Machine Learning (2022) 111:3161–3180
https://doi.org/10.1007/s10994-022-06193-w

Pruning convolutional neural networks via filter similarity analysis

Lili Geng[1,2] · **Baoning Niu**[1]

Received: 29 September 2020 / Revised: 6 December 2021 / Accepted: 10 December 2021 /
Published online: 23 June 2022

Abstract
Deep learning has shown excellent performance in many fields, especially image recognition and retrieval in recent years. The performance of convolutional neural networks (CNNs) is particularly outstanding. CNNs, however, are usually computationally intensive, which hinders the deployment of CNNs in resource-limited devices. Methods of network compression, pruning methods in particularly, removing redundant structures of CNNs, can significantly reduce the computational complexity of CNNs. Most of the state-of-the-art pruning methods for CNNs, however, have two defects. (1) Filters, also called convolutional kernels that are matrices used to extract features in an image, are pruned by ranking their weight without considering the effects of their actual output, which results in the deletion of important filters and the difficulty in determining the pruning threshold on weight. (2) Filters are pruned either in the forward direction or in isolation, which are difficult to control the loss of accuracy. This paper proposes a novel pruning method called filter similarity analysis with backward pruning (FSABP). FSABP calculates the similarity coefficients of filters in each layer, and deletes the filters associated with small similarity coefficients. The smaller the coefficient the more similar the filters. Filters are pruned layer by layer in the backward direction starting from the last convolution layer, which can effectively control the loss of accuracy by avoiding early removal of the shallow convolution filters. Experiments on LENET, VGG-16 and ResNet-50 show that FSABP can reduce parameter redundancy at the cost of negligible loss of accuracy and even improve accuracy in some cases. The results on LENET also suggest that FSABP is applicable to both deep and shallow CNNs.

Keywords Convolutional neural networks · Deep learning · Network compression · Pruning

Editors: João Gama, Alípio Jorge, Salvador García.

✉ Lili Geng
genglily@163.com

Extended author information available on the last page of the article

1 Introduction

Deep learning has made significant progress in the past few years, especially in the field of image recognition and retrieval. Convolutional neural networks (CNNs), in particular, have achieved remarkable performance, and significantly outperform the traditional methods in recognizing visual features. CNNs have been designed deeper and deeper to improve their accuracy (He et al., 2016; Krizhevsky et al., 2012; Simonyan and Zisserman 2014; Szegedy et al., 2015) and are characterized as with a large number of parameters, computationally intensive, and a big memory footprint. It is challenging to deploy CNNs on the resource-limited devices without sacrificing their accuracy (Howard et al., 2017; Iandola et al., 2016; Ma et al., 2018; Sandler et al., 2018; Zhang et al., 2018).

Network pruning reduces computational complexity and the memory footprint by effectively compressing the architecture of CNNs. The convolutional layers and the full connection layers usually contain a large number of parameters and a significant portion of them is redundant after training (Han et al., 2015). Pruning the redundant parameters will not significantly reduce the accuracy of the networks. With fewer parameters, CNNs are not only simplified and much fast, but also possibly deployed on the resource-limited devices when the reduced resource demands can be met.

Pruning is usually done by deleting filters and weights. A filter, also called a convolutional kernel, is a matrix used to extract features of an image. Filter pruning is the most typical method. The state-of-the-art filter pruning methods, however, have two defects. (1) Filters are pruned by ranking their weight and setting pruning thresholds, while ignoring their actual output, which result in the deletion of important filters and is difficult to determine the pruning threshold. (2) There is no standard pruning direction in the existing methods, and it is difficult to control the loss of precision whether the pruning filter is forward or isolated.

For the defect (1), we argue that the similarity of filters is more important than the weight. In many cases, small weight plays an important role in loss functions (Li et al., 2017). Based on the facts, we propose a pruning method called the filter similarity analysis (FSA), which takes the similarity of filters into account when pruning filters. For the defect (2), we argue that backward pruning is more appropriate than forward or isolated pruning in terms of controlling the loss of accuracy of CNNs. As a feed forward network with hierarchical natures, CNNs use the shallow filters to learn basic local features and the deep filters to capture semantic features (Zeiler and Fergus, 2014). The effects of deleting a filter usually propagate to the deeper layers, but the shallower layers. Backward pruning has less or no such an effect, which avoid the early deletion of important filters and the excess loss of accuracy.

The contributions of our paper are three aspects.

1 FSA is proposed to prune filters of CNNs. Similar filters in a layer are found by calculating their similarity coefficients of the weight matrices. Those filters with high similarity are kept, others are deleted. By finding and removing redundant filters in terms of extract similar features, FSA can retain the ability of a convolutional layer to extract rich features while reducing the complexity of CNNs.
2 FSA with backward pruning, FSABP in short, is proposed to control the loss of the accuracy of CNNs and filters are pruned layer by layer starting from the last convolutional layer to the first. The accuracy varies significantly in the forward pruning process. When the loss of accuracy is too great the chance is slim to recover from the loss through fine

tuning. In the process of backward pruning, however, the loss of accuracy is small and can be easily recovered.

3 We conduct experiments to show that the proposed methods are effective for both shallow and deep CNNs, and applicable to any convolutional layer and not affected by the size of CNNs.

The rest of the paper is organized as follows: Sect. 2 introduces related work. Section 3 describes our ideas and pruning methods in detail. The experimental results are discussed in Sect. 4. Section 5 is the conclusions.

2 Related work

Pruning is the most widely used network compression method. Because there are a large number of redundant parameters in CNNs, the performance of CNNs is almost unaffected by deleting a certain proportion of parameters. Pruning methods can be divided into two categories: weight pruning and filter pruning.

The strategy of weight pruning increases the sparsity of the weight matrix to make some weights tend to zero by introducing regularization terms into the objective function. Small weights are removed. The process of pruning is iterative. After pruning small weights, the network is fine-tuned to restore accuracy. Weight pruning requires producing sparse matrices, which are complex in calculation and requires special hardware to efficiently support sparse matrix operations.

The process of filter pruning consists of training the original CNNs, ranking the filters by predefined criterions, and reserving the top-ranked filters and pruning the rest. The pruned CNN is then fine-tuned and retrained to achieve the same or even higher accuracy than the original CNN. Compared to weight pruning, filter pruning directly deletes filters from CNNs and controls the pruning ratio through setting thresholds and has two advantages, no sparse matrix generated and no need for huge disk storage and memory in the reasoning stage. The key to filter pruning is how to select the filters to be deleted. The selection criterion can be filter specific or filter nonspecific. The filter specific criterion is derived directly from the weight of a filter and can be categorized as follow. (1) Random pruning: Mittal et al. (2019) obtain the same performance as the state-of-the-art pruning methods by randomly pruning 25–50% filters, and suggest that the inherent plasticity of CNNs allows the loss of accuracy caused by pruning to be recovered by fine-tuning. (2) Entropy-based: Luo and Wu (2017) propose to use entropy to evaluate the importance of filters. The parameters of VGG-16 (Simonyan and Zisserman, 2014) and ResNet-50 (He et al., 2016) are reduced by 16.64× and 1.47×, and the accuracy of both are decreased by 1%. (3) Energy-Aware: Yang et al. (2017) directly prune filters according to the energy consumption of CNNs. (4) Filter similarity: Singh et al. (2018) iteratively divide two filters with the greatest similarity into a pair and remove one filter from each pair and obtain a very good FLOPs (floating point operation per-second) compression rate in various benchmark tests. (5) Zero activation: Hu et al. (2016) pruned the filters with a lot of zero activations, reinitialized the network with initial weight, and then retrained the network, which achieve 2× compression ratio without loss of accuracy.

