL-0.05 [18]	92.03	98.95	1.84	3.36M	189.49M
SSS [50]	93.02	99.40	0.85	3.93M	183.22M
Hinge [53]	93.59	-	0.28	2.98M	191.16M
APRS(ours)	**94.73**	**99.61**	**-0.86**	**2.92M**	**109.09M**
GAL-0.1 [18]	90.73	-	3.14	2.67M	171.92M
HRank [10]	92.34	-	1.53	2.64M	108.61M
Guo *et al.* [71]	93.76	-	0.11	2.85M	-
APRS(ours)	**94.49**	**99.67**	**-0.65**	**2.64M**	**91.57M**
ABCPruner [22]	93.08	-	0.79	1.67M	82.81M
HRank [10]	91.23	-	2.64	1.78M	73.70M
HAP [52]	93.66	-	0.21	1.51M	93.18M
APRS(ours)	**94.21**	**99.57**	**-0.34**	**1.51M**	**87.23M**

APRS obtains substructure vectors based on the analysis results of one-off pruning all layers. But after getting the sub-structure vector, we employ progressive fine-tuning to restore the accuracy during the fine-tuning stage.

4.2 Results and Analysis

Results on CIFAR-10. We analyzed the performance of the mainstream network mentioned above on CIFAR-10. A variant of VGG [5] and DenseNet-40 with 3 dense blocks and 2 translation layers are used in our experiments, respectively.

Once the final sub-structure vector is determined, we immediately calculate the corresponding layer-by-layer pruning rates and guide the pruning to obtain the determined pruning model. The pruned model inherits the weights and is then fine-tuned. Take account of one-off pruning too many parameters may affect the network performance and fail to recover accuracy, we adopt a progressive fine-tune scheme, pruning a small number of filters, and fine-tuning the remaining network multiple times. In the fine-tune stage, we prune and fine-tune layer by layer to minimize the loss. We take the performance of the pre-trained model as the baseline and compare with other pruning methods. The experiment results show the accuracy of pruned sub-nets in proposed APRS outgo the original baseline.

VGG-16. VGG-16 only consists of plain layers, so we can easily calculate the sensitivity of each filter in each layer to guide pruning. We determine the number of filters in each layer, then divide by the total number of filters in

Table 3: Pruning results of ResNet-110 on CIFAR-10. Acc(%)↓ refers Top-1 accuracy decrease. ResNet-110 denotes the baseline in our APRS. PR refers the pruning rate.

Model	Top-1(%)	Top-5(%)	Acc(%)↓	Parameters(PR)	FLOPs(PR)
ResNet-110	93.53	99.81	-	1.72M(0.0%)	252.89M(0.0%)
L1 [5]	93.55	-	-0.02	1.68M(2.3%)	213.00M(15.8%)
GAL-0.05 [18]	92.55	-	0.98	0.95M(44.7%)	130.20M(48.5%)
HRank [10]	94.23	-	-0.70	1.04M(39.5%)	148.70M(41.2%)
APRS(ours)	**94.39**	**99.82**	**-0.77**	**0.99M(42.4%)**	**139.27M(45.0%)**
ABCPruner [22]	93.58	-	-0.05	0.56M(67.4%)	89.87M(64.5%)
APRS(ours)	**93.97**	**99.29**	**-0.44**	**0.53M(69.2%)**	**87.86M(65.3%)**

the current layer to get the pruning rate. The sub-network with the smallest overall sensitivity is searched through Pareto front planning until the limit of $\mathbb{I}_{size}$ is reached, so as to obtain the most accurate combination of L-layer pruning rates. Table 2 shows the performance of different methods, including filter sorting methods such as L1, and several adaptive importance-based methods [50,18]. Compared with the state-of-the-art Hessian-Aware Pruning [52] recently, APRS has achieved a very big leap. Compared to [10], we achieved a 2.11% improvement in accuracy, reaching 94.49% top-1 accuracy. For intuitive comparison, we set the similar model target size. When the model size $\mathbb{I}_{size}$ is set as 2.64M, the top-1 accuracy of APRS is 3.76% higher than that of GAL-0.01 [18], 2.15% higher than that of HRank [10], and the FLOPs almost half as those of GAL-0.01.

