Design Studies and Intelligence Engineering
L.C. Jain et al. (Eds.)

doi:10.3233/FAIA220017

Compact Deep Neural Networks via Soft and Smooth Filter Pruning

Zonghui Fu[a], Xiaodong Wang[a], Wei Li[a] and Zhiqiang Zeng[a,1]
[a]*College of Computer and Information Engineering, Xiamen University of Technology, Xiamen, China*

Abstract. Filters pruning methods are widely used to accelerate the inference process of Deep Neural Networks (DNNs). Among them, Soft Filter Pruning (SFP) has achieved increasing attention due to its compatibility and flexibility. However,conventional SFP directly sets the pruned filters to zero during training phase, discarding the training information of the pruned filter completely. In order to solve the above problem, this paper proposed a soft and smooth pruning method to retain the training information among pruned filters. Instead of the zeroing the pruned filters, our method imposes a filter weakening strategy, which gradually forces the pruned filters to zeros. Such a gradual pruning framework will give the pruned model more chances to recover the lost information and boost the pruning performance. To verify the the actual effectiveness of our proposed method, we conduct several experimental results on one dataset of metal surface images captured in a controlled industrial environment, *i.e.*,KSDD, using Resnet-20 with various pruning rates. Experimental results show that our filter weakening strategy consistently achieves superior performance over the compared methods, especially when a large amount of filters is pruned. By pruning 60% of filters, our method only drops 2.52% on Accuracy.

Keywords. Filter pruning, ResNet, CNN, KSDD

1. Introduction

Recently, deep learning technology has developed rapidly, and various network structures have continuously emerged. Among them, deep learning networks represented by Convolutional Neural Networks (CNNs), *e.g.*, AlexNet [1], VggNet [2] , ResNet [3], have received widespread attention and continue to improve the accuracy of pattern recognition in various application fields. Recent studies demonstrate that building deep network structures is conducive to high performance. However, these deep networks usually company with heavy computation burdens due to huge numbers of parameters and their high memory usage [4]. In this situation, despite the excellent learning capabilities of these deep convolutional neural networks, they are difficult to apply directly to hardware platforms with limited computing resources. For example, in the terminal field, the popularization of applications posed significant challenges for deep learning technology. In order to further improve the practicality of deep learning networks,researchers have tried to compact deep learning algorithms for embedded platforms, such as mobile devices and

[1] Correspoding Author: Professor with the College of Computer and Information Engineering, Xiamen University of Technology; E-mail: lbxzzq@163.com

terminal devices [5,6]. According to previous studies [4–7], The underlying intention for compacting deep neural networks is that these networks often contain many redundant neurons, which can contribute a little to the learning ability of the networks. Recently, there are lots of studies pay attention to compacting the deep neural networks [8–11]. Among them, filter pruning methods aim to evaluate and remove the entire redundant filters, which are compatible with the hardware platforms, have recently gained increasing interest [8].

There are mainly two types of existing filter pruning methods, *i.e.*, Hard Filter Pruning (HFP) method [7–9] and Soft Filter Pruning (SFP) method [11] . The difference between HFP and SFP can be seen in Figure 1. HFP means that the convolution kernel that needs to be pruned is directly removed from the network when each convolution kernel pruning operation is performed, which may heavily hurt the quality of the model [8–10]. SFP allows filters that have been pruned to be updated while training. Thus, the network capacity can be restored from the pruned network, obtaining higher accuracy than HFP [11,12]. However, SFP directly sets the pruned filters to zeros during training phase, which will cause a serious decrease in the accuracy of the pruning model in the case of a large pruning rate. To retain the ability of the pruned filters that have been trained, we propose a weakening factor to improve the flaws of the soft pruning method . Concretely, our method use a gradual decay strategy to remove the filters that will smoothly converge to zero, where the training information in these filters can be better remain [13].