The filter nonspecific criterion is derived from the changes of the characteristics associated with filters and can be categorized as follow. (1) Adjacent layer filter: Hu et al. (2018) used a genetic algorithm to select filters in order to compress the ultra-deep CNNs. Luo

et al. (2017) proposed a framework named ThiNet to delete the filters by calculating the statistics of the next layer. (2) Feature map: Anwar and Sung (2016) proposed a pruning method for feature mapping. They designed the strategy of selecting the least antagonistic pruning mask. The pruning template was generated randomly, and the network was pruned by using the verification set to select the best mask. (3) Loss function: Molchanov et al. (2016) proposed a pruning criterion based on Taylor expansion to find filters that have less influence on the loss of accuracy and deleted them to reduce the size of CNNs. Based on this criterion, the performance of the pruned CNNs is superior in fine-grained classification tasks.

The advantage of the filter specific criterion is that the calculation is straight forward, while the disadvantage is that the filters are pruned individually without taking into account their similarities with others, which tend to produce excess loss of accuracy and the difficulty in determining the pruning threshold. The advantage of the filter nonspecific criterion is the loss of network accuracy is not significant, while the disadvantage is that the calculation is complex. Instead of judging the importance of filters individually, our proposed method is based on their similarity. The difference between ours and Singh et al. (2018) is that our similarity is based on the Markov distance of the weight matrix of a filter, instead of the Pearson correlation coefficient of filter pairs. We calculate the similarity coefficient according to the weight matrix of the filter, and delete the filter whose coefficient is less than the mean value.

Another issue related to filter pruning is the pruning direction. There is no preferred pruning direction in the literature. Filters are deleted either forward or in isolation. Hu et al. (2016) and Luo and Wu (2017) prune part of a convolutional layer. Yang et al. (2017) determined the direction of pruning according to the energy consumption of each layer. Anwar and Sung (2016) prune the network once at a given compression ratio and then retrain it. They can be categorized as isolated pruning. Hu et al. (2018) and Singh et al. (2018) prune each layer sequentially from input to output, which can be categorized as forward pruning. The isolated pruning leads to low compression rate, while the forward pruning leads to excess loss of accuracy while increasing compression rate. Our proposed method prunes backward, from the last convolution layer to the first convolution layer, in order to achieve a balance between the compression rate and the loss of accuracy.

3 Pruning with filter similarity analysis

This section describes the pruning method by filter similarity analysis. The motivation and problem are described in Sect. 3.1. Section 3.2 details the filter similarity calculation method. The pruning order, an important factor affecting the network accuracy, is discussed in Sect. 3.3.

3.1 Motivation and problem description

CNNs extract the features of an image, such as edges, texture, color, etc., by convolutional operations, and then perform a specific image processing task based on the extracted features. The number of filters is usually thousands in deep CNNs. The weights of a filter are derived from network learning. It is inevitable that there exist a lot of similar filters which extract the same or similar features. Keeping a part of the similar filters can ensure the

extraction of the corresponding features, while removing the remaining filters does not significantly reduce the network accuracy.

How to judge the similarity of two filters is the first problem to be solved in order to prune filters. The weights of a filter determine the type of features extracted by the filter, so similar filters might be found by analyzing the weight values.

CNNs are composed of input layers, convolutional layers, pooling layers, and fully connected layers. A CNN N can be described using Eq. 3.1.

$$\begin{cases} A_i = X & i = 1 \\ A_i = f\left(h_i\left(A_{i-1}, W_i\right)\right) & i > 1 \end{cases} \tag{3.1}$$

X is the input data, a two-dimensional matrix composed of image pixels. The first layer is the input layer and, therefore, $A_1 = \mathrm{X}$. When $i > 1$, the ith layer can be convolutional, pooling, or fully connected. Convolutional and fully connected layers contain weight parameters and are represented as one or more higher-dimensional matrices. $f(\cdot)$ is the activation function and $h(\cdot)$ represents the output calculations for different types of layers as defined in Eq. 3.2.

$$h_i = \begin{cases} A_{i-1} * W_i + b_i & \textit{the ith layer is the convolutional layer} \\ down\left(A_{i-1}\right) & \textit{the ith layer is the pooling layer} \\ W_i \cdot A_{i-1} & \textit{the ith layer is the full conection layer} \end{cases} \tag{3.2}$$

If a layer is a convolutional layer, convolutional calculations ($A_{i-1} \times W_i$) involving filters and input data are carried out, where b_i is the bias item. If a layer is a pooling layer, downsampling on the input data happens, and $down(\cdot)$is called the down-sample function. If a layer is a full connection layer, the matrix multiplication ($W_i \cdot A_{i-1}$) is applied on the filter and the input data, which flattens the input matrix into a vector.

We target to delete the filters of the convolutional layers of a network. To formally describe the process of pruning filters, we denote $F(c, l_c, n, k)$ as the cth convolutional layer with l_c filters in the network N, each filter having a set of weights, represented by a $n \times k \times k$ matrix, where, k represents the size of the receptive field (a $k \times k$ matrix) of a filter, and n is the number of channels of the input data. The goal of pruning is to make the number of filters as small as possible while ensuring the network accuracy, thus reducing the total number of weight parameters and the size of the network. The pruning effect is usually measured by the change of top-1 accuracy of the network. We denote the function $g(\cdot)$ as the network accuracy. Ensuring network accuracy after pruning means that the loss of accuracy is less than a given threshold θ. Therefore, the optimal pruned network can be described using Eq. 3.3.

$$N^* = \underset{N' \subseteq N}{argmin} \left|N'\right| = \sum_c F\left(c, l_c, n, k\right) \quad s.t. \quad g(N) - g\left(N'\right) \leq \theta \tag{3.3}$$

In Eq. 3.3, N, N', N^* represents the original network, pruned network and the smallest pruned network, respectively. $|N'|$ is the size of the pruned network. $g(N)$, $g(N')$ is the top-1 accuracy before and after network pruning, and θ is the threshold of the loss of accuracy. N^* is the optimal network satisfing the loss threshold.

Equation 3.3 represents the optimal pruning, the one with the least total number of filters in the network to satisfy the threshold. For each convolutional layer, there are multiple combinations of filters to be pruned. A network with m convolutional layers has $2^{l_1+l_2+\cdots+l_m}$

possible combinations. The search space of the optimal network is too large to be practical. Therefore, we propose a heuristic pruning method, which takes reverse pruning strategy and determines the number of filters to be pruned layer by layer starting from the last convolutional layer to the first.