ResNet-110. Table 3 is the comparison result with other methods on ResNet-110. With parameter reductions and FLOPs close to the GAL [18], APRS achieves excellent top-1 accuracy (94.39% *vs.* 92.55%), which is better than the baseline model. Compared with HRank [10], APRS has advantages in all aspects (45.0% *vs.* 41.2% in FLOPs reduction, 42.4% *vs.* 39.5% in parameters reduction, 94.39% *vs.* 94.23% in top-1 accuracy). Compared with another automatic search pruned model method [22], APRS also attains better top-1 accuracy (93.97% *vs.* 93.58%) with slightly smaller parameters and FLOPs. Therefore, the Wasserstein distance can effectively reflect the relative importance of filters. Theoretically, APRS can fully remove the redundancy of the network since all channels can hypothetically be combined greedily.

DenseNet-40. Table 4 summarizes the experimental results on DenseNet-40. We observed the potential of APRS in removing FLOPs. Despite Liu *et al.* achieve a pruning performance close to the baseline [6], they only reduces FLOPs by 38.2%. In contrast, APRS can eliminate 45.7% FLOPs while maintaining high performance. Compared with GAL [18] and HRank [10], it has better compression rate in terms of parameters and FLOPs.

Results on ImageNet. Table 5 reflects our performance for ResNet-50 on ImageNet to verify the proposed APRS. Generally, APRS outperforms other pruning methods in any aspects, including top-1 and top-5 accuracies, as well

Table 4: Comparison on pruning approaches using DenseNet-40 on CIFAR-10. PR represents the pruning rate.

Model	Top-1(%)	Top-5(%)	Parameters(PR)	FLOPs(PR)
DenseNet-40	94.75	99.84	1.04M	0.282B
Liu *et al.*-40% [6]	94.81	-	0.66M(36.5%)	0.190B(32.6%)
HRank [10]	93.87	-	0.66M(36.5%)	0.167B(40.8%)
GAL-0.01 [18]	94.29	-	0.67M(35.6%)	0.182B(35.5%)
APRS(ours)	**94.37**	**99.85**	**0.63M(39.4%)**	**0.153B(45.7%)**

Table 5: Pruning results of ResNet-50 on ImageNet. PR represents the pruning rate.

Method	Top-1(%)	Parameters(PR)	FLOPs(PR)
Baseline	76.15	25.50M(0.0%)	4.09B(0.0%)
CP [13]	72.30	-	2.73B(33.2%)
GAL-0.5 [18]	71.95	21.20M(16.8%)	2.33B(43.0%)
SSS [50]	74.18	18.60M(27.0%)	2.82B(31.0%)
He *et al.* [54]	74.61	-	2.38B(41.8%)
HRank [10]	74.98	16.15M(36.7%)	2.30B(43.7%)
MetaP [55]	72.27	15.64M(38.7%)	2.31M(43.5%)
APRS(ours)	**75.58**	**16.17M(35.4%)**	**2.29B(44.0%)**
HAP [52]	75.12	14.13M(53.74%)	2.70B(66.1%)
GDP [56]	71.19	-	1.88B(54.0%)
APRS(ours)	**75.35**	**14.69M(57.6%)**	**1.94B(52.6%)**

as FLOPs and parameters reduction. Specifically, our accuracy is improved by 3.31%(75.58% *vs.* 72.27%) in contrast to MetaP [55] while the parameter reduction and FLOPs are maintained. In detail, APRS removes 1.58× parameters and 1.79× FLOPs and the accuracy drops by only 0.57%, which is 0.6% better than HRank [10] under the same FLOPs and parameters, and outperforms the state-of-the-art HAP [52]. Furthermore, APRS greatly reduces model complexity compared to SSS [50] (44.0% vs. 31.0% for FLOPs). Summarily, more FLOPs can be removed in our APRS than state-of-the-art methods. Our method outperforms previous studies when removing nearly half of the parameters. Compared with HAP [52], we remove more parameters with almost the same performance.

Ablation Experiment. We conduct two ablation experiments using ResNet-50 on ImageNet to demonstrate the effectiveness of Wasserstein distance in APRS. In the first ablation experiment, we apply a random constant pruning rate and demonstrate that sensitivity-based pruning is effective. The results in Table 6 show that when the pruning rate of the parameters is more than 20% , the accuracy of random pruning lags far behind APRS (less than 5.24%). In the second ablation experiment we use the reverse order of what APRS recommends,

Table 6: Ablation studies of sensitivity indicators for ResNet-50 on the ImageNet dataset. Anti-APRS means pruning in the reverse order recommended by APRS, Random is achieved by randomly assigning channel sensitivity. ↓% means the percentage of the portion removed.