We mainly highlight the following contributions:

- We propose a soft and smooth filter pruning framework to accelerat the deep convolutional neural networks, which forces the redundant filters gradually converge to zeros and better prevents the losses of training information.
- We introduce a flexible filter weaken strategy, which can smoothly alternate among different distributions of filter weaken factor.
- We test our proposed method on one metal surface defects dataset, *i.e.,* KSDD,using Resnet-20 with various pruning rates.The experimental results demonstrate that our method is effective.

2. Related work

Deep learning network model compression focuses on reducing the complexity and saving storage space, so as to accelerate the inference of the model . According to previous studies, current deep neural network compression mainly includes low-rank decomposition, quantization, and network pruning. Low-rank decomposition usually depends on the matrix decomposition method ,which can save calculation costs. However, low-rank decomposition methods suffer from the problem of non-convergence [14].Compressing neural networks with using quantization can be regarded as a data compression study. For example, Gong *et al.* [15] used the k-means algorithm to compress parameters by quantizing the weight matrix. Another group of network accelerating is network pruning. Generally, the network pruning technology could reduce the computational consumption of deep neural networks by removing unimportant filters. The pruned network does not depend on special hardware for deploying to run on small or mobile devices.This paper mainly focuses on network pruning.

The study of network pruning has a long story. In the late 1980s and early 1990s,

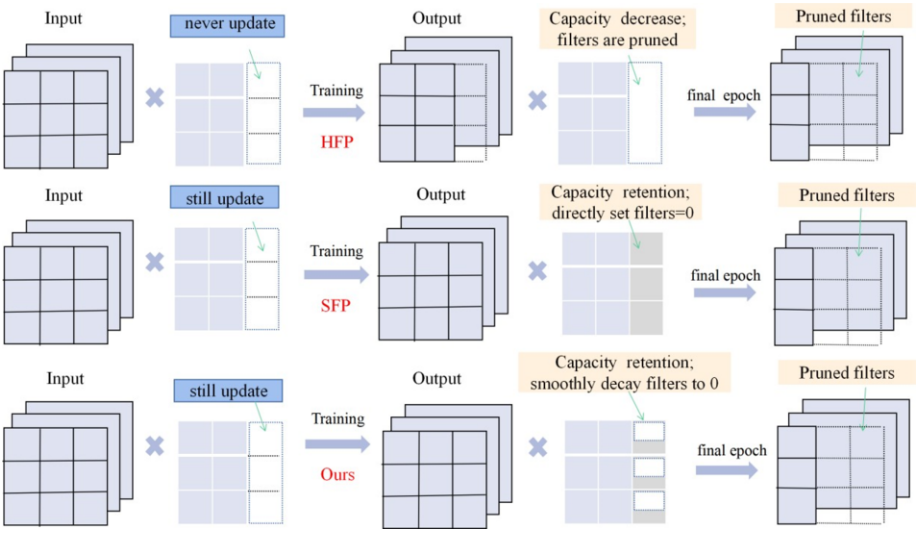

Figure 1. Comparison of hard filter pruning (HFP) and soft filter pruning (SFP). The pruned filters are marked as the dashed boxes. In HFP, the pruned filters cannot be updated in the following training process. Therefore, HFP may easily reduce the capacity of the model and damage the performance. On the contrary, SFP keeps the pruned filters and allows them to be updated afterwards. Therefore, the model capacity is restored from the pruned model, thereby improving accuracy.

