International Journal of Computational Intelligence Systems (2024) 17:129
https://doi.org/10.1007/s44196-024-00518-4

RESEARCH ARTICLE

APSSF: Adaptive CNN Pruning Based on Structural Similarity of Filters

Lili Geng[1,2] · **Baoning Niu**[1]

Received: 25 October 2023 / Accepted: 7 May 2024

Abstract
Convolutional neural network (CNN) pruning is a technique used to remove redundant parameters from the network. By doing so, it aims to greatly reduce the computational complexity and scale of the network while still preserving its accuracy. In the CNN, the majority of parameters are weights that form filters. When it comes to pruning, it is more effective to focus on removing redundant filters rather than insignificant weights within filters. The essence of filter pruning lies in determining the significance or contribution of each filter. Filters that have a significant contribution are kept, while others are pruned. Current methods for calculating contribution in pruning often rely on weight magnitude or filter similarity. However, approaches based solely on assume that small weights are unimportant and ignore correlation between filters, which leads to a significant loss of network accuracy. Those based on filter similarity flatten filter tensors into a vector when calculating filter similarity, and lose the important structural information of filters, or the superposition information of the weight convolution in the corresponding space position. These limitations can compromise the accuracy and effectiveness of the pruning process. This paper proposes an adaptive CNN pruning method based on the structural similarity of filters (APSSF) by taking both the structural characteristics of and the correlation between filters into the consideration for pruning filters. APSSF efficiently calculates the distance between the filters by factoring in information from all the dimensions of filters, and clusters the filters according to the distance threshold determined adaptively according to the compression rate, and deletes a certain number of filters from each category. On the CIFAR10 and ImageNet datasets, APSSF outperforms several state-of-the-art methods. On the CIFAR100, APSSF reduces parameters of networks by 91.71% and 74.80% on VGG-16 and ResNet-34, respectively. The accuracy was decreased only by 0.03 on VGG-16, while on ResNet-34, it was increased by 0.04.

Keywords Convolutional neural networks · Filter pruning · Clustering · Similarity

1 Introduction

Convolutional neural networks (CNN) have remarkable achievements in the field of computer vision in recent years, especially in image recognition and classification. CNNs are specifically designed to handle the spatial structure of images and exploit the local correlations between pixels. This makes CNNs well-suited for tasks that involve analyzing and understanding visual data. As the scale of data and the structure of models continue to expand, the parameter count and computational complexity of CNNs show an exponential growth trend, bringing enormous challenges to model training and inference. These models typically require significant computational resources and datasets for training, and in practical applications, they often require high computational performance and storage resources for inference and deployment. For instance, popular CNN architectures like VGG [1], ResNet [2], and Xception [3] have demonstrated impressive accuracy of over 90% on large datasets. CNN models have achieved excellent performance in many computer vision tasks and have become the benchmark models for some tasks. These models have undergone extensive research and validation, demonstrating high reliability and stability, and can serve as benchmark models for comparison with other models. When evaluating new models or algorithms, using these benchmark models can provide a

✉ Baoning Niu
niubaoning@tyut.edu.cn

Lili Geng
genglili0049@link.tyut.edu.cn

[1] College of Computer Science and Technology (College of Data Science), Taiyuan University of Technology, Yuci, Shanxi, People's Republic of China

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

Published online: 23 May 2024

basis for reference and comparison. The VGG and RESNET benchmark models are widely used in many computer vision tasks and have become the benchmark models for these tasks. Many of the current large models have been developed and expanded based on VGG and RESNET. For example, models based on RESNET, such as RESNEXT [4] and Wide RESNET [5], have improved performance by increasing the model's width, depth, or introducing new model structures. These large models still adhere to some design principles of VGG and RESNET, and have been innovated and optimized based on them. However, the large number of parameters in these models leads to excessive memory consumption and computational complexity. This becomes a hurdle for deploying CNN on resource-constrained devices and limits their wide adoption. Compared to large models, using small models may be more suitable for specific tasks. For example, a small model can be used for training a medical question-answering system to provide accurate medical knowledge and answers; for specific language pairs in machine translation tasks, small models can be trained. Small models can be customized according to the grammar structure and cultural characteristics of the language to provide more accurate translation results; for specific domain named entity recognition tasks, such as in medicine and finance, training with small models can improve the accuracy and recall rate of named entity detection. To address this challenge, model compression has emerged as a research focus. The aim is to reduce the size of the model while maintaining its accuracy, thus enabling deployment on resource-constrained devices. Achieving model compression has therefore become crucial in promoting the widespread utilization of CNN.

CNN are feedforward neural networks composed of an input layer, convolutional layers, pooling layers and fully-connected layers. A convolutional layer consists of a number of filters which extract features of images. A filter is a high-order tensor [6] structured with weights. CNN with multiple convolutional layers usually have thousands of filters, inevitably redundant, to get good identification and classification performance. Deleting redundant parameters or filters does not affect the model accuracy, but also accelerates model training [7]. It is easier to obtain high-accuracy by training a pruned network [8] than by training a small model from start.

Pruning CNN involves two conflicting goals: minimizing the number of parameters or filters while maximizing model accuracy. The key to successful pruning lies in finding the right balance between model scale and accuracy. Since the majority of parameters in CNN are concentrated within filters, filter pruning is an effective approach for compressing CNN. Filter pruning entails selectively removing a certain number of filters in each convolutional layer. This approach is considered more effective and interpretable compared to weight pruning, which involves removing selected weights within a filter [9]. The key to filter pruning is to determine the contribution of a filter. There are two categories of filter pruning according to the discriminant rule of contribution, pruning based on the weight magnitude and pruning based on the filter similarity.

Methods that rely on weight magnitude often establish pruning criteria based on the weights of a filter and the statistics that affect the loss function. These methods remove filters either by assigning a score to a filter based on its weights, assuming that small weights are not important, or by measuring the contribution of a filter using the probability distribution of its weights. However, this assumption does not always align with the actual outputs of the model. To further investigate this assumption, we conducted an experiment on the VGG-16 architecture. We calculated the sum of filter weights and proceeded to delete the filters with 50% smallest and largest weights based on the sum size. The results revealed an 8.10% decrease in model accuracy after removing the filters with small weights, while a 0.40% increase in accuracy was observed after removing the filters with large weights. These findings indicate that not all small weights can be considered unimportant, highlighting the limitations of solely relying on weight magnitude as the pruning criterion.

In addition, these methods overlook the importance of structural similarity among filters. The structural similarity refers to the discrepancy generated by the different spatial positions of the tensor structure of the filter to convolute the input data. Filters that convolve at corresponding spatial positions of an image tend to exhibit similarity due to the similarity of pixel structures in small areas. Mapping filter weights to scalars can lead to the loss of valuable similarity information. By disregarding the structural similarity between filters, these methods fail to fully utilize the inherent relationships within the model. It highlights the need for an approach that takes into account the structural characteristics of filters while effectively pruning the network to achieve optimal results.

Methods that are based on filter similarity are generally more reasonable than those relying solely on weight magnitude since they take into account the structural characteristics of filters. However, it is worth noting that many of these methods tend to flatten filter tensors into vectors when calculating filter similarity. This flattening process leads to the loss of crucial information related to the superposition of weights in the corresponding spatial positions during convolution. By flattening the filter tensors, important spatial information is disregarded, which can impact the accuracy and effectiveness of the pruning process. It is necessary to explore alternative methods that can preserve the structural information of filters and capture the full potential of their contributions in the model.

To conclude, the disadvantages of the methods for filter pruning are

- Methods based on weight magnitude treat filters as independent entities, potentially overlooking the structural similarity between them. Mapping filter tensors into scalars can lead to the loss of important structural information.
- Methods based on filter similarity usually flatten the filter tensor into a vector when calculating their similarity. The superimposed information of weight convolution in the corresponding space position is not considered, and the structural information of the filter tensor is lost when calculating the similarity.
- Most methods determine the number of classify using a fixed threshold based on similarity. The fixed threshold needs to be determined through multiple tests, and the calculation is time-consuming. And the fixed threshold cannot obtain the optimal classification.

To address these limitations, there is a need for developing advanced pruning methods that consider the structural information of filters, preserve their superimposed information, and offer more flexible and efficient ways to determine the optimal number of filters to prune.

In view of the above problems, we exploit the structural characteristics of filters to differentiate filters, and propose an adaptive CNN pruning method based on the structural similarity of filters (APSSF). The core concept of APSSF is to leverage the structural characteristics of filters to differentiate and identify filters for pruning, thereby reducing redundant parameters while diversifying feature extraction. The key idea of this approach is to recognize that filters are interconnected entities in the feature space. Filters that contribute little to feature extraction are identified and pruned based on their similarities to other filters. By pruning these less -significant filters, the model can maintain its accuracy while reducing its complexity. To compute filter similarity efficiently, we reduce the dimensionality of weight tensors while preserving important channel structural information. This helps in clustering the filters in a meaningful way. During clustering, the number of clusters, or the distance threshold, is determined adaptively according to the parameters of the compression rate using Augmented Lagrange method.

By utilizing the structural similarities of filters and employing adaptive pruning techniques, APSSF aims to achieve significant parameter reduction while maintaining model accuracy, making it a valuable contribution to the field of CNN pruning. The contributions of this paper are as follows:

- A CNN pruning method called APSSF is proposed to find similar filters based on the structural characteristics of filters. By calculating filter similarity and clustering filters accordingly, APSSF selects and retains a specific number of filters within each category.
- The parameter of compression rate is introduced to regulate the rate at which filters are pruned. The distance threshold used to determine the number of clusters is adaptively calculated using the Augmented Lagrangian optimization method. This ensures optimal and flexible pruning based on the desired compression rate.
- An efficient method is proposed for calculating the similarity between filters based on filter tensors. The method compresses filter tensors to reduce their dimensions while maintaining important structural information. The similarity between filters is then measured based on the compressed tensors, ensuring the preservation of the triangular inequality of distance even after compression.

2 Related Work

The process of filter pruning is as follows: (1) Training the original CNN: Initially, the original CNN is trained on the target dataset to establish a baseline performance; (2) Sorting filters: Filters are sorted according to some criterion, such as weight magnitude, filter similarity, or a combination of factors. This sorting process helps identify the filters that will potentially be pruned; (3) Retaining top-ranked filters: A certain number of filters, typically those ranked at the top of the sorted list, are selected to be retained. These filters are considered to have the most significant contributions to the model and its accuracy; (4) Fine-tuning the pruned CNN: Finally, the pruned CNN, which consists of the selected filters after the previous step, undergoes a fine-tuning process. This fine-tuning aims to reoptimize the model to achieve the same or even higher levels of accuracy as the original CNN.

In filter pruning, accurately determining the contribution of filters is crucial. There are two common approaches, weight magnitude based and filter similarity based. Weight magnitude-based approaches establish pruning criteria based on the filter weights and the impact on the loss function. These methods often calculate the importance of a filter by assuming that larger weights are more significant than smaller weights. The representative method is proposed by Li et al. [10]. It uses L1 norm, the sum of the absolute values of the weights of a filter, to determine the important contribution of a filter. Filters with large L1 norm are retained, while those with small L1 norm are removed. Another method by He et al. [11] implements a geometric median-based technique to identify and prune redundant filters. In addition, Liu et al. [12] imposes L1 regularization on the scaling factor in the batch normalization (BN) layer. The value of the BN scaling factor then approaches to zero, pruning the channel of the small scaling factor.

The loss function is not only affected by filters with large weights but also by those with small weights. Therefore, determining the contribution of a filter solely based on the statistics that affect the loss function is a more reasonable approach compared to directly calculating scores from filter weights. Methods such as ThiNet proposed by Luo et al. [13] utilize statistics computed from the next layer to remove filters in a layer. This approach captures the impact of filters on the loss function by considering their influence on subsequent layers. Molchanov et al. [14] utilize Taylor expansion to approximate the loss function and identify filters with low impact, which are then pruned. Building upon this work, Molchanov et al. [15] later improves their work using the first-and second-order Taylor expansions to approximate the contribution of the filter. It iteratively removes those filters with smaller scores by estimating the contribution of a filter to the final loss. Yang et al. [16] use energy consumption as its criterion to prune CNN. While these methods do not rely on the assumption that smaller weights are unimportant, they ignore the similarities among filters.

Although these methods do not rely on the assumption that smaller weights are unimportant, they tend to overlook the similarities among filters. Capturing filter similarities can further enhance the pruning process by taking into account preserving critical structural information.