In FSA algorithm, the filter to be pruned needs to be identified after calculating the filter similarity. For the cth convolutional layer with l_c filters, all the filters can be collectively represented by a filter vector $T_c = (t_0, t_1, \ldots, t_{l_c})$, where each component corresponding to a filter, can be 0 or 1, respectively indicating that the filters are deleted or retained. The pruning result can be described using Eq. 3.4.

$$FSA(N, m, l) = \{N', T' | T' = \{T_1, T_2, \ldots, T_m\}\} \tag{3.4}$$

where $FSA(N, m, l)$ represents applying FSA method to the network N with m convolutional layers and l filters to obtain the network N', T' is a set of all convolutional layer filter vectors constructed by the FSA algorithm.

Combining Eqs. 3.3 and 3.4, q, the number of filters after pruning all the convolutional layers can be calculated by Eq. 3.5.

$$q = \sum_{c=1}^{m} \left(l_c - |T_c|_{t_j=0} \right) \quad s.t. \quad 0 \le j \le l_c \tag{3.5}$$

where $|T_c|_{t_j=0}$ is the number of filters to be deleted at the c-th layer, that is, the number of 0 s in the vector T_c, $l_c - |T_c|_{t_j=0}$ is the number of filters remaining after pruning.

3.2 Filters similarity calculation

FSA determines the number of filters to be pruned and their indexes based on the similarity of the filters. The similarity of data objects is generally measured by distance. The similarity of filters we used is not the commonly used distance, but is based on the similarity between different channel dimensions in the filter weight matrix, known as the filter similarity coefficient. The adjacent pixels of an image are usually similar, and the results of convolution operations on the same location of the filter are similar. Judging filter similarity needs to consider the similarity between channels.

For an input data with a channel number of n, the filter weight W is a $n \times o \times o$ matrix, and o represents the height and width of the matrix, which is the filter receptive field. When calculate the similarity or correlation of high-dimensional data, it is common practice to conduct dimension reduction in order to extract key information and get rid of the noise caused by irrelevant dimensions.

A filter weight matrix has three dimensions, including channel, height, and width. The number of channels of a weight matrix is the same as the number of the input data channels. Retaining the channel dimensions in the filter weight matrix ensures the integrity of the multi-channel input data. We choose to reduce the height of the weight matrix to 1, which is one of the two dimensions representing the filter receptive field. This reduces the filter weight matrix to two dimensions, denoted as $\overline{W}$, $\overline{W} = \left[\overline{W}_{ij}\right]_{n \times o}$. In Fig. 1, the input data is the $32 \times 28 \times 28$ blue matrix, which is operated by a 5×5 convolutional filter, and the weight matrix of the filter is the $32 \times 5 \times 5$ yellow matrix. The dimension reduction based on the updated weights after convolution reduces information loss. The weight matrix is compressed by taking the means of the height to get the 32×5 green matrix.

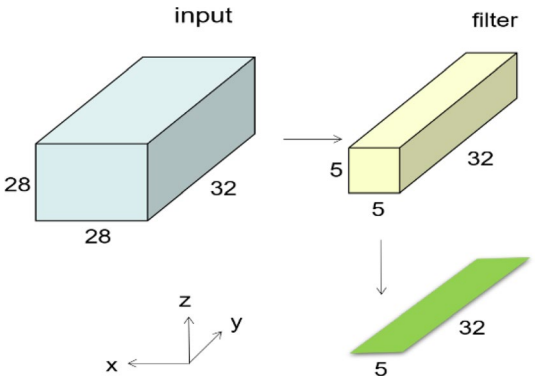

Fig. 1 Dimension reduction of the filter weight matrix. The 32×28×28 input data are convolved with 5×5 filter to produce a 32×5×5 weight matrix. Take the mean value on the z-axis to reduce the dimension, so that the matrix becomes a matrix of 32×5

The compressed two-dimensional (2-D) matrix can be seen as a set of row vectors or column vectors. Row vectors represent samples from corresponding channels, and column represent properties. Channel similarity is calculated based on their distance, and the similarity coefficient of a filter is the average of the channel similarity. The column dimension is corresponding to the filter receptive field, extracts information from the adjacent regions in the input data, and the correlation of this information is extremely high. This is the reason why we calculate the similarity between channels.

Maharanobis distance calculates the distance between samples with multiple dimensions, and the covariance in it calculates the correlation between the dimensions of a sample, which match our needs to calculate the similarity of samples consisting of channel or row vectors in a filter, and the correlation of the samples consisting of the column vectors.

The covariance of the weight matrix is calculated using Eq. 3.6.

$$S = \sum \overline{W} = E\left[\left(\overline{W} - E\left[\overline{W}\right]\right)\left(\overline{W} - E\left[\overline{W}\right]\right)^{T}\right] \tag{3.6}$$

Equation 3.7 shows how to calculate the similarity (d_{ij}) between row vectors. The relationship between the similarity of matrix row vectors and the similarity of the weight matrix of a filter is explained in Fig. 2.

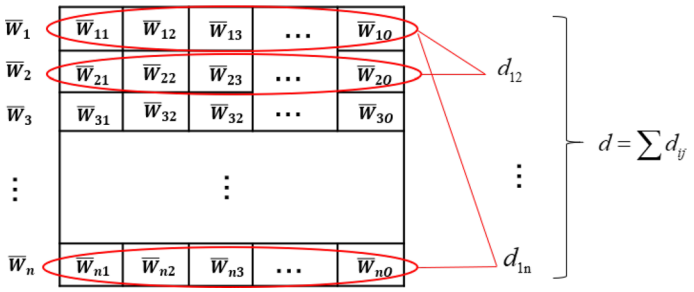

Fig. 2 The similarity of the filter 2-D matrix. $\overline{W}_i (1 \le i \le n)$ represent the different row vectors of the filter 2-D matrix. n is the number of channels. O represents the width of the matrix. Calculate the Markov distance similarity of the row vector and summed. d is the similarity of 2-D matrix

$$d_{ij} = \sqrt{\left|(\overline{W}_i - \overline{W}_j)^T S^{-1}\left(\overline{W}_i - \overline{W}_j\right)\right|} \quad (0 \le i \le n, 0 \le j \le n, \quad i \ne j) \tag{3.7}$$

The similarity coefficient of a filter, denoted as d, is the sum of the similarity of the matrix row vectors, as shown in Eq. 3.8. similarity and takes the mean as the similarity coefficient of the filter.

$$d = \sum_{i}^{n} \sum_{j}^{n} d_{ij} \quad (0 \le i \le n, \quad 0 \le j \le n, \quad i \ne j) \tag{3.8}$$

The mean of the similarity of weight matrix d to the number of channels is the similarity coefficient δ of the filter.