the pruning of neural networks has already been studied. For example, S. J. Hanson proposed a pruning method based on amplitude, applying weight attenuation related to its absolute value to each hidden unit in the network to minimize the number of hidden units [16]. Another example is the OBD [17] proposed by Y. LeCun in the early 1990s, based on the second derivative of the loss function with respect to the weight (the Hessian matrix for the weight vector) to measure the importance of the weight in the network. How- ever, because of the general environment at that time, neural networks were not a partic- ularly mainstream branch of machine learning, so there was not a large number of stud- ies for network pruning afterward. Despite that, their definition of network compression problems and problem-solving ideas were excellent. In recent years, many works have made profound impacts on network compression. By 2012, most of the researchers focus on improving the accuracy of neural networks and continuously deepen the weighting network to improve accuracy. For example, the accuracy of the ImageNet dataset has reached a new high every year [18–20]. During 2015-2016, a series of work on model compression of deep neural networks was published in this field [21–23]. For example, Han *et al.* [22] proposed to accelerate the classic networks, *i.e.*, AlexNet and VGG, com- bining pruning, quantization, and Huffman coding, and other methods to compress the network. Besides, they also design an iterative method to compensate for the accuracy loss caused by pruning. After this work, researchers begin to realize that there indeed exists lots of redundant parameters in DNN. In the following years, the field of model compression has become more plentiful. For example, in 2018,Wang *et al.* [24] proposed the subspace clustering explore the relevant information in the filter and feature map, so as to simplify and compress the parameters. Nevertheless, most of the existing methods are focusing on weight pruning. He *et al.* [11] proposed the soft pruning method to per- form network be updated during the training process. In this way, the model capacity can be recovered

from the pruned model, which solves the problem of accuracy recovery and obtains higher accuracy than the previous methods.

As far as we know, SFP sets the pruned filter to zeros during training and updates model capacity during the next training epoch. However, during the first few trimming periods of SFP, the accuracy of the test set was significantly reduced. To make better use of the trained pruning filter, this paper proposed a soft and smooth filter pruning method, which uses a monotonically decreasing parameter strategy to attenuate the pruning weight. Since our method works in a soft pruning manner, it is compatible with the training procedure of the neural networks without additional fine-tuning costs.

3. Methodology

Firstly, we introduce some symbols and annotations in this section. For the convenience of description, the entire network containing $\mathbf{L}$ convolutional layers is defined as $\mathbf{W} = \{\mathbf{W}^1, \mathbf{W}^2, \ldots, \mathbf{W}^{\mathbf{L}}\}$, with the weight matrix of the i-th layer denoted as $\mathbf{W}^i \in \mathbb{R}^{\mathbf{K}_{i+1} \times \mathbf{K}_i} \times \mathbf{N} \times \mathbf{N}$, where $\mathbf{K}_i$ is the number of input channels, $\mathbf{K}_{i+1}$ is the number of output channels, and $\mathbf{N}$ is the convolutional kernel size. In addition, for the i-th convolutional layer, the input feature map $\mathbf{I}_i$ and the output feature map $\mathbf{S}_i$ are denoted as $\mathbf{K}_i \times \mathbf{H}_i \times \mathbf{W}_i$ and $\mathbf{K}_{i+1} \times \mathbf{H}_{i+1} \times \mathbf{W}_{i+1}$, respectively. The calculation formula for the convolution operation of the i-th layer can be denotes as:

$$\mathbf{S}_{i,j} = \mathbf{W}_{i,j} * \mathbf{I}_i, \text{ for } 1 \le j \le \mathbf{K}_{i+1}, \tag{1}$$

where $\mathbf{S}_{i,j} \in \mathbb{R}^{\mathbf{H}_{i+1} \times \mathbf{W}_{i+1}}$ denotes the j-th output channel of the i-th layer, and $\mathbf{W}_{i,j} \in \mathbb{R}^{\mathbf{K}_i \times \mathbf{N} \times \mathbf{N}}$ means the j-th filter of the i-th layer.

In the filter pruning procedure, the corresponding feature map will be removed when the filter is pruned. Therefore, it can decrease the memory footprint consumption as well as the computing costs of the model remarkably. Let us assume that the filter pruning rate in the i-th layer is denoted $\mathbf{F}_i$. In this way, the number of filters will be removed from $\mathbf{K}_{i+1}$ to $\mathbf{K}_{i+1}(1-\mathbf{F}_i)$ in the i-th layer, which is reduced by $\mathbf{K}_{i+1} \times \mathbf{F}_i$, and the size of the pruned output feature map $\mathbf{S}_{i',j}$ can be reduced from $\mathbf{K}_{i+1} \times \mathbf{H}_{i+1} \times \mathbf{W}_{i+1}$ to $\mathbf{K}_{i+1}(1-\mathbf{F}_i) \times \mathbf{H}_{i+1} \times \mathbf{W}_{i+1}$. Therefore, we can reduce the output size of the i-th layer, which is the input size of the $i+1$-th layer, to obtain a higher speedup. According to SFP , the weights that are pruned in the i-th layer are directly set to zeros, which can be written as