The filter similarity-based approach focuses on differentiating filters using their spatial attributes. Recognizing that filters are interconnected entities in space, this approach aims to identify filters with minimal contribution to feature extraction, ultimately removing them based on similarity discrimination. The similarity measurement and clustering method vary among different methods employing this approach. Commonly used similarity measurements include Euclidean distance, cosine similarity, or normalized cross-correlation (NCC) similarity. These measurements convert the three-dimensional filter into a one-dimensional vector without simplifying the computation. For example, Chu et al. [17] measure filter similarity using the Euclidean distance metric, resulting in a compact model with minimal accuracy loss after removing highly similar filters. Shao et al. [18] focuses on the similarity between filters or feature maps in the same layer. They use cosine similarity to measure the similarity between channels. MSVFP [9] combines filter magnitude and filter similarity to determine the importance of filters.

The frequently used clustering method is k-Means clustering, where the number of clusters or similarity threshold is typically fixed. Li et al. [19] use the k-Means++algorithm to enforce filters into a specific cluster. The filter closest to the center of a cluster is retained, the others are removed. A fixed threshold is set to determine the number of clusters. CSHE [18] uses k-Means to cluster filters with a fixed number of clusters. ICP [20] utilizes the DBSCAN clustering algorithm to cluster feature maps, and channel pruning is performed according to the number of clusters.

One common limitation of these pruning methods is that they flatten the filter tensors into vectors, resulting in the loss of weight convolution superposition information in the corresponding spatial positions. In addition, these methods rely on fixed thresholds for determining the number of clusters, which often require multiple tests to find the optimal value. To overcome these limitations, further research can explore methodologies that preserve the superposition information of weight convolution and adopt adaptive methods for determining the number of clusters, enhancing the efficiency and effectiveness of filter pruning algorithms.

Our proposed method differs from existing state-of-the-art approaches in two key aspects: (1) Consideration of spatial characteristics: Our method places particular emphasis on exploiting the spatial characteristics of filter tensors, paying close attention to the superposition properties within the dimensions corresponding to the channel of filters. By taking into account this valuable information, our method aims to preserve crucial structural details and improve the overall effectiveness of filter pruning; (2) Adaptive determination of clustering threshold: In contrast to previous methods that rely on a fixed threshold for clustering, our approach introduces an adaptive mechanism to determine the clustering threshold. This addresses the limitations associated with fixed thresholds, which often require extensive trial and error to reach optima. By adaptively determining the clustering threshold, our method aims to overcome such challenges and achieve more accurate and efficient filter pruning.

3 Adaptive CNN Pruning

APSSF is based on structural similarity of filters to prune filters in CNN. The core of APSSF is to find an appropriate method for measuring the similarity between filters, while achieving adaptive filter clustering. Section 3.1 introduces the third-order weight tensor of filters used for computing filter similarity. Section 3.2 defines filter similarity and discusses the efficient calculation method. Section 3.3 introduces the filter clustering method. Section 3.4 discusses the adaptive CNN clustering pruning.

3.1 The Weight Tensor of Filters

CNN is a hierarchical network model that consists of data input, convolutional layer, pooling layer, activation function, fully-connected layer, and output. Filters of the convolutional layer produce a large number of parameters through convolution operations. The need to apply filters in image processing is due to the abundance of redundant and irrelevant information contained in the image. Filters,

also known as the convolution kernel, are used to extract meaningful features from the images. These filters are small tensors that are convolved with the input image to generate a feature mapping. By applying filters to images, we can capture important visual patterns and structures. These patterns can represent various characteristics of the image, such as edges, textures, and shapes. Filters help to highlight these features and suppress irrelevant information, making it easier for the network to learn and make accurate predictions.

The application of filters in CNNs reduces the complexity of image processing in several ways:

(1) Parameter sharing: In CNNs, filters are shared across the entire image or feature map. This parameter sharing significantly reduces the number of parameters compared to fully connected networks, where each neuron is connected to every input. By sharing parameters, CNNs can capture local patterns and generalize them across the entire image, leading to more efficient and compact models.
(2) Translation invariance: Filters in CNNs are designed to be translation invariant, meaning they can detect the same pattern regardless of its location in the image. This property allows CNNs to effectively handle variations in object position and scale, reducing the complexity of image processing.
(3) Hierarchical feature extraction: CNNs typically consist of multiple layers, with each layer learning increasingly complex and abstract features. The filters in the early layers capture low-level features like edges and textures, while filters in the deeper layers capture high-level features like object shapes and semantic information. This hierarchical feature extraction reduces the complexity of image representation and enables the network to learn more discriminative features.

Understanding the structure of filters is the first step for measuring filter similarity.

The filter is a third-order tensor, $W = [W]_{n\times k_h\times k_r}$.$k_h \times k_r$ is 3×3, 5×5 or 7×7 in general, and called the receptive field of a filter, which is symmetric according to the central pixel. n is the number of channels. There are two types of n in CNNs.

(1) In input layer, n is determined by the type of the input image, for RGB images, n=3, and for black and white images, $n=1$.
(2) n is equal to the number of filters in other layers, and is also the input channel of the next layer after the convolution output.

Many filters constitute a convolutional layer, which is represented by a fourth-order tensor, $L = [L]_{n\times k_h\times k_r\times m}$. m is the number of filters of a convolutional layer. As show in Fig. 1.

Convolution is the mathematical operation for two real variable functions [21]. The convolution operation is represented by "$*$".

$$f(t) = (x * w)(t) \tag{1}$$

In CNN terminology, the first parameter x of the convolution is usually called the input, and the second parameter w is called the filter weight. The output $f(t)$ is called the feature map.

The pixels in a local area of the input image are convolved into each corresponding pixel in the output matrix, where the element of the matrix is the weight. The convolution operation is shown in Fig. 2 and Eq. (1). x, w and o represent the input, the filter weight and the output, respectively.

$$\begin{aligned} O_{12} = {} & x_{12}\cdot w_{11} + x_{13}\cdot w_{12} + x_{14}\cdot w_{13} \\ & + x_{22}\cdot w_{21} + x_{23}\cdot w_{22} + x_{24}\cdot w_{23} \\ & + x_{32}\cdot w_{31} + x_{33}\cdot w_{32} + x_{34}\cdot w_{33} \end{aligned} \tag{2}$$

Convolution is that the filter slides on the input data from the upper left corner, and multiplies and sums with the corresponding position data to get an output value. The filter then moves to the right to do the same operation. And so on, from left to right, from top to bottom, to get the feature map of the filter output.

Filters act on the local area of an image to obtain the local features through the convolution operation. A filter containing n channels forms an output channel by aggregating the n feature maps produced by convoluting along the height and width directions. Each dimension of the filter tensor represents different information. This is the reason why computing a score from a tensor loses dimensional information. Convolution is the dot product of the tensor slice and the pixels of the corresponding image area. Because a filter slides a small step (a pixel) on an image when convoluting, the image pixels of two adjacent sliding windows are usually

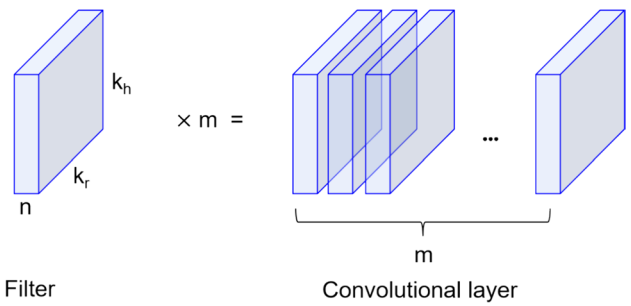

Fig. 1 Structure of convolution layer

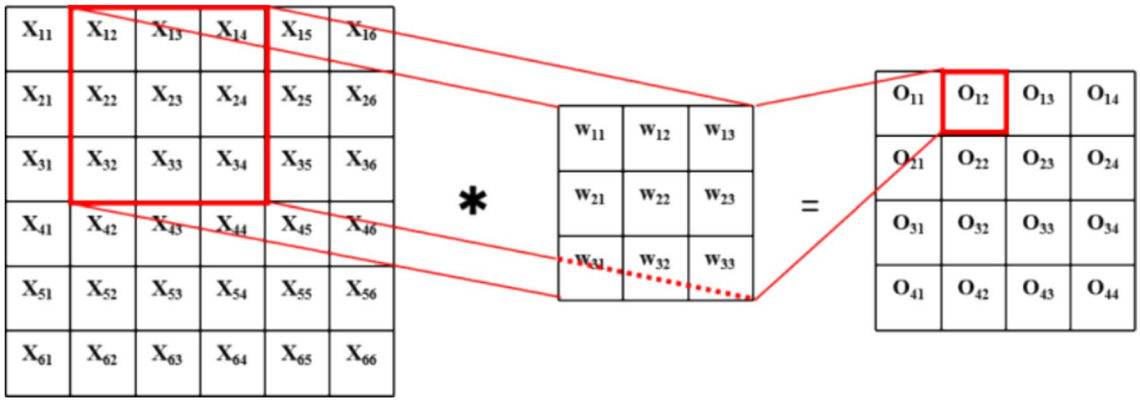

Fig. 2 Convolutional operation

highly similar. The filters that perform convolution operation on the same spatial position of the image are also high similar. Mapping a filter tensor to a scalar ignores the differences caused by different spatial positions. We introduce how to calculate the filter similarity in the next section.

3.2 The Calculation of Filter Similarity

The filters in shallow convolutional layers are responsible for extracting basic structural features, such as texture features, from the input data. In contrast, filters in deeper layers of the model are designed to capture more abstract and semantic features, which are combinations of the basic structural features. As filters in different convolutional layers focus on extracting different categories of features, it becomes meaningful to compare the similarity of filters within each layer.Therefore, we compute the similarity of filters in each convolution layer independently and pruning layer by layer. By evaluating the similarity of filters at each layer, we can prune the network layer by layer, considering the specific characteristics and contributions of the filters within that layer. This approach ensures a more targeted and effective pruning process, as filters within the same layer are expected to have similar roles and provide redundant information.

Because the weight tensor has a large number of parameters, the computational complexity of filter similarity is $O\left(2^{nk_hk_r}\right)$, where the parameters are the three dimensions of the weight tensor. It is necessary to find an approach to efficiently calculating the similarity.

The filter similarity is defined by Eq. (3).

Definition: Filter similarity (FS). The similarity between two filters (W, W') is

$$FS^{(WW')} = \frac{1}{\sum_n \sum_{k_h} \sum_{k_r} \left|[W]_{n\times k_h\times k_r} - [W]'_{n\times k_h\times k_r}\right|} \tag{3}$$

where, $[W]_{n\times k_h\times k_r}$ represents the weight tensor of a filter. n, k_h, andk_r are the width, height and length of the weight tensor, respectively. $\sum_n \sum_{k_h} \sum_{k_r} \left|[W]_{n\times k_h\times k_r} - [W]'_{n\times k_h\times k_r}\right|$ is the distance of filters.

Computing the distance between two filters requires performing a three-layer nested loop.

$$\text{for } (i = 1; i \le n; i++)$$

$$\text{for } \left(j = 1; j \le k_h; j++\right)$$

$$\text{for } \left(k = 1; k \le k_r; k++\right)$$

$$\sum_n \sum_{k_h} \sum_{k_r} \left|[W]_{n\times k_h\times k_r} - [W]'_{n\times k_h\times k_r}\right|$$

$[W]$ and $[W]'$ have $2^{nk_hk_r}$ states, respectively. The calculation complexity of the filter similarity is $O\left(2^{nk_hk_r}\right)$. To obtain an efficient pruning algorithm, we employ dimension reduction to reduce the computational complexity. While it is possible that dimension reduction may alter the similarity values between filters, it is important to note that our objective is not to preserve the absolute similarity values, but rather to maintain the relative similarity relationships between filters. The primary aim of dimension reduction is to identify a subset of filters that exhibit similar characteristics. By focusing on the similarity relationships, rather than the exact similarity values, we can effectively reduce the computational complexity while still capturing the important structural information within the filters.

By employing dimension reduction, we aim to retain the essential similarity relationships between filters, allowing us to efficiently execute the pruning process. This approach enables us to strike a balance between computational efficiency and effective filter pruning.