After obtaining similarity coefficients δ(δ = d/n) for all filters, the mean of the similarity coefficients ε(ε = δ/m) is calculated for all filters. Filters with similarity coefficients less than the mean were removed. The similarity coefficient of a filter reflects the total effects of the pair-wise similarity among channels of a filter. Filters of the same convolutional layer extract the same attribute features (Zeiler and Fergus, 2014). For example, if a layer extracts texture features, all the filters in the layer extract them, but with different texture directions determined by the similarity between channels, namely the similarity coefficients. Such similarity is independent of the weights, and the similarity coefficient of a filter with small weights might be larger than that with large weights. Our method differs from the method that determines the importance of the filter based on the weights. A large similarity coefficient indicates that the corresponding filter extracts the significant different feature from others, and small one indicates similar features being extracted by others. Filter pruning tries to remove as many filters with similar functions as possible. We found that the filters with small similarity coefficients take a large proportion of all the filters (Fig. 3). The proportion of the filters with small similarity coefficients varies for different layers. It is tedious to determine an appropriate ratio for each layer. Taking the mean value of the similarity coefficients as the pruning threshold is not only simple, but also can remove the majority of the filters with small

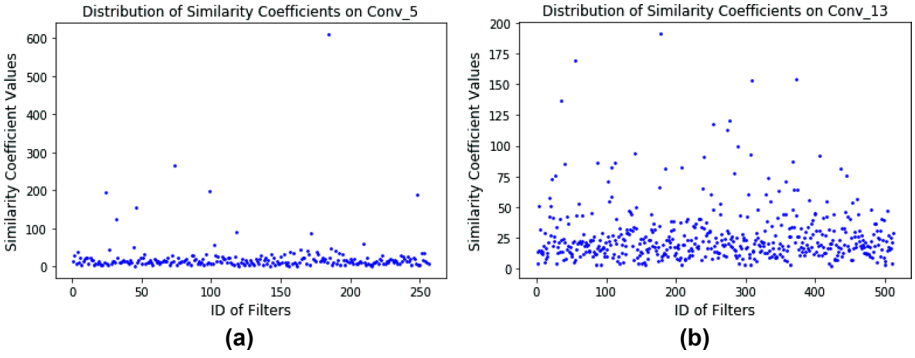

Fig. 3 Distribution of similarity coefficients. **a** and **b** show the distributions of similarity coefficients in the fifth and thirteenth convolutional layers in the VGG-16, respectively. Filters with small similarity coefficients take a large proportion of all the filters. The x-axis represents the ID of filters, from 0 to the number of filters. The y-axis represents the values of similarity coefficients

similarity coefficients. The experiment results (Fig. 6) show that, after removing those filters with similarity coefficients less than the mean, filters with both large weights and small weights are retained, with a distribution similar to that of the original network.

The calculation process of the filter similarity coefficient is as follows:

Algorithm 1 Filter Similarity Analysis

Input: the matrix of filter weight

T← ϕ ; A← ϕ ; K ← [the index of filters]; m; n;

for each item k ∈ K index do
  W ←the matrix of filter weight;
  $\overline{W}$ ←reduction dimension of the filter weight matrix;
  S = The covariance of $\overline{W}$ was calculated;
  for *i* ∈ I, I ← (0, n) do /*n is the number of input data channels */
    for *j* ∈ J, J ← (0, n) do
      calculate d_{ij} by Eq.3.7
    end for
  end for
  d = sum d_{ij};
  δ = d/n;
  T← δ;
end for
ε=T/m; /* calculate mean of the similarity coefficient δ for all filters in T. The m is the number of filters*/
for i ∈ I, I ← (0, m) do
  When T_i < ε, A← i;
end for

Output: The array A of filter indexes to delete

Because different filters extract different feature information, pruning filters with high similarity is equivalent to removing filters extracting similar features. For example, if two filters extract similar color characteristics of a picture, their weight matrices tend to be similar to ensure they extract similar feature information. Filters that extract the same or similar features form a set, deleting some filters in each set according to the FSA algorithm to ensure the overall feature extraction capability and integrity of the convolutional layers. Figure 4 clearly shows the changes in the outputs of filters of VGG-16 before and after pruning. The upper part of the figure is the pre-pruning outputs, and the red boxes mark the outputs with obvious different features. After FSA pruning, the extracted features of the filters retained in the lower part of the figure are as rich as without pruning. Multiple similar features in this case are yellow blocks, and

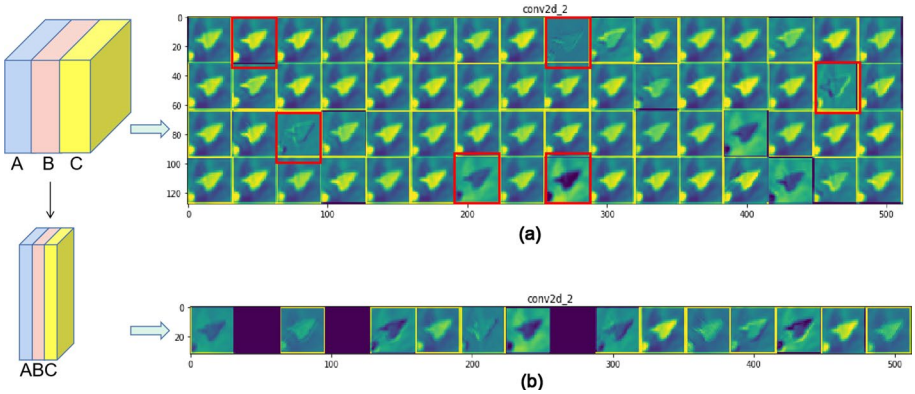

Fig. 4 Visualization of filter pruning results. A, B and C represent the different groups of filters. Filters in each set are reduced after pruning. **a** is the feature map of each filter in the unpruned convolution layer. **b** is the feature map of each filter after pruning. Those red boxes in **a** marks the different feature map of the filters. As can be seen from **b**, part of the feature map of different features is retained after pruning filters

only two of them are retained. The more similar filters are removed, the higher the pruning rate is.

3.3 Backward pruning

The state-of-the-art pruning methods select only prune certain convolutional layers in the network, and do not take full account for the effect of pruning order on accuracy of the network. We propose a backward pruning strategy (FSABP). Since CNN is a feedforward network, and the outputs of filters of the shallow convolutional layers are progressively transmitted to the deep convolutional layers.

For example, the first convolutional layer extracts smaller local features of the image (e.g., edge, color, texture, etc.). The second convolutional layer obtains more abstract concepts from the output of the first layer and extracts larger image features (e.g., eyes, ears, mouth, etc.). The features extracted by deep convolutional layers are the fusion of the features extracted by shallow convolutional layers. In general, shallow convolutional layers extract more general and reusable features of datasets, and deep layers learn increasingly complex and abstract visual concepts.

As the network getting deep, the number of parameters increases sharply. The deeper the network layer, the more filters in the convolutional layers, and the greater the parameters and redundancy in each layer. Pruning starting from more redundant and deep convolutional layers has little effect on the accuracy of the network. The shallow convolutional layers contain fewer parameters and less redundancy. Pruning shallow convolution layers not only reduces the accuracy of the network, but also is difficult to repair the loss of accuracy. Pruning from the last convolutional layer maximizes the accuracy of the network and balances the network size and accuracy. Figure 5 shows the backward pruning.