$$\mathbf{S}_{i,j} = \mathbf{W}_{i,j} \odot \mathbf{G}_{i,j}, \text{ for } 1 \le j \le \mathbf{K}_{i+1}, \tag{2}$$

where $\mathbf{G}_{i,j}$ is a indicator matrix which is the same shape as $\mathbf{W}_{i,j}$, denoting whether the j-th filter of the i-th layer is pruned . In addition, the symbol $\odot$ is the Hadamard product. To be more specific, $\mathbf{G}_{i,j} = 0$ if the filter $\mathbf{W}_{i,j}$ is pruned. Otherwise, $\mathbf{G}_{i,j} = 1$ indicates that $\mathbf{W}_{i,j}$ is retained.

To make better use of the training information inside those pruned weights, we assume that the pruning methods should not directly drop the weights of the pruned filters, that is, setting them to zeros. In contrast, we hope that there are some useful information among the pruned filters. In other words, the pruning methods should give the model more chances to transfer these useful information to the retrained filters. To achieve this goal, following the previous works [11–13], we propose to add a weakening factor a into Eq.(2), which can be equivalently rewritten as follows:

$$\mathbf{S}_{i,j} = \mathbf{W}_{i,j} \odot \mathbf{G}_{i,j} + \alpha \mathbf{W}_{i,j}(1 - \mathbf{G}_{i,j}), \text{ for } 1 \le j \le \mathbf{K}_{i+1}, \tag{3}$$

Obviously, when we set α=0, then Eq.(3) will reduce to the conventional SFP method. Similar to SFP, we prune models while training. Concretely, we first train the model for one epoch, then we prune the model by Eq.(3). After each round of pruning, the filters in the model are sorted from small to large in importance. For the i-th convolutional layer, we select the first $\mathbf{F}_i$ filters as the pruned filters and apply the weakening factor a to them. Generally, we set $\alpha_0 \in [0,1]$. In this case, the information among the pruned filters is not dropped completely, which can contribute to alleviate the decrease in accuracy caused by pruning filter .

It is worth mentioning that it is not safe to remove the pruned filters when their weight values are not zeros. Thus, we propose to gradually reduce a with the increase of pruned rounds. Formally, suppose the initial value of α is α_0, where $a_0 \in [0,1]$. We hope that α_0 should converge to zero (or close to zero) at the end of the pruning and training phase. Let α_n be the value of α_0 after n pruning rounds. We propose the following function on a_0 to encode a_n as follows:

$$\alpha_{\mathrm{n}} = \frac{a_{\mathrm{n}}}{1+e^{\beta \times \left(\frac{n}{n_{max}} - 0.5\right)}}, \text{ for } 1 \le n \le n_{\max}, \tag{4}$$

where n_{max} is the max value of training epochs, β is the coefficient which controls the reducing speed of α_0.

From Eq.(4), we can see that if a smaller β is provided , Eq.(4) tends to be a linear function of n. In contrast, if we provide a larger β , Eq.(4) works more like a reverse sigmoid function of n. In this paper, we perform the "training and pruning" process using Eq.(4) until all the values of pruned filter weights are converged to zeros, or we reach the maximum pruning rounds. Finally, we remove all the pruned filters to obtain the compact model.