One way to reduce the computational complexity is to reduce the dimensions of the weight tensor and then calculate the distance. We reduce both the height k_h and width k_r of the tensor to 1, preserving the channel n dimension. After tensor dimension reduction, the weight tensor becomes a vector containing the input channel information. The method of finding the average value is used to compress $k_h \times k_r$ to 1×1 and reduce a convolutional layer $L = [L]_{n \times k_h \times k_r \times m}$ to a two-dimensional matrix $F = [F]_{m \times n}$. The weight tensor is transformed into the weight vector $\vec{W} = \left(w_1, w_2, \ldots w_n\right)$ after tensor dimension reduction. The sum of weights is

$$\sum \vec{W}_n = \frac{1}{k_h \times k_r} \sum \sum \sum [W]_{n \times k_h \times k_r} \tag{4}$$

The weight of the ith filter is represented as $\vec{W}_j = (w_{i1}, w_{i2}, \cdots, w_{in})$. We have

$$F = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix}$$

Each row of matrix F represents a filter, and each column represents a channel dimension of the filter. w_{ij} represents the value of the jth dimension of the ith filter. Filter similarity can be derived from the distance between the row vectors using Eq. (5).

$$FS^{\left(\vec{W}_i \vec{W}_j\right)} = \frac{1}{D_{ij}} = \frac{1}{d\left(\vec{W}_i, \vec{W}_j\right)} \tag{5}$$

$\vec{W}_i$ and $\vec{W}_j$ are the weights of the ith and jth filters. $d(\cdot)$ is the distance function, which is Manhattan distance and easy to calculate.

$$d\left(\vec{W}_i, \vec{W}_j\right) = \sum \left|\vec{W}_i - \vec{W}_j\right| = \sum_n \left|w_{in} - w_{jn}\right| \tag{6}$$
$$\text{s.t.} 0 \le \text{i} \le \text{m}, 0 \le \text{j} \le \text{m}$$

From Eqs. (5) and (6) we have

$$\begin{aligned} FS^{\left(\vec{W}_i \vec{W}_j\right)} &= \frac{1}{\sum \left|\vec{W}_i - \vec{W}_j\right|} = \frac{1}{\frac{1}{k_h \times k_r} \sum \sum \sum \left|[W_i] - [W_j]\right|} \\ &= \left(k_h \times k_r\right) FS^{(W_i W_j)} \end{aligned} \tag{7}$$

This means dimension reduction did not change the similarity relationship between two filters. For three filters $W_i, W_j,$ and W_k, if $FS^{(W_i W_j)} < FS^{(W_i W_k)}$, we have $\left(k_h \times k_r\right) FS^{(W_i W_j)} < \left(k_h \times k_r\right) FS^{(W_i W_k)}$, which is $FS^{\left(\vec{W}_i \vec{W}_j\right)} < FS^{\left(\vec{W}_i \vec{W}_k\right)}$. This means that the similarity relationship among a set of filters is not changed. Only the channel dimension is left after the dimension reduction. The computational complexity of the similarity of m filters is $O(m2^n)$, which is orders of magnitude efficient than before dimension reduction.

3.3 Filter Clustering

With the efficient calculation of filter similarity, we are now ready to discuss filter clustering. Filter clustering involves partitioning the filters into different categories, where filters within the same category exhibit a high degree of similarity. The process of the clustering includes the following steps: (1) Initialization: each filter is initially considered as a separate category. The Manhattan distance between pairs of filters is computed, resulting in a distance matrix, $D = d_{ij}$; (2) Merge categories: The two categories with the minimum distance are merged into a new category; (3) Calculate average distance: the average distance between the new category and the other categories is calculated; (4) Iteration: Steps 2 and 3 are repeated until the clustering ends. More details can be seen in Algorithm 1.

The C_{num} represents the number of clusters formed during the filter clustering process. It indicates the total number of categories or clusters into which the filters have been partitioned based on their similarity. C_P and C_q indicate two categories containing p and q filters, respectively. G_{pq} is the category distance and calculated by averaging of the filter distances across the two categories as described by Eq. (8).

$$G_{pq} = \frac{1}{pq} \sum_{i \in C_p} \sum_{j \in C_q} D_{ij} \tag{8}$$

The output of Algorithm 1 is a set of categories representing the clustered filters, as well as the number of clusters that have been formed. However, to successfully conduct filter clustering, it is necessary to determine either the distance threshold or the number of clusters required.

In our approach, we make use of the distance threshold as the end condition for the algorithm. The distance threshold can be adaptively adjusted based on the desired compression rate. This provides a flexible and efficient way to control the pruning process and achieve the desired trade-off between model size reduction and accuracy preservation.

Using the distance threshold, our method ensures that the pruning process is adaptive and can be fine-tuned according to specific requirements. This allows for a more nuanced approach to filter clustering, as it adjusts the threshold based on the desired compression rate, leading to effective pruning and improved model efficiency.

Algorithm 1 Filter clustering

Input: The weight tensor of filters; W
Output: The set of categories: $C = \{C_1, C_2, \ldots, C_k\}$; The number of clusters: C_{num}.
Initialize an empty list C to store the set of categories.
Initialize C_{num} as 0, representing the initial number of clusters.
For each filter j from 1 to m (total number of filters):
    Initialize C[j] as a separate category containing only filter j.
    Increment C_{num} by 1.
Create a distance matrix M of size m × m.
For each filter i from 1 to m:
    For each filter j from i+1 to m:
        Compute the distance between filters i and j using the distance function d and store it in M[i, j].
        Set M[j, i] equal to M[i, j] #since the distance matrix is symmetric.
Perform the merge process:
    For each filter i from 1 to m:
        For each filter j from i+1 to m:
            While M[i, j] > distance threshold:
                Merge the closest categories C_i and C_j into a new category.
                Update C[i] by combining the filters from C_i and C_j.
                Decrement C_{num} by 1.
                Update the remaining categories C_{j+1} to C_{num}.
                Delete the row and column corresponding to filter j in the distance matrix M.
                Update the distance matrix M by calculating the average distance between the new category and the other categories.
Return the set of categories C and the number of clusters C_{num}.

Algorithm 1 initializes each filter as a class to create a distance matrix, calculates the distance between the filters, merges the nearest class greater than the distance threshold, and updates the distance matrix iteratively until the end of the clustering. Algorithm 1 contains two nested loops, the time complexity of the first loop is $O(m)$, and the time complexity of the second loop is $O(m^2)$, so the overall time complexity is $O(m^2)$, where m is the number of filters. In addition, the algorithm also includes the calculation of distances between filters and clustering operations, the time complexity of these operations depends on the specific distance calculation method and clustering algorithm. Therefore, the time complexity of Algorithm 1 can be represented as $O(m^2)$ or high-order complexity.

For the selection of the Manhattan distance, it is a commonly used metric for measuring similarity in filter pruning because it is a suitable method for comparing filter responses. The Manhattan distance is advantageous for measuring filter similarity because it considers the absolute differences between elements of two filters. This characteristic allows it to effectively capture the structural similarity between filters by focusing on their respective filter weight values. In contrast, other distance metrics such as Euclidean distance or cosine similarity may emphasize overall distance or angle between filters, which may not represent their structural similarity as effectively. Using the Manhattan distance, the method can better distinguish filters with similar structural characteristics, facilitating more accurate clustering and pruning.

In Algorithm 1, the use of the Manhattan distance as the distance metric for merging clusters is based on the following characteristics and principles:

Manhattan distance is a simple and intuitive distance metric. It measures the distance between two vectors by calculating the sum of the absolute differences of their corresponding elements. This distance metric is easy to understand and compute.

Manhattan distance is suitable for handling high-dimensional data. In convolutional neural networks, weight vectors are typically high-dimensional, and therefore, the Manhattan distance can effectively measure differences between different weight vectors.

Manhattan distance can provide better clustering effects when merging clusters.Using the Manhattan distance, similar weights can be clustered together, leading to better weight-pruning effects. This is because the Manhattan distance can capture the absolute differences between weights, making it more likely for similar weights to be merged together.

In summary, the selection of the Manhattan distance as the distance metric for merging clusters is based on its simplicity, applicability, and robustness to outliers. Using the Manhattan distance, better weight clustering and pruning effects can be achieved, thereby improving the performance of the adaptive clustering pruning algorithm.

3.4 Adaptive Pruning

Adaptive CNN pruning is based on the filter clustering, removing the filters in the same category, and retaining only one of them. The number of clusters determines the size of the pruned network. The problem solved by CNN pruning is minimizing the loss and the number of clusters. We define the problem as an optimization over the filter distances that incorporates conflicting desires of minimizing the loss and minimizing the number of distances (clusters).In the context of adaptive CNN pruning, the process involves filter clustering, where filters belonging to the same category are pruned, and only one filter is retained as a representative. The number of clusters directly impacts the size of the pruned network. The key problem addressed by CNN pruning is to find an optimal balance between minimizing the loss incurred by the pruning process and minimizing the number of distances or clusters produced.

To tackle this problem, we define it as an optimization task over the filter distances, taking into account the conflicting objectives of loss minimization and cluster maximization. The goal is to find an optimal configuration that simultaneously reduces the network size while preserving accuracy.

We can solve this problem by introducing Lagrange multipliers and penalty terms to construct the Augmented Lagrangian Function. The Augmented Lagrangian Function is a powerful tool that can help us update parameters during the iteration process, leading to faster convergence to the optimal solution.

First, we need to understand the concepts of Lagrange multipliers and penalty terms. Lagrange multipliers are auxiliary variables used to construct the Augmented Lagrangian Function, and they help maintain the convexity of the objective function during the optimization process. Penalty terms are additional terms that can enforce certain constraint conditions in the optimization problem.

When constructing the Augmented Lagrangian Function, we incorporate Lagrange multipliers and penalty terms into the cost function, resulting in an optimization problem that contains more information. The solution to this optimization problem will help us find the optimal solution to the original problem.

We define the cost function for clustering as $P : R^n \to R^+$, $P(d) = \sum_{i=1}^{m} p\left(d_i\right)$ satisfies $P(0) = 0$ and $P(d) > 0$ *if* $d \neq 0$, where d is the filter distance and $P(d)$ is the number of filter distances. Equation (9) shows the loss function $L(d)$ in the constraint form.

$$min_d L(d)\ s.t. P(d) \leq c \tag{9}$$

We use the Augmented Lagrangian method to transform it into a constrained optimization problem. By introducing Lagrange multipliers and penalty terms, the Augmented Lagrangian Function, as shown in Eq. (10), is constructed. In the process of solving the Augmented Lagrangian Function, an iterative computation method is employed. Equations (11–16) represent the iterative process, describing the specific steps and update rules for using the Augmented Lagrangian method to solve constrained optimization problems. Through iterative computation, the optimal solution d is obtained. During the computation, λ is calculated using the update formula and θ is computed based on the updated λ. Ultimately, the relatively optimal filter distance d and variable θ are obtained. These optimal solutions will help us achieve better performance in practical problems.

Given a variable θ, $d - \theta = 0$, $\theta \geq 0$, and satisfying $P(\theta) \leq c$, we have

$$\mathcal{L}(d, \theta, \lambda, \mu) = L(d) - \sum_{i=1}^{m} \lambda_i \left(d_i - \theta_i\right) + \mu \sum_{i=1}^{m} \left(d_i - \theta_i\right)^2 \tag{10}$$

Equation (10) represents the Augmented Lagrangian Function, where $L(d)$ is the original loss function, λ is the Lagrange multiplier, and μ is the penalty parameter. The objective of this function is to transform the constrained optimization problem into an unconstrained optimization problem by introducing Lagrange multipliers and penalty terms.