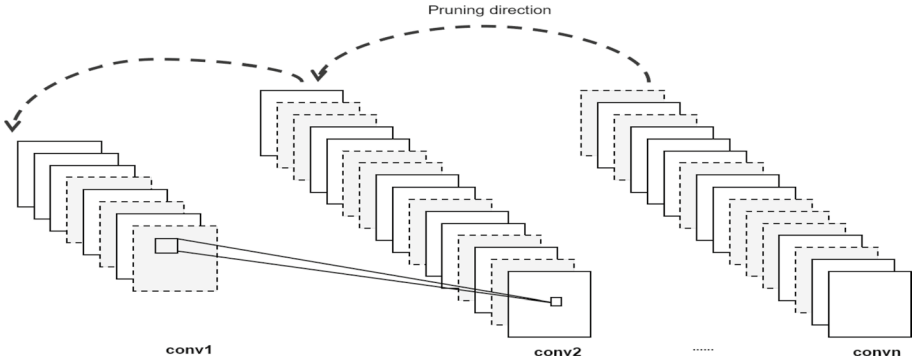

Fig. 5 Backward direction pruning. Filters are prune layer by layer in the backward direction starting from the last convolution layer

4 Experiments

This section describes the experimental design. Sections 4.1 and 4.2 introduce the network model, datasets and evaluation indicators. Section 4.3 discusses the experimental results on LENET (Lecun & Bottou, 1998) and VGG-16. Section 4.4 compares the results of forward and backward direction pruning, and the Sect. 4.5 describes the experimental results on ResNet-50.

4.1 Experimental networks and datasets

Our experiment was carried out on Intel Xeon Brone 3106@1.70 GHz CPU and NIVIDIA GeForce GTX 1080Ti GPU. We evaluated FSA on keras and selected three typical CNNs, including LENET, VGG-16 and ResNet-50. LENET and VGG-16 are sequential convolutional stack networks. ResNet-50 is residual network.

In the experiment, we used MNIST, CIFAR-10, KAGGLE and ISLVRC 2012 (ImageNet Large Scale Visual Recognition Challenge) datasets. The MNIST dataset is a classic dataset in the field of machine learning, including 60,000 training images and 10,000 test images, which are grayscale images of handwritten numbers. The CIFAR-10 dataset consists of 60,000 32×32 resolution color images, 50,000 training images and 10,000 test images, and 10 categories. The KAGGLE dataset contains 25,000 color images of dogs and cats, 12,500 images for each category. ISLVRC is a subset of the ImageNet dataset, and contains over a million color images and 1000 categories. We randomly selected 100 of these categories for our network training.

In the table, "Conv" denotes the convolutional layer and "FC" is the full connection layer. The shape column represents the layer structure.

LENET includes two convolutional layers (Conv_1 and Conv_2) and two fully connection layers (FC_1 and FC_2). The first convolutional layer has 20 filters, and the filter size is 5×5, while and the second convolutional layer has 50 filters of size 5×5. The accuracy can reach 99.2% on the MNIST dataset. The composition of VGG-16 includes 3×3 convolutional layer with increasing depth, max pooling, two full connection layers of 4096 nodes and the softmax classifier. As show in Tables 1 and 2.

Table 1 LENET basic structure

Type	Shape	Output size
Conv_1	5×5×20	28×28×20
Max pooling	2×2	14×14×20
Conv_2	5×5×50	14×14×50
Max pooling	2×2	7×7×50
FC_1	500	500
FC_2	10	10
Softmax	Classifier	10

Table 2 VGG-16 basic structure

Type	Shape	Output size
Conv_1	3×3×64	32×32×64
Conv_2	3×3×64	32×32×64
Max pooling	2×2	16×16×64
Conv_3	3×3×128	16×16×128
Conv_4	3×3×128	16×16×128
Max pooling	2×2	8×8×128
Conv_5	3×3×256	8×8×256
Conv_6	3×3×256	8×8×256
Conv_7	3×3×256	8×8×256
Max pooling	2×2	4×4×256
Conv_8	3×3×512	4×4×512
Conv_9	3×3×512	4×4×512
Conv_10	3×3×512	4×4×512
Max pooling	2×2	2×2×512
Conv_11	3×3×512	2×2×512
Conv_12	3×3×512	2×2×512
Conv_13	3×3×512	2×2×512
Max pooling	2×2	1×1×512
FC_1	4096	4096
FC_2	4096	4096
FC_3	Category	10
Softmax	Classifier	10

4.2 Evaluation indicators

The compression rate (CR) is an indicator to evaluate the pruning effect of CNNs.

$$CR = \frac{l}{q} \tag{4.1}$$

Here l is the total number of filters and q is the number of remaining filters. However, the high CR does not mean a good pruning effect because the evaluation of CNNs also depends on the accuracy of the network, the calculation speed, and the number of

parameters, etc. The commonly used evaluation indicators of CNNs include the accuracy of Top-1 and Top-5, error rate, the number of the parameters and FLOPs.

CIFAR-10 has 10 categories. When a network predicts an image, it ranks the probability the image belongs to the 10 categories from high to low. Top-1 refers to the accuracy of the categories ranked the first matching the actual categories. Top-5 refers to the accuracy of the categories ranked from the first to the fifth matching the actual categories.

FLOPs is short for floating point operation per-second in the network. The FLOPs of CNNs (*net_FLOPs*) consists of FLOPs of the convolution layer (*conv_FLOPS*), the pooling layer (*pooling_FLOPs*) and the full connection layer (*FC_FLOPs*).

$$conv\_FLOPs = C_{in} \times C_{out} \times \mathrm{k} \times \mathrm{k} \times H_{out} \times W_{out} \times \mathrm{BatchSize} \tag{4.2}$$

$$pooling\_FLOPs = C_{in} \times H_{in} \times W_{in} \times BatchSize \tag{4.3}$$

$$FC\_FLOPs = C_{out} \times C_{in} \times Batchsize \tag{4.4}$$

$$net\_FLOPs = conv\_FLOPs + pooling\_FLOPs + FC\_FLOPs \tag{4.5}$$

In the above formula, (C_{in}, H_{in}, W_{in}) is the input feature. $k \times k$ is the size of the filter. (C_{out}, H_{out}, W_{out}) is the output feature. The *Batchsize* is the number of samples selected for one training.

The number of the parameters (*num_Param*) is as follows:

$$num\_Param = C_{out} \times \left[\left(C_{in} \times k \times k\right) + 1\right] \tag{4.6}$$

Since the pruning is a kind of damage to CNNs, it is almost impossible to improve all evaluation indicators at the same time. In general, the higher the CR, the greater the loss of accuracy. The effect of the network pruning depends not only on the choice of the pruning method but also on the influence of the training methods. The network pruning is a comprehensive optimization process. This paper focused on the test of the pruning algorithm, and does not particularly optimize the training process of CNNs, so it is still possible to improve the accuracy of the pruned CNN.