4. Soft and Sooth Filter Pruning

We illustrate our method in Figure 2. From Figure 2, it can be seen that our method generally contains three steps. Firstly, in the n-th training epoch, our method evaluates the importance of filters using some metrics, such as l_2-norm (used in this paper) and l_1-norm, to rank the filters according to their importance. Then, the redundant filters or unimportant filters are selected and pruned by reducing their weight values. After that, we retrain the pruned model for one epoch to recover the lost information due to pruning filters. This "evaluating, pruning, and retraining" procedure is performed until all the

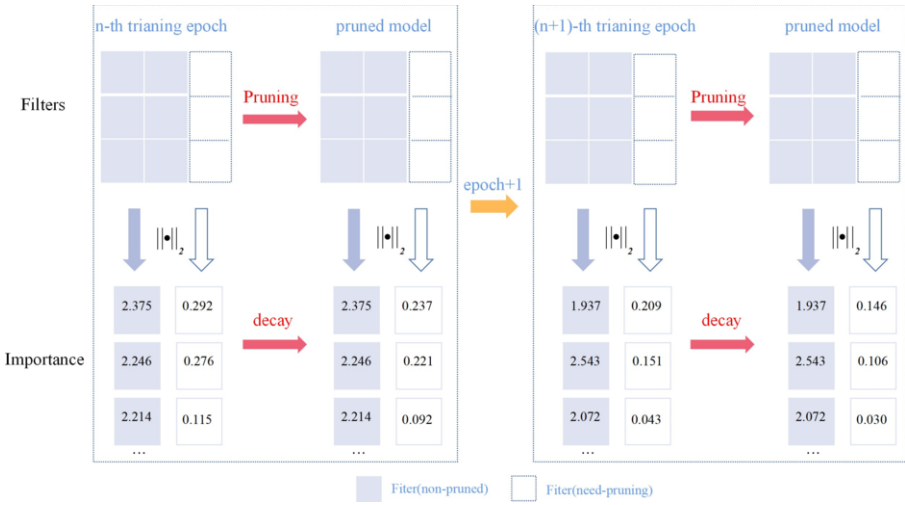

Figure 2. Overview of our method. After each training epoch, we evaluate and sort the filters according the their importance (*i.e.*, the l_2-norm in this paper). The selected redundant filters (white square) are chosen to be smoothly pruned. Then, we retrain the pruned model for one more epoch, letting it recover itself. After that, we reselect and prune the redundant filters. This "training and pruning" process is repeated until all the selected filters converges to zeros.

pruned filters are zeros. Finally, we can safely remove these redundant filters with zero values to obtain the compact model.

It should be noting that our method illustrated in Figure 2 is quite different from SFP. Concretely, after selecting the redundant filters, SFP directly set them to zeros. Although SFP does not removes these redundant filters from the pruned model, hoping the pruned filters to be recovered in the following procedure. Nevertheless, such a tremendous changing on the values of filters may heavily hurt the performance of the model. In contrast, our method gradually attenuates the filter after pruning, which can greatly avoid the sharp decline in accuracy caused by pruning . As shown in Eq.(3) and Eq.(4), the initial value of α is between 0 and 1 (in this paper we set it to 1). In other words, at the beginning of the training phase, our method will keep almost all the trained information in the pruned filter. Then, the selected redundant filters will be pruned smoothly by decaying the corresponding filter weight via α. The model status may slightly changed after pruning, giving the pruned model more chances to recover itself and achieve better performance. In the next training epoch, α is also decayed under the constraints of the decay strategy in Eq.(4). As the process continues, α will gradually decay to 0, forcing the pruned filters to zeros, which can be safely deleted. From this point of view, we can treat our method as a smooth version of the SFP.

5. Experiments

5.1. Experimental setting

Dataset: A dataset of images captured in a controlled industrial environment in a real case, *i.e.*, KSDD, is selected to evaluate the proposed method. KSDD is a dataset of metal surface defects, which has a total of 399 pictures of size 500 × 1240 pixels or 500

Algorithm 1 Soft and Sooth Filter Pruning

Require: training set $\mathbf{D}$, pruning rate $\mathbf{F}_i$, initial weakening factor α_0 of α, the user defined parameter β, the model with parameters $\mathbf{W}$
Ensure: The pruned model and its optimal parameters $\mathbf{W}^*$

```
Initialize the model parameter W_0
for n = 1; n<n_max; n++ do
    Train model parameters W(n+1) based on data set D and W_n
    Decrease weight weakening factor α_n using Eq.(4)
    for i = 1; i<L; i++ do
        Compute the l_2-norm of each filter
        Select the first K_{i+1}F_i filters with smaller l_2-norm values
        Decrease the parameter scales of selected filters with α
    end for
    Get the softly pruned model parameters W_{n+1}
end for
return Obtain the compact model with final parameters W*
```