The optimal d can be obtained by $d^* = \underset{\mathrm{d}}{\operatorname{argmin}} L(d)$. λ_i^k are calculated by the update formula from Eqs. (11) to (13). k is calculated by Eq. (14). After k iterations, d^k is the optimal solution.

$$\begin{aligned} \nabla_d \mathcal{L}\left(d^k, \theta^k, \lambda^k, \mu_k\right) =& \nabla P\left(d^k\right) - \sum_{i=1}^{m} \lambda_i^k \nabla \left(d_i^k - \theta_i^k\right) \\ &+ \sum_{i=1}^{m} \left(d_i^k - \theta_i^k\right)^2 = 0 \end{aligned} \tag{11}$$

$$\nabla P\left(d^k\right) - \sum_{i=1}^{m} \left(\lambda_i^k - 2\mu_k \sum_{i=1}^{m} d_i^k \right) \nabla \sum_{i=1}^{m} \left(d_i^k - \theta_i^k\right) = 0 \tag{12}$$

$$\lambda_i^{k+1} := \lambda_i^k - 2\mu_k \sum_{i=1}^{m} d_i^k, i = 1, \cdots, m \tag{13}$$

$$\begin{aligned}\left(d^k,\theta^k\right) &= arg\min_{d,\theta}\mathcal{L}_\mu(d,\lambda) = arg\min_{d,\theta}P(d)+\sum_{i=1}^{m}\left\{-\lambda_i\left(d_i-\theta_i\right)+\mu\left(d_i-\theta_i\right)^2\right\}\\ &= arg\min_{d,\theta}P(d)+\mu\sum_{i=1}^{m}\left\{\left(d_i-\theta_i-\frac{\lambda_i}{\mu}\right)^2\right\}\\ s.t.\,\theta_i &\geq 0, i=1,\cdots,m\end{aligned} \tag{14}$$

From the Eq. (14) we have

$$\theta_i^k = max\left(d_i^k-\frac{\lambda_i}{\mu},0\right) \tag{15}$$

$$d^k = \arg\min_d p(d)+\sum_{i=1}^{m}\Psi\left(d_i,\lambda_i,\mu\right)$$

$$\Psi\left(d_i,\lambda_i,\mu\right) = \begin{cases}-\lambda_i d_i+\mu d_i^2 & \text{if } d_i-\frac{\lambda_i}{\mu}<0,\\ -\frac{\lambda_i^2}{\mu} & \text{otherwise}\end{cases}. \tag{16}$$

Equation (11) represents the update of parameters d and θ in each iteration by computing the gradient of the Augmented Lagrangian Function to minimize the function.

Equation (12) calculates the next iteration of the Lagrange multiplier based on the gradient of the Augmented Lagrangian Function and the update rule for the Lagrange multiplier.

Equation (13) provides the update rule for the Lagrange multiplier to compute the value for the next iteration.

Equations (14) and (15) describe how the variables are updated in each iteration to minimize the Augmented Lagrangian Function. The max function in Eq. (15) ensures that the value of θ satisfies the constraint.

Equation (16) defines an auxiliary function used to compute the update process for parameter d. Different calculation methods are chosen based on different conditions to ensure that the updated parameter d satisfies the constraint.

In summary, Eqs. (10–16) describe how the Augmented Lagrangian method updates the parameters d and θ in each iteration and how the Lagrange multiplier is updated to solve constrained optimization problems.

We use the Augmented Lagrangian optimization method to calculate d and θ, as shown in Algorithm 2. d and θ are the relative optimal solution. We set the compression rate parameter to control the filter pruning rate and, in the adaptive CNN pruning algorithm, obtain the pruned optimal solution. Algorithm 2 contains a loop, the number of iterations depends on the quantity of μ. In each iteration, distance minimization calculation and some simple mathematical operations are required. Therefore, the time complexity of Algorithm 2 mainly depends on the complexity of distance minimization calculation, which can be represented as O(k), where k is the quantity of μ.

Algorithm 2 The Augmented Lagrangian method

Input: Initial weight tensor W, distance function d, constraint c
Output: Optimal solution d
Initialize the iteration count k, learning rate μ, Lagrange multiplier λ
Initialize d as the initial weight tensor W.
While k < maximum number of iterations do :
Update λ as λ - 2μ * ∑d.
Update θ as max(d - λ/μ, 0).
Update d as argmin P(d) + ∑Ψ(d, λ, μ).
Increment k by 1.
Return the optimal solution d.

Adaptive CNN pruning framework consists of two key components: setting the parameters of the compression rate and determining the variable distance threshold (d).

(1) Parameters of the Compression Rate

The compression rate serves as an evaluation metric for network compression. It represents the ratio of the initial number of filters in the network to the number of filters remaining after pruning. However, pruning filters based on a fixed compression rate alone may not result in the desired network performance, and determining appropriate thresholds can be challenging.

To address this, we introduce a parameter of the compression rate (δ) to control and maintain the compression rate within a specified range during the pruning process. By adjusting δ, we gain better control over the compression level, allowing for more fine-tuned and expected performance outcomes.

The parameter δ plays a crucial role in the process of filter clustering. It is primarily used to dynamically adjust the distance threshold, thereby controlling the number of clusters. By cleverly adjusting δ, we can effectively control the granularity and quantity of clusters, making them more adaptable to actual requirements. First, let us delve into the impact of δ on the granularity and quantity of clusters. When δ is small, it leads to more clusters. This is because a smaller δ means that we have stricter requirements for similarity measurements within the dataset, and only data points with small distances will be assigned to the same cluster. As a result, the number of clusters will

increase, and the number of data points in each cluster will be relatively small. However, when δ is large, it may result in fewer but more widespread clusters. A larger δ means that we have looser requirements for similarity measurements within the dataset, and data points with larger distances will also be assigned to the same cluster. As a result, the number of clusters will decrease, but the number of data points in each cluster will be relatively large. In conclusion, by adjusting this key parameter δ, we can make the filter clustering method adaptable to different application scenarios. In practical applications, choosing the appropriate δ value is crucial because it directly affects the quality and effectiveness of clustering. In this way, the filter clustering method can provide us with more flexible and efficient clustering services.

By precisely adjusting δ, a more accurate trade-off between compression rate and performance preservation can be achieved. A smaller δ may lead to a higher compression ratio but could sacrifice some performance, while a larger δ may preserve more information, aiding in maintaining the model's performance. Therefore, by adjusting the value of δ, the optimal adjustment parameter δ can precisely influence the trade-off between compression rate and performance preservation according to specific application requirements and performance demands. The specific impacts are as follows:

Factors affecting the compression rate: When δ is small, the similarity threshold is low, resulting in more filters being aggregated into the same category, thereby increasing the number of filters retained after pruning and improving the compression rate. Conversely, when δ is large, the similarity threshold is high, leading to fewer filters being aggregated into the same category, reducing the number of filters retained after pruning and lowering the compression ratio.

Factors affecting performance preservation: When δ is small, due to the retention of more filters, the model's performance may be relatively better as more parameters and features are preserved, but it may also increase computational and storage overhead. Conversely, when δ is large, due to the retention of fewer filters, the model's performance may be less affected as the model's complexity and storage requirements decrease, but it may also lose some feature information, leading to performance degradation.

Balancing compression rate and performance preservation: Based on specific application requirements and performance demands, the compression rate and performance preservation can be balanced by adjusting δ. If a higher compression rate is required, a smaller δ can be chosen to retain more filters and improve the compression rate. If higher performance preservation is required, a larger δ can be chosen to preserve more feature information and reduce performance loss.

Therefore, by adjusting the parameter δ, the trade-off between compression rate and performance preservation can be precisely influenced. Based on specific application requirements and performance demands, suitable values of δ can be flexibly chosen to achieve the best compression effect and performance preservation results.

In general, the number of pruned filters is set to 50–75% of the original number. The ratio of the total number of filters F_{num} to the parameter δ represents the range of the number of pruned filters, F_{num}/δ. If $\delta = 2$, F_{num}/δ represents the compression rate of 50%; $F_{\text{num}}/2\delta$ represents the compression rate of 75%. The number of clusters $C_{\text{num}} : C_{\text{num}} = P(d)$ satisfies the inequality $F_{\text{num}}/\delta < C_{\text{num}} < F_{\text{num}}/2\delta$. That is, C_{num} is between 50% and 75% of the number of filters, so that the number of filters preserved according to clusters satisfies the range of the compression rate.

(2) Determination of the Variable Distance Threshold (d)

The variable distance threshold (d) plays a crucial role in the filter clustering process. It determines the similarity threshold for merging filters into clusters. To tackle the challenge of selecting an optimal threshold, we employ the augmented Lagrangian optimization method. By iteratively adjusting and optimizing d, we can dynamically determine the appropriate number of clusters based on specific performance requirements.

APSSF does not use the fixed distance threshold to determine the number of clusters, instead, automatically adjusts the distance threshold during clustering according to the parameters of the compression rate. Let d be the distance threshold and determine whether the number of clusters is within the range of compression rate during the clustering. During each clustering iteration, when the number of the cluster is greater than F_{num}/δ, the compression rate is lower than 50%, and the number of clusters is too large, so the d is increased. When the number of clusters is less than $F_{\text{num}}/2\delta$, the compression rate is higher than 75%, and the d should be reduced. The change rate for d is set to ϑ. We use $\vartheta = 0.001$ according to the experimental results. We retain the first filter in each class C_i according to the filter clustering results.

The adaptive determination of the variable distance threshold (d) is a key aspect in the adaptive clustering pruning algorithm, as it determines the conditions for merging categories. Choosing the appropriate distance

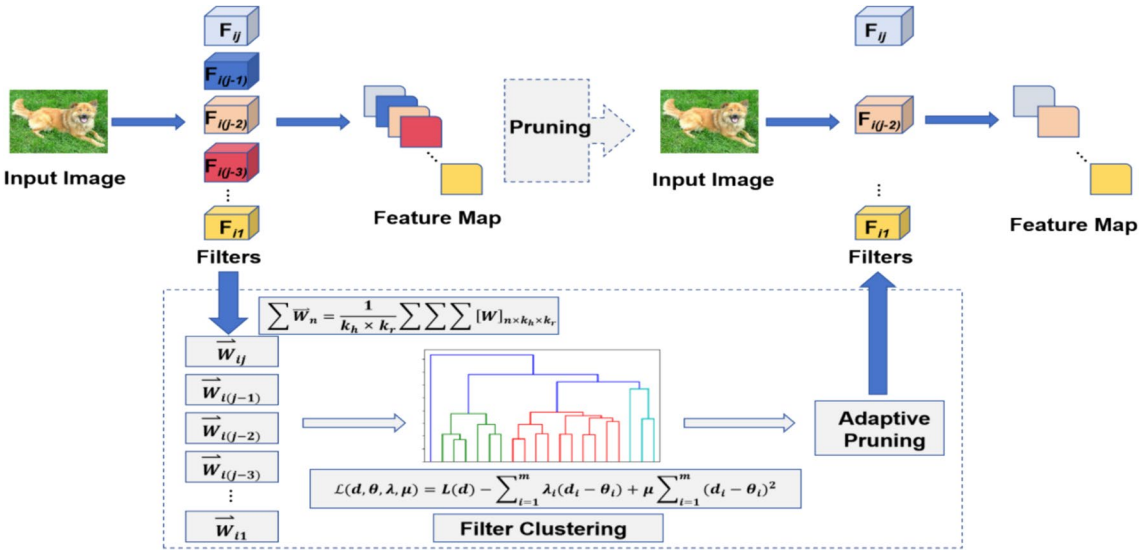

Fig. 3 The overview of the APSSF method. The APSSF method first performs dimensionality reduction on the filter tensor F, reducing it to a vector W. Then, it automatically determines the similarity distance based on the Augmented Lagrangian method and compression rate parameter, achieving adaptive clustering, and conducts pruning based on the clustering results

threshold can affect the clustering effect and the extent of pruning. The change of d was determined by the ϑ, the following are the reasons for choosing $\vartheta = 0.001$ and basic principles:

The value of ϑ should be small enough to merge similar weights. A smaller ϑ value can ensure that only very close weights are merged together, thereby maintaining a higher clustering quality. At the same time, the value of ϑ should not be too small to avoid over-merging weights, leading to information loss and performance degradation. A larger ϑ value may lead to over-pruning, thereby affecting the model's performance. Through experimentation and accumulated experience, the value of $\vartheta = 0.001$ has been found to provide good clustering and pruning effects in many cases. This value has been proven to be a reasonable choice in practice. It is important to note that the specific value of d may vary due to differences in datasets, tasks, and models. Therefore, choosing the appropriate value of ϑ requires adjustment and optimization based on specific circumstances.

In summary, the selection of $\vartheta = 0.001$ is based on considerations of clustering quality and pruning effects. This value is determined through experiment and experience, and can be considered a reasonable choice for the adaptive clustering pruning algorithm. However, adjustments and optimizations may be necessary to achieve the best results for different datasets and tasks.

Adaptive CNN pruning algorithm is shown in Algorithm 3. Algorithm 3 contains multiple calls to Algorithm 1 and Algorithm 2, so its time complexity depends on the number of these calls. In the worst case, the time complexity of Algorithm 3 may be relatively high, depending on factors such as the number of filters and clusters.