4.3 The results of pruning LENET

We train the LENET from scratch and set the batch size of 32 for 1000 epochs and the cross-entropy loss is used as the criterion function. The optimizer uses Adma (Adaptive moment estimation). We achieved the Top-1 accuracy of 99.29% and the Top-5 accuracy of 99.99%. In order to verify the validity of FSA algorithm, we firstly prune the full connection layer and the second convolutional layer of LENET, then prune the first convolutional layer after fine-tuning. The FSA-1 is the network after the first round of iterative pruning, and the FSA-2 is the network after the second round of iterative pruning.

Finally, the Top-1 and Top-5 accuracy of the FSA-1 is 99.01% and 100%, respectively. The number of parameters has been reduced from 1.25 to 0.096 M. The total number of parameters is cut by 92.41% and FLOPs is cut by 73.45%. The Top-1 and Top-5 accuracy of the FSA-2 is 98.67% and 99.99%, respectively. The number of parameters and the FLOPs are 0.021 M and 0.27×10^7, respectively, and are reduced by 98.32% and 94.03%, respectively.

Table 3 compares our method to the state-of-the-art methods, including GAL (Lin et al., 2019) and CFP (Singh et al., 2018).

Compared with GAL-λ(λ is sparsity factor), λ = 0.1, FSA-2 has a much higher CR (6.20). Both CR and the accuracy of FSA-2 are better than those of CFP-4(CFP-x, x is the number of pruning iterations). FSA has a high compression ratio and a small loss of accuracy.

We retrain two networks ($\overline{FSA}$_1 and $\overline{FSA}$_2) with the same structure as the *FSA*-1 and *FSA*-2, and use the same parameter setting and dataset. The error rate, Top-1 and Top-5 of $\overline{FSA}$_1 are 1.27%, 98.73% and 99.99%, respectively, while those of $\overline{FSA}$_2 are 1.41%, 98.59% and 99.99%, respectively. The accuracy of the original network is less than that of the pruned networks.

4.4 The results of pruning VGG-16

Experiments on LENET prove that our algorithm is effective to shallow CNNs. Next, we train VGG-16 from scratch on CIFAR-10 and KAGGLE datasets. The results also suggest the effectiveness of our algorithm to deep CNNs by comparing the evaluation indicators of the baseline with those of the pruned networks.

4.4.1 VGG-16 on CIFAR-10

The VGG-16 is trained from scratch using the CIFAR-10 dataset. And the network is trained with batch size of 32 for 1000 epochs and the cross-entropy loss is used as the criterion function. Meanwhile, the optimizer uses SGD (Stochastic Gradient Descent) and the regularization term is L2.

When pruning the first and the second convolutional layers, a little drop of the number of parameters causes a significant loss of accuracy. These two layers have nothing to be pruned. Table 4, lists the change of evaluation indicators after each layer of backward pruning. The number of filters decreases from 4224 to 1303 after pruning, and CR reaches 3.24. Final Top-1 accuracy of VGG-16 is 85. 51%, 0.19% lower than the original VGG-16, and 15.34% higher than the APoz (Hu et al., 2016). Top-5 accuracy of VGG-16 is 99.34%,

Table 3 The performance of LENET on the MNIST dataset

Method	CR	acc (%)	FLOPs	FLOPs↓ (%)
CFP-1	1.046	99.09	1.95×10^5	95.56
CFP-2	1.047	99.05	1.58×10^5	96.41
CFP-3	1.048	98.80	1.39×10^5	96.84
CFP-4	1.05	98.23	1.95×10^5	97.98
GAL-0.01	2.56	99.05	0.43×10^7	81.20
GAL-0.05	4.13	98.95	0.17×10^7	92.60
GAL-0.1	4.63	98.97	0.10×10^7	95.60
Baseline	1.00	99.03	4.52×10^7	0
FSA-1	**3.54**	**99.01**	$\mathbf{1.20\times10^7}$	**73.45**
FSA-2	**6.20**	**98.67**	$\mathbf{0.27\times10^7}$	**94.03**

"Baseline" refers to the LENET that we trained from scratch. "FLOPs↓" is the percentage decrease of the FLOPs. "acc" denotes the accuracy of the network. The data of FSA are in bold

Table 4 The results of backward pruning VGG-16

CONV	Filters	Top-1	Top-5	Para(M)	Para↓ (%)	FLOPs	FLOPs↓ (%)
Baseline	4224	85.70	99.05	15.27	0	3.39×10^8	0
Conv_13(512)	166	86.16	99.08	13.49	11.66	2.29×10^8	32.34
Conv_12(512)	155	86.02	98.99	11.31	25.93	1.92×10^8	43.27
Conv_11(512)	175	84.28	98.92	9.29	39.16	1.58×10^8	53.43
Conv_10(512)	103	82.97	98.94	6.76	55.73	1.15×10^8	66.13
Conv_9(512)	129	86.42	99.26	4.64	69.61	0.79×10^8	76.79
Conv_8(512)	160	86.77	99.03	3.41	77.67	0.58×10^8	82.90
Conv_7(256)	88	87.19	99.43	2.78	81.79	0.47×10^8	86.06
Conv_6(256)	89	87.48	99.41	2.27	85.13	0.38×10^8	88.66
Conv_5(256)	55	84.62	99.22	1.87	87.75	0.32×10^8	90.64
Conv_4(128)	90	84.81	99.19	1.81	88.15	0.31×10^8	90.93
Conv_3(128)	93	85.51	99.34	1.76	88.47	0.29×10^8	91.20

"Baseline" refers to the VGG-16 that we trained from scratch. "Para." is the number of parameters and M means million. "Para↓" is the percentage decrease of the parameters. "FLOPs↓" is the percentage decrease of the FLOPs. The "CONV" column is the convolutional layer. Forexample, "conv_13(512)" refers to the 13th convolutional layer and there are 512 filters. The "Filters" column is the number of filters left after pruning

0.29% higher than the original VGG-16, and 9.65% better than the APoz. The number of parameters decreases by 88.47% and FLOPs drops by 91.20%, which are better than those of CFP, GAL, GA (Hu et al., 2018), and LWM (Li et al., 2017), as shown in Table 5.

The distribution of filter similarity before and after pruning is shown in Fig. 6. Figure 7 shows that the change of filter weights does not cause the change of distribution. The normal distribution of weights suggests that they are caused by random factors, and the randomly initialized training network after pruning can still maintain the original performance without relying on the original filter weights. It is the number of filters that affects the accuracy of a network, rather than weights.