× 1270 pixels in two classes. The dataset consists of 52 images with visible defects and 347 images without any defects. In the training process of this article, all input images are uniformly resized to 206 x 704 pixels. Following [25] we split the dataset into 264 trainning pictures and 135 testing pictures.

network: In this paper, we will revolve about pruning a challenging ResNet-type network, *i.e.*, ResNet-20. We select this networks for its excellent performance and wide range applications.

Settings: We use the deep learning framework, *i.e.*, Pytorch, and conduct all experiments on the NVIDIA Ge Force RTX 2080 Ti GPU. In our experiments, the model we use is trained from scratch and the network is trained for 100 epochs with batch size of 1. We repeat the experiment five times and report the average results along with the standard deviation. Following the previous work [11], to reduce the complexity of the determination of pruning rate for each convolutional layer, we set the same pruning rate $\mathbf{F}_i$ for each convolutional layer, which is also called structure pruning. To testify the effectiveness of our filter pruning, we conduct experiments with several different pruning rates for comparison. We impose the stochastic gradient descent (SGD) as the optimizer while training. To avoid the oscillating problem, three milestones, *i.e.*, 30 ,60 and 80, are deployed with the learning rate decay as 0.2. Concretely, the training process will adjust the learning rate as the number of training epoch increases. In the beginning, a learning rate of 0.01 is applied to our model. The learning rate is set to 0.002 from the 30-th epoch to the 60-th epoch, and the learning rate is 0.0004 from the 60-th epoch to the 80-th epoch. After the 80-th epoch, the model is relatively stable and the learning rate is set to 0.00008. The value of weight decay and momentum are respectively set to 0.0005 and 0.9. Resizing and Horizontal flipping are applied for data augmentation. Our method contains one hyperparameter, *i.e.*, β. By default, we set $\beta = 30$.

5.2. Experimental results

Results. As can be seen from Table 1 ,we summarize the pruning results of our method and SFP on KSDD dataset , where "Accu. Drop" is the accuracy rate of the baseline model (the model without pruning) minus the accuracy rate of the pruning model. The smaller the number of "Accu. Drop", the better the acceleration effect. It can be concluded from the table that compared with other filter pruning technologies with different pruning rates, our method shows superior performance than SFP on KSDD.

We separately compare the results of our method and SFP on KSDD with the change of pruning rate in Figure 3. From Figure 3, it is obvious that our method outperforms SFP on Resnet-20, especially when the pruning rate is larger than 20%, the accuracy difference between the two methods is obvious. When the pruning rate increases to 60%, our method is significantly better than SFP.

Table 1. Comparison results of pruning ResNet-20 on KSDD.

Network	Method	Baseline Accu. (%)	Pruned Accu. (%)	Accu.Drop(%)	FLOPs	Pruned
Resnet-20	SFP(20%)	94.67±1.28	93.18±1.27	1.49	2.87E+07	29.3
	OURS(20%)	94.67±1.28	93.48±1.09	1.19	2.87E+07	29.3
	SFP(40%)	94.67±1.28	92.00±1.51	2.67	1.87E+07	54.0
	OURS(40%)	94.67±1.28	93.18±1.78	1.49	1.87E+07	54.0

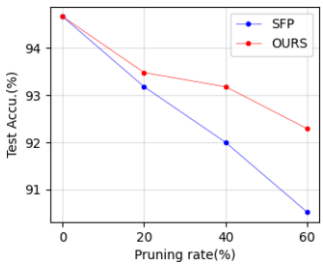

Figure 3. Comparison of test accuracy trends of pruning ResNet-20 on KSDD by SFP and OURS with different pruning rates.