Algorithm 3 Adaptive CNN pruning

Input: Initial weight tensor W, distance function d, initial distance threshold d_threshold, compression rate parameter δ, the change rate ϑ.
Output: Pruned weight tensor W_pruned.
Initialize the maximum number of iterations max_iter, learning rate μ, Lagrange multiplier λ, iteration count k as 0, the change rate ϑ=0.001.
Initialize d as the initial distance threshold d_threshold.
While k < max_iter do:
Use the Algorithm 2 to compute the optimal solution d.
Determine the range of pruned filter numbers based on the compression rate parameter δ: num_filters = F_{num} / δ.
Perform Algorithm 1 based on the optimal solution d, obtaining the clustered filter set C and the number of clusters C_num.
If C_{num} > num_filters * 2, increase d_threshold by d=d+ϑ.
If C_{num} < num_filters, decrease d_threshold by d=d-ϑ.
Increment k by 1.
Retain the first filter in each category C according to the filter clustering results, obtaining the pruned weight tensor W_pruned.
Return the pruned weight tensor W_pruned.

The overview of the proposed APSSF method is presented in Fig. 3.

The APSSF pruning method achieves a high compression rate and reduces a large number of parameters. Due to the adoption of efficient similarity calculation and clustering methods, the time complexity of the APSSF algorithm is relatively low, which can improve training speed. For example, in the training of VGG-16 with Epoch = 200 and batch = 128, the training time before pruning was 5800s, and after pruning, the training time decreased to 3000s, resulting in a reduction of 48.28% in training time.

The combination of parameterized compression rates and adaptive determination of the distance threshold in our APSSF framework facilitates effective and efficient filter pruning. This approach not only provides improved control over network compression but also ensures that the desired performance outcomes are achieved.

The pruned model resulting from the application of APSSF is referred to as the indicator model. In Sect. 4 of the paper, we design an original model with identical depth, width, number of filters, and structure as the indicator model. By training both the indicator model and the small original models from scratch, we compare their performances.

Remarkably, the experimental results show that the indicator model significantly outperforms the small original models in terms of performance. This observation demonstrates the effectiveness and superiority of the APSSF-based pruning method in producing a pruned model that retains superior performance compared to its smaller, original counterparts.

These findings highlight the benefits of employing APSSF for filter pruning, as it not only preserves performance but also achieves better results compared to smaller networks designed from scratch. Thus, APSSF proves to be a powerful approach for achieving efficient network compression without compromising on performance.

The adaptive CNN pruning algorithm has some relationships and differences with existing pruning methods. The adaptive CNN pruning algorithm is a method that integrates the ideas of weight pruning and neuron pruning. It achieves filter pruning by clustering similar filters effects by merging categories with adaptive distance thresholds. The adaptive CNN pruning algorithm can be seen as an improved pruning method, as it introduces adaptability based on traditional pruning methods, making it better suited for different datasets and tasks.

The adaptive CNN pruning algorithm has certain advantages in terms of computational efficiency. Using clustering to merge similar filters, it can reduce computational and storage requirements. Compared to traditional pruning methods, the adaptive CNN pruning algorithm can complete the pruning process more quickly. In addition, the algorithm can further improve computational efficiency through parallel computing. Parallel computing can be used to accelerate the execution of the algorithm when merging categories and pruning filters.

The adaptive CNN pruning algorithm also has certain advantages in maintaining performance. By selecting distance thresholds reasonably, the algorithm can effectively prune while maintaining high model performance. This is because it can preserve important filters, thereby reducing

the impact on model performance. Furthermore, the adaptive CNN pruning algorithm can further improve performance through fine-tuning. After pruning, the model can be retrained using fine-tuning techniques to recover or enhance its performance.

4 Experiments

This section focuses on evaluating the effectiveness of APSSF by introducing the evaluation indicators, datasets, models, and performing an analysis of the experimental results. First, we present the evaluation indicators that were used to assess the performance of the pruned model. (See the Sect. 4.1). These indicators could include the accuracy, the number of parameters and the floating-point operations. Next, we describe the models used in the experiments. This can include details about the architecture, number of layers, filter sizes, and other relevant specifications. The original model and the indicator model pruned using APSSF are compared in terms of their performance and efficiency (See the Sects. 4.2 and 4.3). Finally, we analyze the experimental results obtained from evaluating the indicator network pruned by APSSF. This analysis may involve comparing its performance with that of the original model and the small, newly designed models. (See the Sect. 4.4).

4.1 Evaluation Indicator and Datasets

The evaluation of pruning CNN involves several indicators, including accuracy, the number of parameters, and the floating-point operations. Here is a brief explanation of each indicator:

(1) Accuracy (Acc): Acc is an important indicator to measure the performance of a model. Pruning inevitably causes performance degradation of the model and reduces the accuracy. A good pruning method not only has little impact on accuracy, but also even can restore accuracy after fine-tuning.

Its calculation formula is:

$$\mathrm{Acc} = \frac{S_C}{S_T} \tag{17}$$

S_C is number of correctly classified samples, S_T is total number of samples.

Here, the number of correctly classified samples refers to the quantity of samples that the model accurately classified during the prediction process, while the total number of samples refers to the overall number of samples in the dataset. Accuracy is a crucial metric for assessing the overall precision of a model, providing a comprehensive evaluation of the model's classification ability for each category.

During the training process, Accuracy can be used to monitor the model's performance. By calculating the Accuracy at the end of each training epoch, it is possible to understand the model's classification accuracy on the training set and adjust the model's parameters accordingly to improve performance. Furthermore, Accuracy can also be used to evaluate the classification accuracy of different models on the test set, enabling comparisons of different model performance.

In summary, as a comprehensive performance evaluation metric, Accuracy is crucial for assessing the classification accuracy of a model across the dataset, providing important information about the model's overall performance.

Fig. 4 Example diagram of the input image

(2) Number of Parameters (Parameter): This indicator quantifies the size of the model by counting the total number of parameters. Pruning aims to reduce the number of parameters to achieve model compression while maintaining performance.

Assuming the input data size is $N \times N$, the filter size is $F \times F$, the number of input data channels is C_{in}, and the number of filters is C_{out}, the calculation formula for the number of parameters is as follows:

$$\text{Number of Parameters} = C_{\text{in}} \times C_{\text{out}} \times F \times F \tag{18}$$

Here, $C_{\text{in}} \times F \times F$ represents the number of parameters for each filter, and C_{out} represents the number of filters.

In a convolutional layer, each filter has $C_{\text{in}} \times F \times F$ parameters, and there are a total of C_{out} filters, thus the number of parameters is the product of these two quantities.

It is important to note that this parameter count only considers the parameters of the convolutional layer and does not take into account the parameters of other types of layers, such as fully connected layers.

(3) Floating-Point Operations (FLOPs): FLOPs measure the computational complexity and speed of model operations. Decreasing the number of FLOPs helps to clarify the computational efficiency and resource requirements of a model. We want decrease of FLOPs for a model would make this parameter clearer.

In CNN, FLOPs encompass the operations of convolutional layers, pooling layers, and fully connected layers. The calculation formulas are as follows:

Calculation formula for FLOPs in convolutional layers:

Assuming the size of the input feature map is $H \times W$, the size of the convolutional kernel is $K \times K$, the number of input channels is C_{in}, and the number of output channels is C_{out}, the FLOPs calculation formula for a convolutional layer is:

$$\text{FLOPs}_{\text{Conv}} = 2 \times H \times W \times C_{\text{in}} \times C_{\text{out}} \times K \times K \tag{19}$$

Calculation formula for FLOPs in pooling layers:

Assuming the size of the pooling layer is $P \times P$ and the number of input channels is C_{in}, the FLOPs calculation formula for a pooling layer is:

$$\text{FLOPs}_{\text{pooling}} = H \times W \times C_{\text{in}} \times K \times K \tag{20}$$

Calculation formula for FLOPs in fully connected layers:

Assuming the input feature dimension of the fully connected layer is D_{in} and the output feature dimension is D_{out}, the FLOPs calculation formula for a fully connected layer is:

$$\text{FLOPs}_{\text{Conv}} = 2 \times D_{\text{in}} \times D_{\text{out}} \tag{21}$$

By considering the FLOPs calculation formulas for the three types of layers mentioned above, it is possible to determine the total FLOPs for the entire CNN model. Calculating FLOPs allows for the assessment of model computational complexity, enabling evaluations of model efficiency on different hardware platforms. This is crucial for deploying models on resource-constrained devices or carrying out model optimization. The number of parameters and FLOPs can be used to assess model complexity, providing important references for model selection and design. The number of parameters reflects the model's scale and learning capacity, while FLOPs measure the model's computational cost and efficiency. Striking a reasonable balance between them can enhance the model's performance and efficiency.

The experiments are performed using the Keras platform, a popular deep learning framework. Two well-known CNN architectures, VGG-16 and ResNet-34/50, are selected for evaluation. CIFAR10 [22] and CIFAR100 [23] datasets are used on VGG-16. ResNet-50 uses CIFAR10, CIFAR100, and ImageNet [24]. The CIFAR10 has 60,000 color images of 32×32 pixels with 50,000 training images and 10,000 test images for a total of 10 classifications. CIFAR100 is similar to CIFAR10, which has 100 classifications, with 500 training images and 100 test images in each classification. ImageNet about 1.2 million training images, 50,000 validation images, 150,000 test images and 1000 class tags. As show in Fig. 4.

These datasets are widely used in the research community and provide diverse and comprehensive evaluation scenarios for the pruning process. By evaluating the accuracy, parameter count, and FLOPs of the pruned models on these datasets, we can successfully demonstrate the effectiveness and efficiency of the APSSF pruning method.

Table 1 Comparison of VGG-16 on CIFAR10 pruning methods

Method	Acc ±	Parameter (M)	Parameter ↓ (%)	Flops ↓ (%)
GAL ($\lambda = 0.05$)	+1.93	3.36	77.60	39.60
GAL ($\lambda = 0.1$)	+3.18	2.67	82.20	45.20
GA	+0.03	2.35	84.00	56.20
LWM	+0.13	5.40	64.00	34.20
CSHE	−0.02	2.64	82.10	69.00
ICP (ε=0.010)	−0.40	1.94	86.83	70.66
ICP (ε=0.015)	−0.22	0.91	93.82	79.94
ICP (ε=0.020)	+0.31	0.54	96.33	86.20
APSSF-2	−0.02	2.39	84.35	95.38
APSSF-4	−0.03	1.19	92.21	97.72

Table 2 Performance of VGG-16 with $\delta = 2$ & $\delta = 4$ on CIFAR10

Conv_X	APSSF-2				APSSF-4			
	Acc ±	Parameter (M)	Parameter ↓ (%)	Flops ↓ (%)	Acc ±	Parameter (M)	Parameter ↓ (%)	Flops ↓ (%)
Conv_13	0.00	13.46	11.85	74.08	−0.01	13.23	13.36	74.50
Conv_12	−0.02	11.03	27.77	78.71	−0.01	11.05	27.64	78.72
Conv_11	−0.06	9.59	37.20	81.50	0.00	8.82	42.24	83.01
Conv_10	−0.01	6.76	55.73	86.97	−0.03	6.59	56.84	87.32
Conv_9	0.00	5.29	65.36	89.80	0.00	4.38	71.32	91.57
Conv_8	0.00	3.99	73.87	92.32	+0.02	3.04	80.09	94.16
Conv_7	−0.01	3.42	77.60	93.41	+0.02	2.38	84.41	95.42
Conv_6	0.00	2.97	80.55	94.28	0.00	1.82	88.08	96.49
Conv_5	−0.02	2.67	82.51	94.86	−0.01	1.49	90.24	97.14
Conv_4	−0.02	2.52	83.50	95.14	−0.02	1.32	91.36	97.46
Conv_3	−0.01	2.43	84.09	95.28	−0.01	1.24	91.88	97.62
Conv_2	−0.03	2.41	84.22	95.36	−0.05	1.20	92.14	97.70
Conv_1	−0.02	2.39	84.35	95.38	−0.03	1.19	92.21	97.72

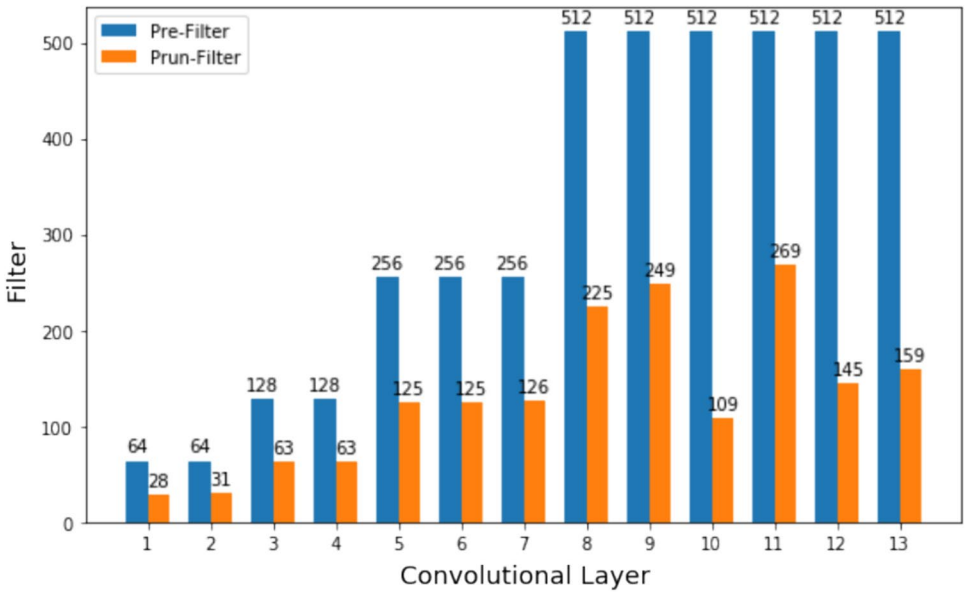

Fig. 5 The number of filters for APSSF-2 on the CIFAR10 in VGG-16

4.2 Pruning VGG-16

We perform the algorithm validation on VGG-16. VGG-16 is a classic deep learning model. The structure of the VGG-16 includes the following components:

13 convolutional layers, where each filter has a receptive field size of 3×3;

3 fully connected layers;

5 pooling layers, all using 2×2 maximum pooling.