Method	Acc(%)	Parameters(↓%)	FLOPs(↓%)
ResNet-50	76.15	0.0	0.0
Random	70.68	21.6	30.42
Anti-APRS	69.47	22.8	27.53
APRS(ours)	**75.92**	**22.3**	**26.50**
Random	65.44	50.7	54.41
Anti-APRS	62.37	50.0	55.77
APRS(ours)	**74.72**	**52.9**	**57.21**

named Anti-APRS. It can be clearly observed that Anti-APRS performs poorly in all cases compared to APRS, with an accuracy of 62.37% at a pruning rate of 50% of the parameters. Random pruning is slightly better than Anti-APRS, while there is still a big gap than APRS. APRS reaches a higher top-1 accuracy, which proves that our proposed method in this paper is powerful. Table 6 confirms that using the overall Wasserstein ranking and considering the common influence of the sensitivity between the layers will bring greater benefits. The impact of each layer on the overall performance is limited in spite of every filter of the deep network can extract features. Therefore, the distribution of pruning force should be more precise and efficient. Table 6 shows that certain more representative filters can be selected to form more effective pruning networks by using APRS, resulting in higher overall performance. Further details may be found in supplementary material.

5 Conclusions

In previous AutoML based methods, we found that the redundancy between convolutional layers varies greatly and the combination of different pruning rates may lead to various accuracy with the same parameter reduction. To search for a suitable pruning rate, we introduce sensitivity analysis to evaluate the impact of filters in each layer. We can initially set the pruning rate of each layer based on sensitivity analysis, then the Pareto optimization algorithm was utilized to search for appropriate combination of pruning rates. We introduced the Sinkhorn algorithm to help us solve the Wasserstein distance iteratively. Finally, we conducted experiments on the mainstream networks VGG-16 and ResNet-110, which proved that using sensitivity to remove the redundancy of the pre-trained model has a good effect. For VGG-16 on CIFAR-10, the top-1 accuracy reached 94.49% with 82.4% of the parameters are removed. In comparison to HRank [10], we achieve up to 2.15% higher accuracy with fewer parameters and

FLOPs. Moreover, for ResNet-50 APRS achieves 75.35% top-1 accuracy (0.8% degradation) on ImageNet, after pruning 42.4% of the parameters.

References

1. Liu, Z., Sun, M., Zhou, T., Huang, G., Darrell, T.: Rethinking the Value of NetworkPruning. In: ICLR (2019)
2. G.Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. In: CVPR (2017)
3. Ta, M., V.Le, Q.: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In: International Conference on Machine Learning, PMLR (2019)
4. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both Weights and Connections for Efficient Neural Networks. In: NeurIPS (2015)
5. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning Filters for Efficient ConvNets. In: ICLR (2017)
6. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning Efficient Convolutional Networks Through Network Slimming. In: ICCV (2017)
7. Singh, P., Verma, V.K., Piyush Rai, V.P.N.: Play and Prune: Adaptive Filter Pruning for Deep Model Compression. In: IJCAI (2019)
8. He, Y., Han, S.: ADC: Automated Deep Compression and Acceleration with Reinforcement Learning. In: CVPR (2018)
9. Miguel Á. Carreira-Perpiñán, Yerlan Idelbayev.: "Learning-Compression" Algorithms for Neural Net Pruning. In: CVPR (2018)
10. Lin, M., Ji, R., Wang, Y., Zhang, Y., Zhang, B., Tian, Y., Shao, L.: HRank: Filter Pruning using High-Rank Feature Map. In: CVPR (2020)
11. Han, S., Mao, H., Dally, W.J.: Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In: ICLR (2016)
12. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet Classification with Deep Convolutional Neural Networks. In: NeurIPS (2012)
13. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: ICCV (2017)
14. Luo, J.H., Wu, J., Lin, W.: Thinet: A filter level pruning method for deep neural network compression. In: ICCV (2017)
15. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient inference. In: ICLR (2017)
16. He, Y., Liu, P., Wang, Z., Hu, Z., Yang, Y.: Filter Pruning via Geometric Median for Deep Convolutional Neural Networks Acceleration. In: CVPR (2019)
17. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S.: AMC: AutoML for model compression and acceleration on mobile devices. In: ECCV (2018)
18. Lin, S., Ji, R., Yan, C., Zhang, B., Cao, L., Ye, Q., Huang, F., Doermann, D.: Towards optimal structured cnn pruning via generative adversarial learning. In: ICCV (2019)
19. Lemaire, C., Achkar, A., Jodoin, P.M.: Structured pruning of neural networks with budget-aware regularization. In: CVPR (2019)
20. You, Z., Yan, K., Ye, J., Ma, M., Wang, P.: Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In: NeurIPS (2019)
21. Yu, R., Li, A., Chen, C.F., Lai, J.H., Morariu, V.I., Han, X., Gao, M., Lin, C.Y., Davis, L.S.: Nisp: Pruning networks using neuron importance score propagation. In: CVPR (2018)