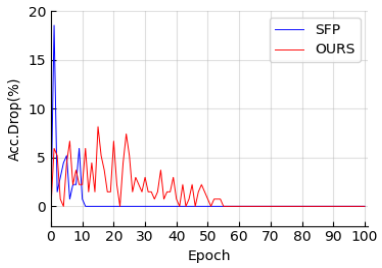

Figure 4. Comparison of accuracy drop of pruning ResNet-20 on KSDD by SFP and OURS as the training epoch increasing.

Convergence analysis. In Figure 4,we compare the comparison of accuracy drop of ResNet-20 on KSDD by SFP and OURS in the training epoch when the pruning rate is 30%."Accu.Drop" refers to the difference between the accuracy of Top-1 before and after pruning. The smaller the value, the less obvious the decrease in accuracy caused by pruning. From the experimental results, we find that the test accuracy drop of SFP converges to 0 at a faster rate, while OURS maintains a slower convergence rate. The reason may be that OURS softens the SFP in terms of the weight decay, making it needs more epochs are needed to converge.

5.3. Ablation Study

Different β values. We observe the relationship between β and α in Figure 5 and the relationship between β and the number of zero in Figure 6. We can see from Figure 5 that as β increases, the curve of the attenuation factor tends to be steep, and the range of α is infinitely close to [0,1]. The smaller the minimum value of α, the greater the

confidence that redundant filters's weights are set to 0 at the end of training. In Figure 6, we show the number of zero-weight filters changes of ResNet-20 on KSDD with different β values and pruning rates. When the pruning rate reaches at 40%, we notice that the greater the β, the larger the number of zero weights. Through this experimental results, it is obvious that as β increases, the effect of pruning is enhanced and the number of redundant parameters of the network will decrease.

Various pruning rates. In this paper the filter weaken strategy belongs to the weight decay. According to Algorithm 1, after each training phrase, the filters in the model are sorted according to their importance and the selected filters are pruned by attenuating the corresponding filter weight with α. To further clarify the performance of our method, we show the comparison of accuracy with various pruning rates on ResNet-20 by SFP and OURS in Figure 7. It can be found that as the pruning rate increasing, the advantages of our method over SFP are magnified.

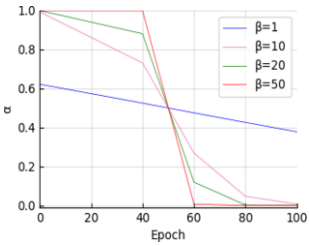

Figure 5. The decay trend of α with different β in the trainng phrase.

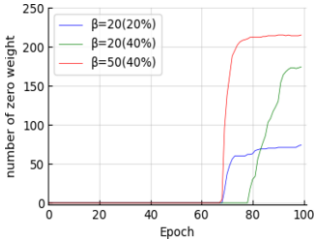

Figure 6. The number of zero-weight filters changes with different β and pruning rates (in brackets) in the training phrase.

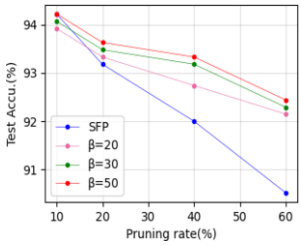

Figure 7. Comparison of test accuracies of different pruning rates for ResNet-20 on KSDD between SFP and ours (β = 20, 30, 50) with the decay strategy.

6. Conclusion

In this paper, we propose a flexible filter weaken strategy, which softens the pruning operation of SFP. Compared with SFP, our method uses a weight decay strategy that gradually decays the weight of filters to zero in the pruning stage to remove the redundant filters. Our method performs well on ResNet-20 with different pruning rates on the KSDD dataset. In addition, we have studied the internal mechanism of our method and noticed that our method is pursuing better results without significantly increase of model convergence speed.