It is important to note that both the convolutional layers and fully connected layers contain parameters, also referred to as weight layers. All of the pooling layers have no parameters. Experiments are performed on CIFAR10 and CIFAR100 datasets with input 3-channel images of 32×32.

By evaluating the APSSF pruning algorithm on VGG-16, specifically on the CIFAR10 and CIFAR100 datasets, we can assess its effectiveness in preserving accuracy and reducing model parameters for the given architecture and datasets.

4.2.1 Pruning VGG-16 on CIFAR10

In our experiments, we set the parameter of compression rate (δ) to 2 (APSSF-2) and 4 (APSSF-4), respectively. δδ is set manually based on the desired level of model compression and performance preservation. These values were chosen considering the trade-off between model size reduction and accuracy preservation. During the experiment, when the

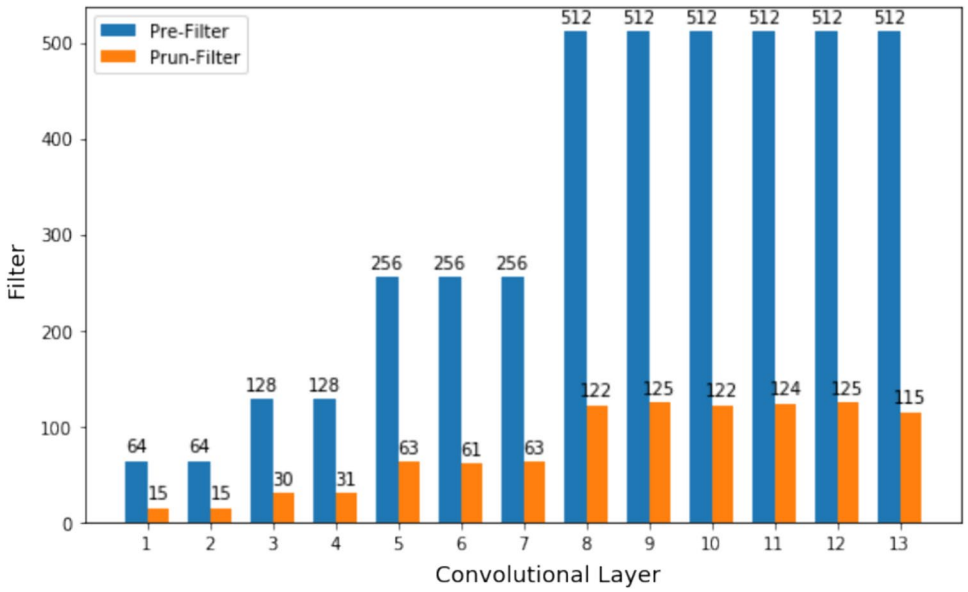

Fig. 6 The number of filters for APSSF-4 on the CIFAR10 in VGG-16

compression rate parameter is less than 2, the compression rate is small, while when the compression rate parameter is greater than 4, the accuracy loss is large. Therefore, the two values of 2 and 4 were finally selected. The results obtained from evaluating APSSF-2 and APSSF-4 on the CIFAR10 dataset are as follows:

APSSF-2: The accuracy achieved by APSSF-2 is 84.26% on CIFAR10, which is only slightly lower (0.02) than the accuracy of the original model. The number of parameters is reduced by 84.35%, resulting in a significant decrease in model size. The FLOPs are reduced by 95.38%, indicating a substantial improvement in computational efficiency.

APSSF-4: The accuracy of APSSF-4 is 83.23%, which is 0.03 lower than the original model. The number of parameters is decreased by 92.21%, indicating a significant reduction in model size. Moreover, the FLOPs are reduced by 97.72%, demonstrating a substantial improvement in computational efficiency.

When comparing APSSF-2 and APSSF-4, both methods provide considerable reductions in model parameters and FLOPs compared to the original model. However, APSSF-2 achieves slightly better accuracy compared to APSSF-4, indicating its effectiveness in preserving model performance.

Table 1 shows a performance comparison of APSSF with other pruning methods, further illustrating the superior performance and efficiency of APSSF.

These experimental results demonstrate that APSSF outperforms existing pruning methods, achieving significant reductions in model complexity while simultaneously maintaining a reasonable level of accuracy on the CIFAR10 dataset.

Based on the comparison presented in Table 1, APSSF is compared with other state-of-the-art pruning methods, including GAL [25], GA [16], LWM [10], CSHE [18], and ICP [20]. APSSF achieves minor loss in accuracy compared to the original model. The accuracy reduction for APSSF-2 is only 0.02, and for APSSF-4, it is 0.03. Although the accuracy of GA, LWM, GAL are increased, the strength of the model pruning is less than APSSF-4, the reduction rates of parameters and FLOPs are also much lower than that of APSSF-4. APSSF outperforms other methods in terms of parameter and FLOPs reduction rates. APSSF-4 achieves the highest reduction rates, indicating its strong pruning capability. GAL, GA, LWM, CSHE, and ICP show less pronounced reductions in parameters and FLOPs compared to APSSF-4.

Taken together, the comparison demonstrates that APSSF achieves a better balance between accuracy and pruning strength compared to other methods. It effectively preserves accuracy while achieving substantial reductions in parameters and FLOPs. The compression rate of FLOPs in APSSF surpasses that of GAL, GA, LWM, CSHE, and ICP, further illustrating its efficiency in model compression.

Detailed performance changes of APSSF-2 and APSSF-4 are presented in Table 2. The tables reveal that, at different pruning strengths, both APSSF-2 and APSSF-4 show a notable loss in accuracy when pruning the first and second convolutional layers. This suggests that these two layers have limited parameter redundancy, resulting in accuracy reductions without significant reductions in model size.

Based on these findings, it can be inferred that pruning the first and second convolutional layers may not achieve a favorable balance between model size and accuracy. Thus, it may be more effective to refrain from pruning these

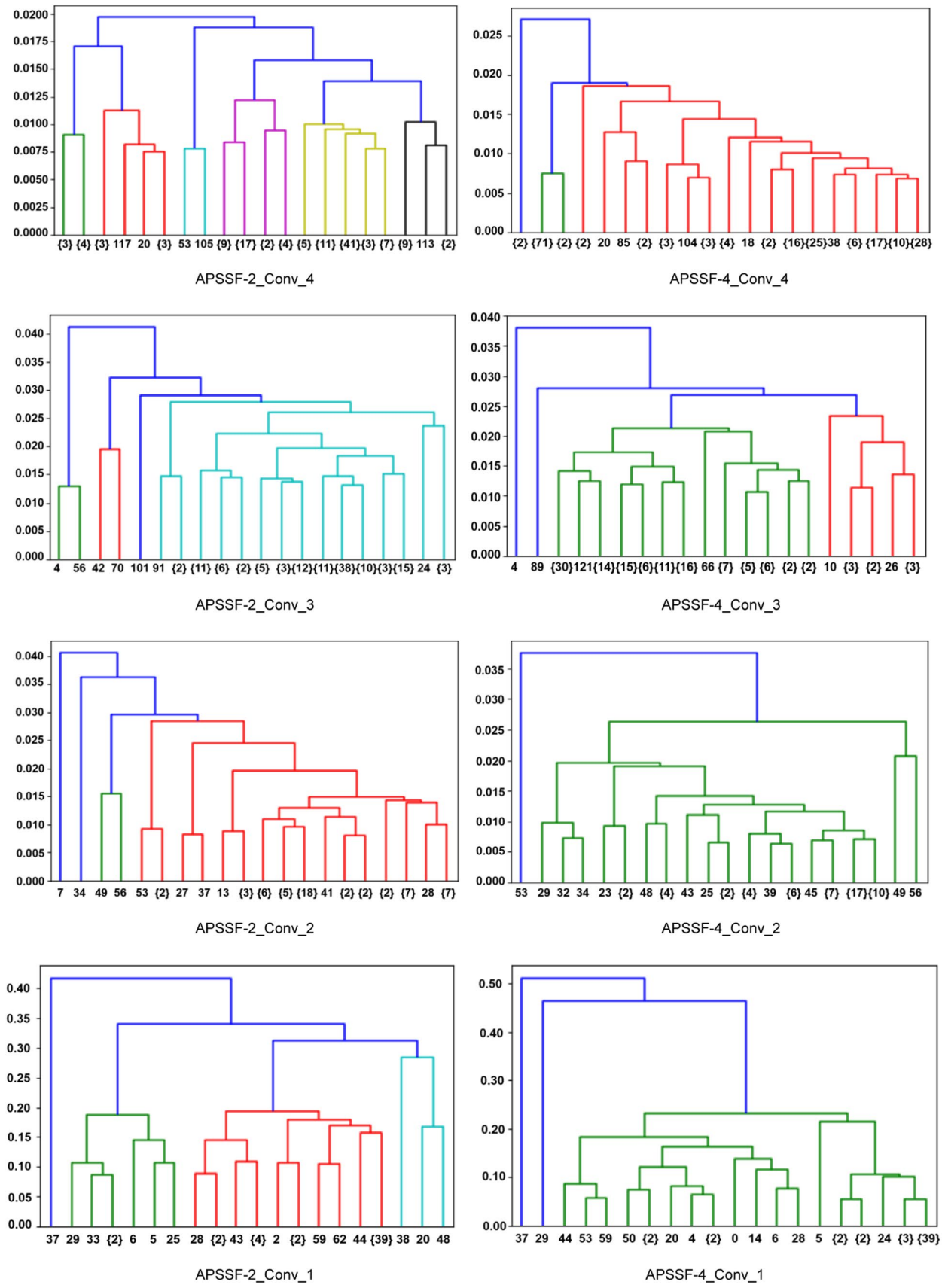

Fig. 7 Pruning cluster tree graph of VGG-16 on the CIFAR10

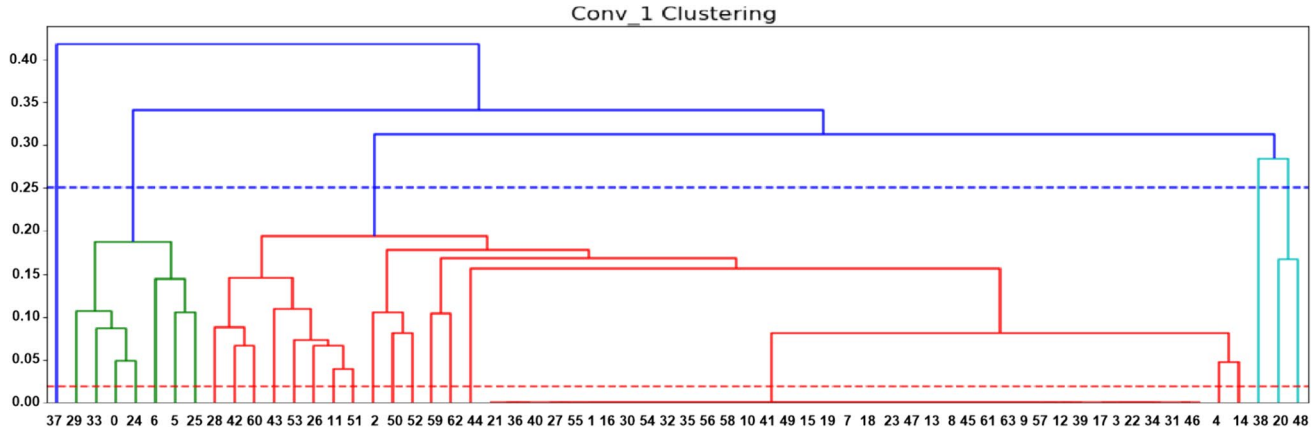

Fig. 8 Cluster tree graph of the first convolution layer in VGG-16. The number of clusters is 28 when pruning the first convolution layer, as shown in the red dashed line. Filters are divided into 28 categories. The first filter in each category is preserved, and the rest are removed. The cluster cutoff is the red dashed line

particular layers to preserve higher accuracy while still achieving notable model compression.