22. Lin, M., Ji, R., Zhang, Y., Zhang, B., Wu, Y., Tian, Y.: Channel Pruning via Automatic Structure Search. In: IJCAI (2020)
23. Li, B., Wu, B., Su, J., Wang, G.: EagleEye: Fast Sub-net Evaluation for Efficient Neural Network Pruning. In: ECCV (2020)
24. Cai, Y., Yao, Z., Dong, Z., Gholami, A., Mahoney, M.W., Keutzer, K.: ZeroQ: A Novel Zero Shot Quantization Framework. In: CVPR (2020)
25. Mitsuno, K., Kurita, T.: Filter Pruning using Hierarchical Group Sparse Regularization for Deep Convolutional Neural Networks. In: ICPR (2020)
26. Dong, Z., Yao, Z., Cai, Y., Arfeen, D., Amir, G., Mahoney, M.W., Keutzer, K.: Hawq-v2: Hessian aware trace-weighted quantization of neural net-works. In: NeurIPS (2020)
27. Alex Krizhevsky, Geoffrey Hinton, e.a.: Learning multiple layers of features from tiny images (2009), technical report, Citeseer
28. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
29. Huang, G., Liu, Z., Maaten, L.V.D., Weinberger, K.Q.: Densely connected convolutional networks. In: CVPR (2017)
30. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch. In: NeurIPS (2017)
31. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: ICLR (2015)
32. Zhang, T., Ye, S., Feng, X., Ma, X., Wang, Y.: StructADMM: Achieving Ultra-high Efficiency in Structured Pruning for DNNs. In: IEEE Transactions on Neural Networks and Learning Systems. vol. 12, pp. 1–15 (2021)
33. Mingbao, L., Rongrong, J., Shaojie, L., Qixiang, Y., Yonghong, T., Jianzhuang, L., Qi, T.: Filter Sketch for Network Pruning. In: IEEE Computer Society. pp. 1–10 (2020)
34. Chang, J., Lu, Y., Xue, P., Xu, Y., Wei, Z.: ACP: Automatic Channel Pruning via Clustering and Swarm Intelligence Optimization for CNN. In: CVPR (2021)
35. Xu, P., Cao, J., Shang, F., Sun, W., Li, P.: Layer Pruning via Fusible Residual Convolutional Block for Deep Neural Networks. In: CVPR (2020)
36. Wanga, Z., Li, C., Wang, X.: Convolutional Neural Network Pruning with Structural Redundancy Reduction. In: CVPR (2021)
37. Zhu, F., Gong, R., Yu, F., Liu, X., Wang, Y., Li, Z., Yang, X., Yan, J.: Towards Unified INT8 Training for Convolutional Neural Network. In: CVPR (2019)
38. Qin, H., Gong, R., Liu, X., Bai, X., Song, J., Sebe, N.: Binary neural networks: A survey. In: PR (2020)
39. Qin, H., Gong, R., Liu, X., Shen, M., Wei, Z., Yu, F., Song, J.: Forward and Backward Information Retention for Accurate Binary Neural Networks. In: CVPR (2020)
40. Kravchik, E., Yang, F., Kisilev, P., Choukroun, Y.: Low-bit quantization of neural networks for efficient inference. In: ICCV (2019)
41. Yao, Z., Dong, Z., Zheng, Z., Gholami, A., Yu, J., Tan, E., Wang, L., Huang, Q., Wang, Y., Mahoney, M., Keutzer, K.: HAWQV3: Dyadic Neural Network Quantization. In: International Conference on Machine Learning, PMLR (2021)
42. Jin, Q., Ren, J., Woodford, O.J., Wang, J., Yuan, G., Wang, Y., Tulyakov, S.: Teachers Do More Than Teach: Compressing Image-to-Image Models. In: CVPR (2021)
43. Guo, J., Han, K., Wang, Y., Wu, H., Chen, X., Xu, C., Xu, C.: Distilling Object Detectors via Decoupled Features. In: CVPR (2021)