Acknowledgement

This paper was supported by National Natural Science Foundation of China (Grant No. 61871464), National Natural Science Foundation of Fujian Province (Grant Nos. 2020J01266,2021J011186), the "Climbing" Program of XMUT (Grant No. XPDKT20031), Scientific Research Fund of Fujian Provincial Education Department (Grant No.JAT200486), Program of XMUT for high-Level talents introduction plan (Grant No. YKJ19003R).

References

[1] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, Advances in neural information processing systems 25 (2012) 1097–1105.
[2] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556 (2014).
[3] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
[4] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, C. Zhang, Learning efficient convolutional networks through network slimming, in: Proceedings of the IEEE international conference on computer vision, 2017, pp. 2736–2744.
[5] Y. D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, D. Shin, Compression of deep convolutional neural networks for fast and low power mobile applications, Computer Science 71 (2) (2015) 576–584.
[6] V. Vanhoucke, A. Senior, M. Z. Mao, Improving the speed of neural networks on cpus, in: Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011, 2011.
[7] H. Li, A. Kadav, I. Durdanovic, H. Samet, H. P. Graf, Pruning filters for efficient convnets, arXiv preprint arXiv:1608.08710 (2016).
[8] S. Han, H. Mao, W. J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, Fiber 56 (4) (2015) 3–7.
[9] X. Zhang, Y. He, S. Jian, Channel pruning for accelerating very deep neural networks, in: IEEE International Conference on Computer Vision, 2017.
[10] J.-H. Luo, J. Wu, W. Lin, Thinet: A filter level pruning method for deep neural network compression, in: Proceedings of the IEEE international conference on computer vision, 2017, pp. 5058–5066.
[11] Y. He, G. Kang, X. Dong, Y. Fu, Y. Yang, Soft filter pruning for accelerating deep convolutional neural networks, arXiv preprint arXiv:1808.06866 (2018).
[12] Y. Guo, A. Yao, Y. Chen, Dynamic network surgery for efficient dnns, arXiv preprint arXiv:1608.04493 (2016).
[13] X. Dong, J. Huang, Y. Yang, S. Yan, More is less: A more complicated network with less inference complexity, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 5840–5848.
[14] M. Jaderberg, A. Vedaldi, A. Zisserman, Speeding up convolutional neural networks with low rank expansions, Computer ence 4 (4) (2014) XIII.
[15] Y. Gong, L. Liu, M. Yang, L. Bourdev, Compressing deep convolutional networks using vector quantization, arXiv preprint arXiv:1412.6115 (2014).
[16] S. Hanson, L. Pratt, Comparing biases for minimal network construction with back-propagation, Advances in neural information processing systems 1 (1988) 177–185.
[17] Y. Lecun, Optimal brain damage, Neural Information Proceeding Systems 2 (279) (1990) 598–605.
[18] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, N. De Freitas, Predicting parameters in deep learning, arXiv preprint arXiv:1306.0543 (2013).
[19] S. Anwar, K. Hwang, W. Sung, Structured pruning of deep convolutional neural networks, ACM Journal on Emerging Technologies in Computing Systems (JETC) 13 (3) (2017) 1–18.
[20] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, R. Fergus, Exploiting linear structure within convolutional networks for efficient evaluation, in: Advances in neural information processing systems, 2014, pp. 1269–1277.
[21] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, Y. Chen, Compressing neural networks with the hashing trick, Computer Science (2015) 2285–2294.
[22] S. Han, J. Pool, J. Tran, W. J. Dally, Learning both weights and connections for efficient neural networks, arXiv preprint arXiv:1506.02626 (2015).
[23] S. Anwar, K. Hwang, W. Sung, Structured pruning of deep convolutional neural networks, ACM Journal on Emerging Technologies in Computing Systems 13 (3) (2015).
[24] D. Wang, L. Zhou, X. Zhang, X. Bai, J. Zhou, Exploring linear relationship in feature map subspace for convnets compression, arXiv preprint arXiv:1803.05729 (2018).
[25] D. Tabernik, S. Sela, J. Skvarc, D. Skocaj, Segmentation-based deep-learning approach for surface-defect detection, Journal of Intelligent Manufacturing 31 (3) (2020) 759–776.