This analysis emphasizes the importance of considering the specific characteristics and redundancy levels within different layers when performing model pruning. By carefully evaluating the trade-off between accuracy and model size reduction, we can optimize the pruning process to achieve the desired balance and enhance the overall efficiency of the model. Fig. 5 and Fig. 6 visually show the number of filters in APSSF-2 and APSSF-4 in the form of bar charts respectively. Through comparative analysis, we can find that the number of filters in both models decreased after filter pruning, especially the APSSF-4, and the number of filters is more obvious. This change means that the model effectively reduces the computational complexity and the number of model parameters while maintaining the high performance.

"Conv_X" represents the X-th convolutional layer in VGG-16. Acc ± is the percentage point of the change in network accuracy after pruning. "Parameter ↓" refers to the percent of parameter reduction. "Flops ↓" refers to the percent of FLOPs reduction.

Figure 7 presents the cluster tree diagram of the first to fifth convolutional layers (Conv_1, Conv_2, Conv_3, Conv_4) for both APSSF-2 and APSSF-4. In this diagram, the X-axis represents the index of filters, while the Y-axis represents the distance between the categories. The vertical lines in the tree graph represent the distance between different categories. A greater distance between these vertical lines indicates a larger dissimilarity between the corresponding categories. This visual representation allows for a clear observation of the clustering process throughout the different layers. By analyzing the cluster tree graph, the clustering process can be discerned,

Table 3 Performance of VGG-16 with $\delta = 2$ & $\delta = 4$ on CIFAR100

Conv_X	APSSF-2				APSSF-4			
	Acc ±	Parameter (M)	Parameter ↓ (%)	Flops ↓ (%)	Acc ±	Parameter (M)	Parameter ↓ (%)	Flops ↓ (%)
Conv_13	−0.02	13.43	12.30	74.22	0.01	13.33	12.95	74.41
Conv_12	−0.01	11.58	24.38	77.78	0.00	11.11	27.45	78.67
Conv_11	0.00	9.60	37.31	81.58	−0.02	8.89	41.95	82.95
Conv_10	0.00	7.91	48.34	92.54	0.01	6.68	56.38	87.22
Conv_9	+0.02	6.13	59.97	88.27	0.03	4.46	70.87	91.48
Conv_8	+0.03	4.94	67.74	90.55	0.01	3.13	79.56	94.04
Conv_7	+0.01	4.35	71.59	91.70	0.00	2.46	83.94	95.33
Conv_6	+0.01	3.89	74.60	92.57	0.00	1.91	87.53	96.39
Conv_5	0.00	3.59	76.56	93.15	−0.03	1.57	89.75	97.03
Conv_4	+0.02	3.45	77.47	93.42	−0.05	1.41	90.79	97.35
Conv_3	−0.01	3.38	77.93	93.57	−0.07	1.32	91.38	97.51
Conv_2	−0.02	3.34	78.19	93.64	−0.04	1.28	91.64	97.60
Conv_1	+0.01	3.32	78.32	93.67	−0.08	1.27	91.71	97.61

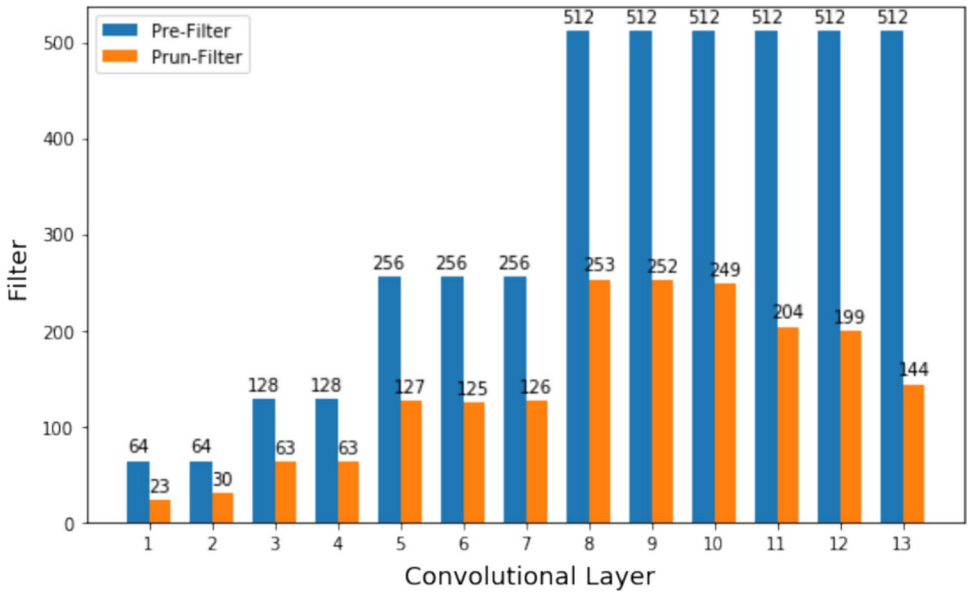

Fig. 9 The number of filters for APSSF-2 on the CIFAR100 in VGG-16

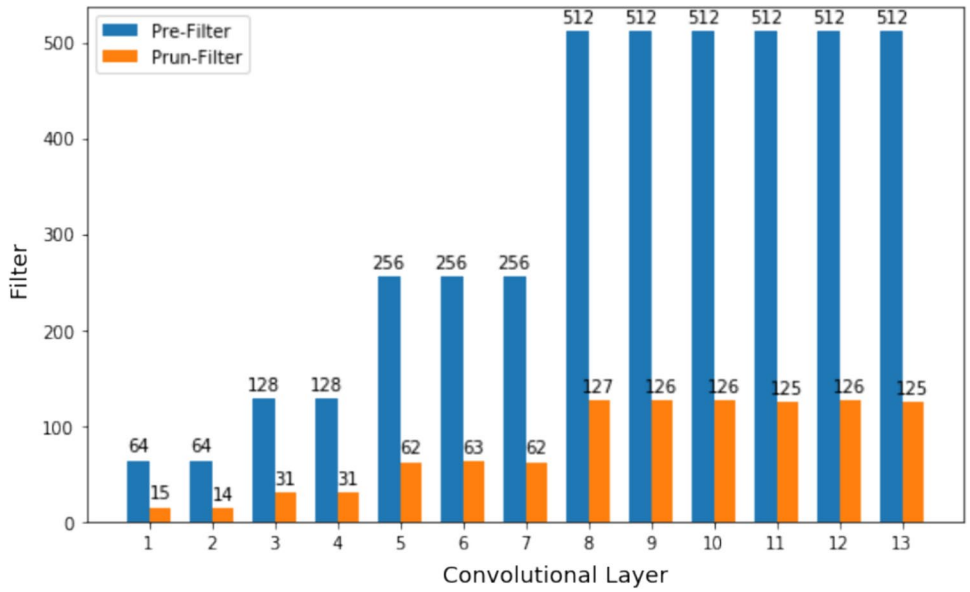

Fig. 10 The number of filters for APSSF-4 on the CIFAR100 in VGG-16

showcasing how filters are grouped based on their similarity. The branches of different colors in the graph represent different classes. Due to the large number of filters, only the 20 nodes of filters are shown in Fig. 7. For example, in the APSSF-2_Conv_1 subgraph, the X-axis represents the index of filters, and the values in parentheses indicate the number of filter nodes included. There are a total of 28 clusters, Fig. 8 is a detailed expansion of the APSSF-2_Conv_1 subgraphs. Cluster 1 includes the 37th filter, cluster 2 includes the 29th filter, cluster 3 includes the 33rd filter, and so on. In addition, cluster 21 comprises the 1st, 3rd, 7th, 8th, 9th, 10th, 12th, 13th, 15th, 16th, 17th, 18th, 19th, 21st, 22nd, 23rd, 27th, 30th, 31st, 32nd, 34th, 35th, 36th, 39th, 40th, 41st, 45th, 46th, 47th, 49th, 54th, 55th, 56th, 57th, 58th, 61st, 63rd filters, and so forth, with cluster 28 encompassing the 48th filter. Based on the clustering results, the 64 filters are divided into 28 clusters, which represent relatively independent subsets. This structure helps us better understand the features and distribution of the data. By observing the filters contained in each cluster, we can infer which filters have similar impact patterns in the feature space. For example, cluster 21 contains 37 filters, indicating that these filters have similar features in a certain feature space and can be classified into the same category or label. This helps us discover the correlations and interactions between data features. Based on the clustering results, we choose to retain the first filter in each cluster and prune the rest. The number of filters is reduced from 64 to 28 in 1st convolutional layer.

Table 4 Performance of ResNet-34 with $\delta = 2$ on CIFAR10

Conv_X	APSSF-2				APSSF-4			
	Acc ±	Parameter (M)	Parameter ↓ (%)	Flops ↓ (%)	Acc ±	Parameter (M)	Parameter ↓ (%)	Flops ↓ (%)
Conv_35	0.00	18.90	11.32	72.12	0.00	17.75	16.71	75.51
Conv_33	0.00	16.49	22.63	75.74	0.00	14.20	33.37	79.64
Conv_30	0.00	14.71	30.98	80.45	0.00	11.53	45.90	82.85
Conv_28	0.00	14.11	33.79	85.54	0.00	10.64	50.08	87.43
Conv_26	0.00	13.52	36.56	88.36	0.00	9.75	54.25	92.46
Conv_24	0.00	12.91	39.42	91.53	0.00	8.85	58.47	95.04
Conv_22	0.00	12.31	42.24	92.71	0.00	7.95	62.70	96.43
Conv_20	+0.01	11.71	45.05	93.87	+0.01	7.05	66.92	97.59
Conv_17	+0.01	11.26	47.17	94.25	+0.01	6.38	70.06	98.23
Conv_15	+0.02	11.11	47.87	95.32	+0.01	6.16	71.10	98.35
Conv_13	+0.02	10.95	48.62	95.57	+0.02	5.93	72.18	98.58
Conv_11	+0.02	10.80	49.32	95.66	+0.02	5.71	73.21	98.61
Conv_8	+0.02	10.69	49.84	95.71	+0.02	5.54	74.01	98.70
Conv_6	+0.03	10.65	50.03	95.75	+0.03	5.48	74.29	98.75
Conv_4	+0.04	10.61	50.22	95.80	+0.04	5.43	74.52	98.80
Conv_2	+0.04	10.58	50.36	95.84	+0.04	5.37	74.80	98.88

Table 5 Comparison of ResNet-34 on CIFAR10 pruning methods

Method	Acc ±	Parameter (M)	Parameter ↓ (%)	Flops ↓ (%)
GAL ($\lambda = 0.5$)	+0.05	4.60	78.46	65.40
GAL ($\lambda = 0.8$)	+0.08	5.85	72.60	45.20
GA	−0.04	4.64	75.54	51.46
LWM	−0.02	9.74	54.36	32.36
APSSF-2	+0.04	10.58	50.36	95.84
APSSF-4	+0.04	5.37	74.80	98.88

In Fig. 8, the cluster tree graph of the first convolutional layer (Conv_1) in VGG-16 is shown. The graph reveals that when the clustering distance is set at 0.25 (indicated by the blue dashed line), the number of clusters is 5. This information can serve as a manual reference for setting the fixed distance threshold and the number of clusters. However, in the process of filter pruning, it can be challenging to manually determine the appropriate distance threshold and the number of clusters. If there are too many cluster categories, the pruning strength may be weakened. On the other hand, having too few cluster categories may result in excessive pruning and loss of accuracy. To address this challenge, finding a suitable number of clusters dynamically based on the model's parameters becomes essential. An adaptive approach can alleviate the need for manually setting thresholds and conducting multiple tests, as it allows for automatic adjustment based on the desired compression rate.