44. Gou, J., Yu, B., Maybank, S.J., Tao, D.: Knowledge distillation: A survey. In: CVPR. p. 1789–1819 (2021)
45. Mirzadeh, S., Farajtabar, M., Li, A., Levine, N., Ghasemzadeh, H.: Improved Knowledge Distillation via Teacher Assistant. In: AAAI (2020)
46. Tang, J., Shivanna, R., Zhao, Z., Lin, D., Singh, A., Chi, E.H., Jain, S.: Understanding and Improving Knowledge Distillation. In: CVPR (2020)
47. Sainath, T.N., Kingsbury, B., Sindhwani, V., Arisoy, E., Ramabhadran, B.: Low-rank matrix factorization for Deep Neural Network training with high-dimensional output targets. In: IEEE Computer Society (2013)
48. Swaminathan, S., Garg, D., Kannan, R., Andres, F.: Sparse low rank factorization for deep neural network compression. In: IEEE Computer Society. vol. 398, pp. 185–196 (2020)
49. Zhang, Y., Chuangsuwanich, E., Glass, J.: Extracting deep neural network bottleneck features using low-rank matrix factorization. In: IEEE Computer Society (2014)
50. Huang, Z., Wang, N.: Data-driven sparse structure selection for deep neural networks. In: ECCV. p. 304–320 (2018)
51. Zhao, C., Ni, B., Zhang, J., Zhao, Q., Zhang, W., , Tian, Q.: Variational convolutional neural network pruning. In: CVPR. p. 304–320 (2019)
52. Yu, S., Yao, Z., Gholami, A., Dong, Z., Kim, S., Mahoney, M.W., Keutzer, K.: Hessian-Aware Pruning and Optimal Neural Implant. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. pp. 3880–3891 (2022)
53. Li, Y., Gu, S., Mayer, C., Gool, L.V., Timofte, R.: Group Sparsity: The Hinge Between Filter Pruning and Decomposition for Network Compression. In: CVPR. pp. 8018–8027 (2020)
54. He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y.: Soft filter pruning for accelerating deep convolutional neural networks. In: IJCAI (2018)
55. 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: ICCV (2019)
56. Lin, S., Ji, R., , Y.L., Wu, Y., Huang, F., Zhang, B.: Accelerating convolutional networks via global & dynamic filter pruning. In: IJCAI (2018)
57. Kurtz, Mark, Kopinsky, Justin, Gelashvili, Rati, Matveev, Alexander, Carr, John, Goin, Michael, Leiserson, William, Moore, Sage, Nell, Bill, Shavit, Nir, Alistarh, Dan: Inducing and exploiting activation sparsity for fast inference on deep neural networks. In: International Conference on Machine Learning, PMLR. vol. 119, pp. 5533–5543. PMLR, Virtual (13–18 Jul 2020)
58. Zhang, Y., Lin, M., Lin, C.W., Chen, J., Wu, Y., Tian, Y., Ji, R.: Carrying out CNN Channel Pruning in a White Box. In: CVPR (2022)
59. Hu, H., Peng, R., Tai, Y.W., Tang, C.K.: Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures. In: ICCV (2019)
60. Chen, L.C., Zhu, Y., Papandreou, G., Schroff, F., Adam, H.: Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation. In: ECCV. pp. 801–818 (2018)
61. Redmon, J., Farhadi, A.: YOLOv3: An Incremental Improvement. In: CVPR (2018)
62. Xu, H., Yao, L., Zhang, W., Liang, X., Li, Z.: Auto-FPN: Automatic Network Architecture Adaptation for Object Detection Beyond Classification. In: ICCV (2019)
63. Mao, H., Han, S., Pool, J., Li, W., Liu, X., Wang, Y., Dally, W.J.: Exploring the Granularity of Sparsity in Convolutional Neural Networks. In: ICCV (2017)

64. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: Adv. Neural Inform. Process. Syst. (2016)
65. Liu, N., Ma, X., Xu, Z., Wang, Y., Tang, J., Ye, J.: AutoCompress: An Automatic DNN Structured Pruning Framework for Ultra-High Compression Rates. In: AAAI (2020)
66. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning Transferable Architectures for Scalable Image Recognition. In: CVPR (2018)
67. Yang, T.J., Howard, A., Chen, B., Zhang, X., Go, A., Sandler, M., Sze, V., Adam, H.: NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications. In: ECCV (2018)
68. Yu, J., Huang, T.: AutoSlim: Towards One-Shot Architecture Search for Channel Numbers. In: ICLR (2019)
69. Ye, S., Zhang, T., Zhang, K., Li, J., Xu, K., Yang, Y., Yu, F., Tang, J., Fardad, M., Liu, S., Chen, X., Lin, X., Wang, Y.: Progressive Weight Pruning of Deep Neural Networks using ADMM. In: CVPR (2018)
70. Russakovsky, O., Jia Deng, H.S., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. In: IJCV (2015)
71. Guo, Y., She, Y., Barbu, A.: Network Pruning via Annealing and Direct Sparsity Control. In: IJCNN (2021)