We re-train two networks (VGG_1 and VGG_2) to verify the advantages of pruning network structure. VGG_1 and VGG_2 have the same structure and the number of filters, of the VGG-16 after pruning, respectively, but different distribution in each layer. With the same parameter setting and datasets. The accuracy of VGG_1 is 87.11%, and that of

Table 5 The results of pruning VGG-16 on CIFAR-10

Method	acc↓ (%)	Para(M)	Para↓ (%)	FLOPs	FLOPs↓ (%)
GA	0.03	2.35	84.00	2.74×10^8	56.20
GAL-0.05	1.93	3.36	77.60	1.89×10^8	39.60
GAL-0.1	3.18	2.67	82.20	1.72×10^8	45.20
LWM	0.13	5.40	64.00	0.21×10^8	34.20
CFP-1	0.26	–	–	0.62×10^8	80.36
CFP-2	0.51	–	–	0.57×10^8	81.93
Baseline	0	15.27	0	3.39×10^8	0
FSA	**0.19**	**1.76**	**88.47**	$\mathbf{0.298\times10^8}$	**91.20**

"acc↓" is the percentage decrease of the accuracy. The data of FSA are in bold

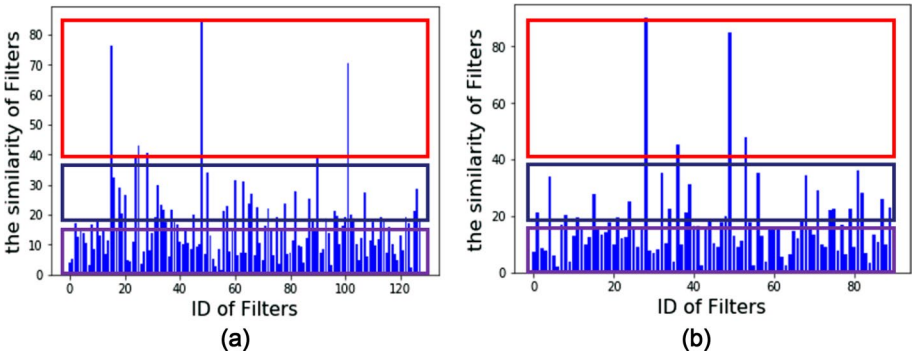

Fig. 6 The filter similarity distribution of the fourth convolutional layer of VGG-16. **a** and **b** show the distributions before and after pruning, respectively. The filters with their similarity falling in a color rectangle form a set. The sets become sparse after pruning. The x-axis represents the ID of filters, from 0 to the number of filters. Although the number of filters decreases after pruning, the distributions before and after pruning are similar

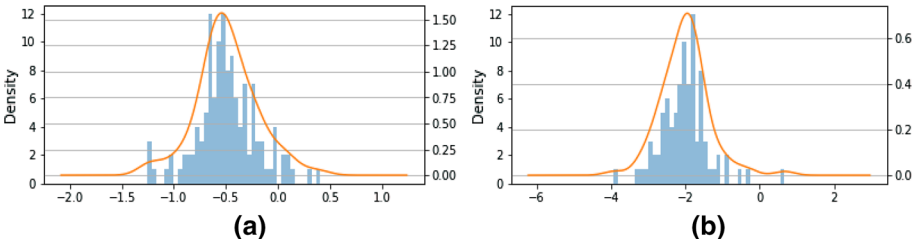

Fig. 7 The filter weight density value of VGG-16 before and after pruning. The filter distribution generally satisfies normal distribution. The x-axis represents the range of weights, the y-axis is the density statistics, and the scale on the right is the equidistant division

VGG_2 is 81.95%. The experimental results show that the network structure obtained by FSA has a higher accuracy than the same-scale networks do.

4.4.2 The forward and backward pruning on VGG-16

In addition to the different pruning methods, the order of pruning is also an important factor affecting the accuracy of a network. We conduct both pruning forward and backward on VGG-16 trained on CIFAR-10 dataset using the same parameter setting. The results are shown in Table 4 for backward pruning, and Table 6 for forward pruning.

There is no pruning data of the tenth layer in Table 6 because the accuracy of Top-1 drops significantly after pruning. In order to ensure the loss of accuracy of the network is under the threshold, the tenth layer is not pruned.

The Fig. 8 shows the variation of Top-1 accuracy when pruning each layer for both forward and backward pruning. Top-1 accuracy of backward pruning is 85.51%, which is better than that of forward pruning.

Table 6 The results of Forward Pruning VGG-16

CONV	Filters	Top-1	Top-5	Para(M)	Para↓ (%)	FLOPs	FLOPs↓ (%)
Baseline	4224	85.70	99.05	15.27	0	3.39×10^8	0
Conv_2(64)	16	85.62	99.11	15.18	0.59	2.58×10^8	23.89
Conv_3(128)	42	85.51	98.98	15.07	1.31	2.56×10^8	24.49
Conv_4(128)	29	84.73	99.05	14.81	3.01	2.51×10^8	25.96
Conv_5(256)	84	82.55	98.63	14.36	5.96	2.44×10^8	28.02
Conv_6(256)	96	82.53	98.67	13.87	9.17	2.36×10^8	30.38
Conv_7(256)	80	78.61	98.01	12.91	15.46	2.19×10^8	35.39
Conv_8(512)	160	82.38	98.80	11.03	27.77	1.87×10^8	48.84
Conv_9(512)	175	82.55	98.96	8.99	41.13	1.53×10^8	54.87
Conv_11(512)	184	81.93	98.71	5.97	61.10	1.01×10^8	70.21
Conv_12(512)	164	82.02	98.56	3.79	74.00	0.64×10^8	81.12
Conv_13(512)	130	76.13	98.10	3.02	80.22	0.51×10^8	84.96

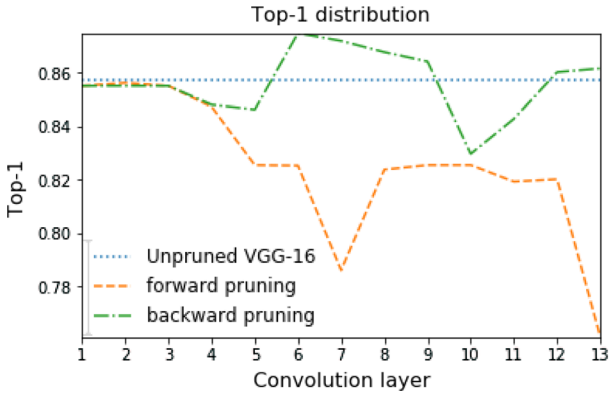

Fig. 8 The variation of Top-1 during pruning. The x-axis is the convolutional layer of VGG-16, and the y-axis is Top-1 after pruning each layer. After the fifth convolutional layer is pruned in forward pruning, the Top-1 drops significantly, and Top-1 of the last layer is reduced 9.57%, the final Top-1 of the network is 76.13%. Top-1 of the 10th layer during backward pruning starting from the 13th convolutional layer reduces by a maximum of 2.53%, and the final Top-1 is 85.51%, which is only 0.19% lower than that of the baseline

4.4.3 VGG-16 on KAGGLE

We train the VGG-16 on different data sets. The VGG-16 is trained from scratch as baseline on Kaggle datasets. The network is trained with batch size of 32 for 100 epochs and the binary-cross-entropy loss is used as criterion function. Meanwhile, the optimizer uses RMSProp (Root Mean Square Prop).

The experimental results are shown in Fig. 9. The number of parameters of the baseline is 19.19 M, the Top-1 is 95.8% and the error rate is 4.4%. After the second convolutional layer is cut out, the number of parameters is 2.12 M and the Top-1 is 97%. After the first convolutional layer is pruned, the number of parameters only reduces by 0.01 M, while the Top-1 is 95.3% and reduces by 1.7%. Therefore, the network can be cut in backward direction until the second convolutional layer to achieve the optimal

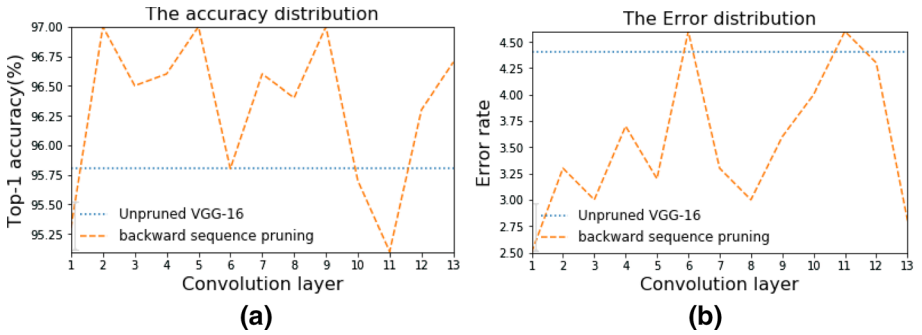

Fig. 9 The accuracy and error rate distribution. The broken line represents the variation of Top-1 accuracy and error rate during the pruning process. It can be seen from **a** that after the 11th layer is pruned, the Top-1 is reduced by up to 0.7%. Finally, Top-1 of the first layer is 95.3%, which is 0.5% lower than the unpruned VGG-16. **b** shows the error rate change of pruning. The final network has an error rate of 2.5%

balance between CR and accuracy. The error rate of the network was finally 2.5%, which was 1.9% lower than the baseline model, and the CR was 5.6.

4.5 The results of pruning ResNet-50

We use the ImageNet dataset to train Resnet-50 and randomly selected 100 categories. The network is trained with batch size of 32 for 1000 epochs and the cross-entropy loss is used as criterion function. Meanwhile, the optimizer uses SGD and the regularization term is L2. We selected the first two layers of standard convolution in each residual block for pruning. The number of parameters decreased by 33.26% and the FLOPs dropped by 59.91%, which is better than the pruning method of CFP, GAL, ThiNet (Luo et al., 2017), and SSS (Huang & Wang, 2018), as shown in Table 7.

Table 7 The Results of Pruning ResNet-50 on ImageNet

Method	acc ± (%)	Para↓ (%)	FLOPs↓ (%)
GAL-0.5	− 4.20	16.86	43.03
GAL-1	− 6.27	42.47	61.37
ThiNet-50	− 5.14	51.45	58.19
ThiNet-30	− 7.73	66.04	73.11
SSS-26	− 4.33	38.82	43.03
SSS-32	− 1.97	27.06	31.05
CFP	− 0.80	–	49.60
FSA	**+ 0.37**	**33.26**	**59.91**

"acc ± (%)" represents the change in accuracy. The effect of our method on accuracy is positive, and the accuracy after fine-tuning is 0.37 higher than that of the baseline model. "ThiNet-30″ and"ThiNet-50″ represent retaining 30% and 50% filters in each block, respectively. "SSS-26″ and"SSS-32″ represent ResNet with a depth of 26 and 32, respectively. The data of FSA are in bold

5 Conclusion

The proposed method prunes filter by taking the similarity between filters into account. Only the part of filters that extracts significant feature is retained. Backward pruning ensures the stability and accuracy of the network during the pruning process. Experimental results show that the method is effective for the pruning of sequential convolutional stack networks and superior to the state-of-the-art pruning methods. The investigation of the effectiveness of our pruning method to the neural networks with different structures will be our future work.

Acknowledgements This work was supported by National Key Research and Development Plan of China No. 2017YFB1401000 and National Key Research and Development Plan of Shanxi Provence No. 201903D421007.

References

Anwar, S., & Sung, W. (2016). Retrieved August 20, 2020 from *Coarse pruning of convolutional neural networks with random masks*. https://openreview.net/forum?id=HkvS3Mqxe

Han, S., Mao, H., & Dally, W. J. (2015). *Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding*. arXiv Preprint arXiv:1510.00149

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR) (pp. 770–778).

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). *Mobilenets: Efficient convolutional neural networks for mobile vision applications*. arXiv preprint arXiv:1704.04861

Hu, H., Peng, R., Tai, Y. W., & Tang, C. K. (2016). *Network trimming: A data-driven neuron pruning approach towards efficient deep architectures*. arXiv preprint arXiv:1607.03250

Hu, Y., Sun, S., Li, J., Wang, X., & Gu, Q. (2018). *A novel channel pruning method for deep neural network compression*. arXiv preprint arXiv:1805.11394

Huang, Z., & Wang, N. (2018). Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 304–320).

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016). *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size*. arXiv preprint arXiv:1602.07360

Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105). Curran Associates, Inc.

Lecun, Y., & Bottou, L. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278–2324. https://doi.org/10.1109/5.726791

Li, H., Kadav, A., Durdanovic, I., Samet, H., & Graf, H. P. (2017). *Pruning filters for efficient convnets*. arXiv preprint arXiv:1608.08710

Lin, S., Ji, R., Yan, C., Zhang, B., Cao, L., Ye, Q., Huang, F., & Doermann, D. (2019). Towards optimal structured CNN pruning via generative adversarial learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 2785–2794).

Luo, J., & Wu, J. (2017). *An entropy-based pruning method for CNN compression*. arXiv preprint arXiv:1706.05791

Luo, J. H., Wu, J., & Lin, W. (2017). *ThiNet: A filter level pruning method for deep neural network compression*. arXiv preprint arXiv:1707.06342

Ma, N., Zhang, X., Zheng, H. T., & Sun, J. (2018). ShuffleNet V2: practical guidelines for efficient CNN architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, Munich, Germany (pp. 116–131).

Mittal, D., Bhardwaj, S., Khapra, M. M., & Ravindran, B. (2019). Studying the plasticity in deep convolutional neural networks using random pruning. *Machine Vision and Applications*. https://doi.org/10.1007/s00138-018-01001-9

Molchanov, P., Tyree, S., Karras, T., Aila, T., & Kautz, J. (2016). *Pruning convolutional neural networks for resource efficient inference*. arXiv preprint arXiv:1611.06440

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 4510–4520).

Simonyan, K., & Zisserman, A. (2014). *Very deep convolutional networks for large-scale image recognition*. arXiv preprint arXiv:1409.1556

Singh, P., Verma, V. K., Rai, P., & Namboodiri, V. P. (2018). *Leveraging filter correlations for deep model compression*. arXiv preprint arXiv:1811.10559

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 1–9).

Yang, T. J., Chen, Y. H., & Sze, V. (2017). Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 6071–6079).

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 818–833). Springer.

Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 6848–6856).

Authors and Affiliations

Lili Geng[1,2] · Baoning Niu[1]

Baoning Niu
niubaoning@tyut.edu.cn

[1] School of Information and Computer, Taiyuan University of Technology, Taiyuan, Shanxi, People's Republic of China

[2] Experimental Center, Shanxi University of Finance and Economics, Taiyuan, Shanxi, People's Republic of China