Figure 8 also illustrates the pruning process of the first convolutional layer. The graph depicts that when pruning this layer, a total of 28 clusters are formed (as indicated by the red dashed line). Each category corresponds to a particular group of filters. Among these clusters, one filter from each category is preserved, while the remaining filters are removed. By dynamically determining the threshold for cluster formation, the adaptive approach helps to optimize the pruning process and achieve a better balance between model size reduction and preserving accuracy.

Adaptive methods improve overall efficiency in model pruning in several aspects:

Reduction of computational complexity: Adaptive methods effectively reduce computational complexity by introducing techniques such as dimension reduction and dynamically adjusting distance thresholds. This makes computing similarity and clustering more efficient, thereby accelerating the model pruning process.

Flexibility and control: Adaptive methods allow parameters such as compression rate and distance threshold to be dynamically adjusted according to specific needs and performance requirements. This flexibility and controllability make the pruning process more intelligent, enabling adjustments based on specific situations and ultimately improving overall efficiency.

Preservation of relative similarity relationships: While reducing computational complexity, adaptive methods are still able to preserve important structural information and relative similarity relationships. This means that during the pruning process, although some absolute similarity values may be lost, the relative similarity relationships are

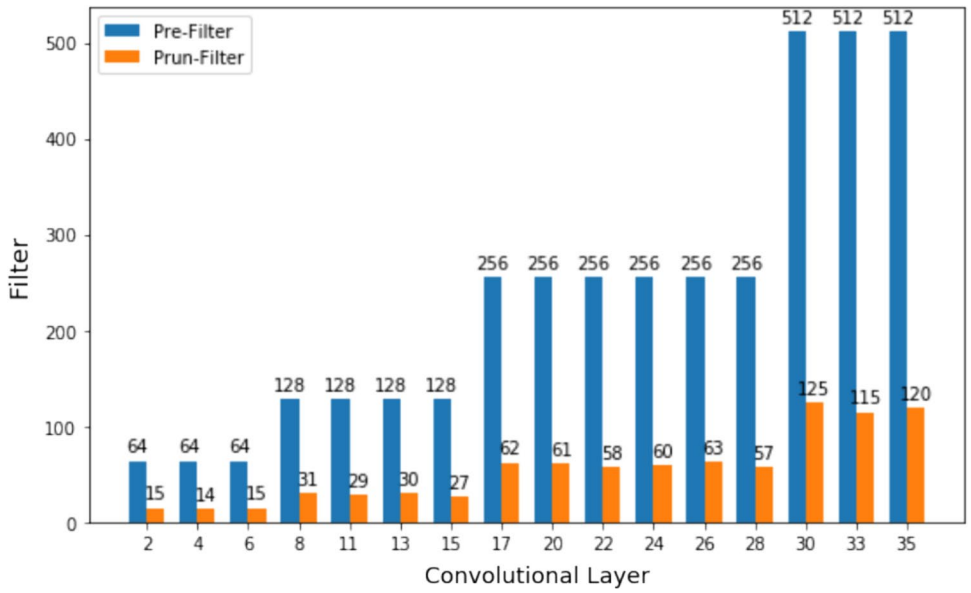

Fig. 11 The number of filters for APSSF-4 on the CIFAR10 in RESNET-34

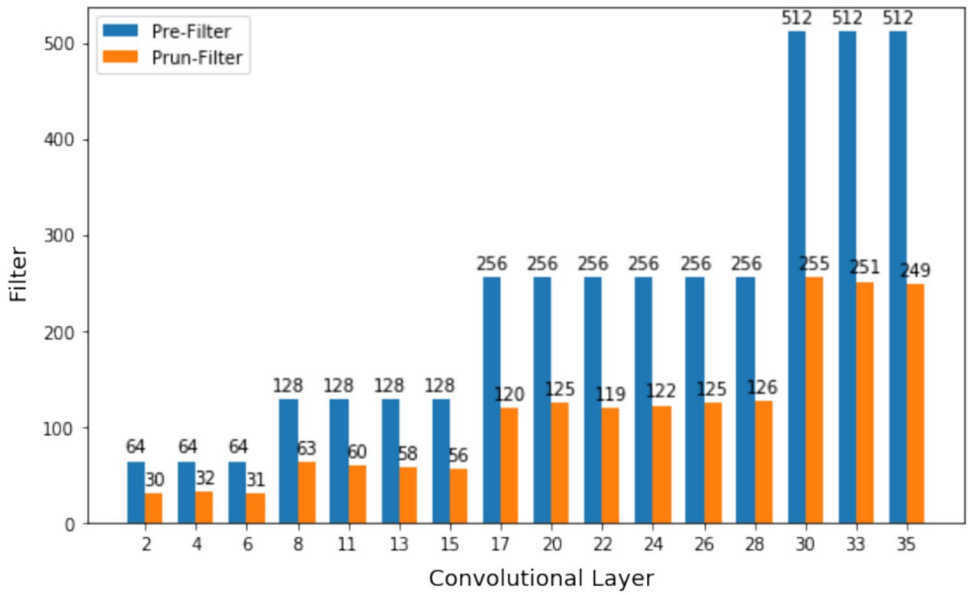

Fig. 12 The number of filters for APSSF-2 on the CIFAR10 in RESNET-34

maintained, effectively reducing the computational burden and improving overall efficiency.

In conclusion, adaptive methods effectively improve the overall efficiency of the model pruning process by reducing computational complexity, providing flexibility and control, preserving relative similarity relationships, and dynamically adjusting the clustering process, making the pruning process more intelligent, efficient, and effective.

4.2.2 Pruning VGG-16 on CIFAR100

A fixed compress rate does not apply for all convolutional layers. Because most architectures of CNN are designed for specific datasets, and the ability of extract the features of the convolution layers is not necessarily suitable for CIFAR10, CIFAR100, and other datasets. The static pruning strategy is suboptimal because each category requires only a few channels. A good pruning strategy should produce different compression rates for each layer. APSSF sets a range of the compression rate within which the number of clusters is adaptively determined. The compression rate varies for each convolution layer after pruning. Tables 3 are the pruning of VGG-16 on CIFAR100, respectively. Table 3 are the experimental results on CIFAR100. The results show that if the value of δ is large, pruning those shallow convolutional layers has a more significant impact on the accuracy.

Table 6 Performance of ResNet-34 with $\delta = 2$ & $\delta = 4$ on CIFAR100

Conv_X	APSSF-2				APSSF-4			
	Acc ±	Parameter (M)	Parameter ↓ (%)	Flops ↓ (%)	Acc ±	Parameter (M)	Parameter ↓ (%)	Flops ↓ (%)
Conv_35	0.00	18.99	11.09	74.64	−0.01	17.80	16.66	75.61
Conv_33	−0.01	16.62	22.18	78.68	−0.01	14.24	33.33	78.77
Conv_30	−0.01	14.84	30.52	82.58	−0.01	11.58	45.78	81.93
Conv_28	−0.01	14.24	33.33	92.76	−0.02	10.69	49.95	85.32
Conv_26	−0.01	13.63	36.18	93.27	−0.02	9.80	54.12	92.58
Conv_24	−0.02	13.02	39.04	93.55	−0.03	8.90	58.33	95.04
Conv_22	−0.03	12.42	41.85	93.70	−0.04	8.01	62.50	96.53
Conv_20	−0.02	11.82	44.66	93.57	−0.03	7.12	66.66	96.76
Conv_17	−0.01	11.37	46.76	93.67	−0.03	6.46	69.75	97.43
Conv_15	−0.01	11.22	47.47	93.89	−0.02	6.23	70.83	97.67
Conv_13	−0.01	11.07	48.17	94.27	−0.02	6.01	71.86	97.87
Conv_11	0.00	10.92	48.87	94.64	−0.01	5.78	72.94	98.21
Conv_8	0.00	10.85	49.20	94.77	0.00	5.61	73.73	98.52
Conv_6	+0.01	10.81	49.39	94.83	0.01	5.56	73.97	98.58
Conv_4	+0.02	10.77	49.57	94.95	0.01	5.50	74.25	98.62
Conv_2	+0.03	10.73	51.45	94.97	0.02	5.44	74.53	98.68

Fig. 9 and Fig. 10 respectively compare the number of filters in each layer before and after VGG-16 pruning on the cifar100 dataset.

4.3 Pruning ResNet

The VGG-16 is standard convolutional structures. The ResNet-34 increased the residual block compared to VGG-16, and it has more parameters and higher accuracy. In the research, only the standard convolutional layers in ResNet-34 were pruned, while the structure of the residual blocks remained unchanged. For the ResNet-34 on the CIFAR dataset, preprocessing includes the following steps:

Data type conversion: The loaded training and testing data is converted to the float32 type. This is because in deep learning models, 32-bit floating-point numbers are commonly used to represent data.

Label processing: One-hot encoding is applied to the labels of the training and testing sets. We convert category labels into the one-hot encoding format.

These preprocessing steps ensure the consistency of data format and compliance with the requirements of deep learning models, enabling subsequent model training and evaluation.

The learning rate (lr) is set to 0.0001, which is an essential hyperparameter controlling the step size of model parameter updates. A smaller lr typically means slower convergence speed, but it may lead to better results. The echop was set to 500. The lr decay is set to 1e-6, indicating that the lr decays exponentially at each update step. Momentum is set to 0.9, serving as a method to accelerate SGD and aiding in finding the optimal solution more quickly in the parameter space. Nesterov momentum is set to True, indicating the use of Nesterov momentum, which is an improved momentum method that converges to the optimal solution more quickly. The choice of these parameters is usually based on empirical observations and experimental results, and can be adjusted based on the specific problem and dataset.

4.3.1 Pruning ResNet-34 on CIFAR10

As demonstrated in Table 4, we conducted experiments by setting δ to 2 (APSSF-2) and 4 (APSSF-4). The accuracy of the pruned network (APSSF-2) is increased by 0.04 on CIFAR10. The number of parameters is reduced from 17.75M to 5.37 M, reduced by 50.36%, and FLOPs is also reduced by 95.84%. Moreover, in our experiments, the accuracy of the APSSF-4 also increased by 0.04 on the CIFAR10. The number of parameters was reduced to 5.37M, indicating a significant reduction of 74.80%. In addition, the FLOPs decreased by an impressive 98.88%

Comparing our method with others, such as GAL, GA [16], and LWM [10], as shown in Table 5, it is worth noting that the FLOPs achieved by APSSF outperformed the other algorithms. Specifically, the FLOPs of APSSF-4 showed a remarkable decrease of 98.88%, exhibiting a superiority of 33.48% over GAL ($\lambda = 0.5$) and a significant improvement of 47.42% over LWM. However, it is important to consider that although the APSSF method showed slightly lower accuracy compared to GAL, its outstanding reduction in FLOPs highlights its competitiveness.

Figures 11 and 12 show the number of filters before and after pruning for each layer when the δ is set to 4 (APSSF-4) and 2 (APSSF-2). The horizontal axis represents the convolutional layers, and the vertical axis represents the number of filters. The blue bars in the graph represent the number of filters before pruning, while the orange bars represent the number of filters after pruning. It is clear to see the changes in the number of filters for each layer before and after pruning.

In this case, the APSSF method adopts an adaptive strategy to determine the number of clusters, and the pruning quantity of each layer automatically seeks a suitable value within the compression rate range. This is different from setting a fixed pruning ratio threshold because it allows each layer to determine the pruning quantity based on its own characteristics and data distribution, rather than simply applying a fixed compression rate. This adaptive pruning method can better adapt to the characteristics of different layers, thereby maintaining the performance and effectiveness of the model after pruning. By automatically finding the appropriate pruning quantity, the APSSF method can achieve more efficient model pruning while maintaining model performance. This adaptability helps to improve the efficiency and accuracy of model pruning, making the pruned model more compact and efficient.

Table 7 Comparison of ResNet-50 on ImageNet pruning methods

Method	Acc ±	Parameter (M)	Parameter ↓ (%)	Flops ↓ (%)
GAL ($\lambda=0.5$)	+0.04	21.20	78.46	43.03
GAL ($\lambda=1$)	+0.06	14.67	72.60	61.37
FPGM	−0.01	–	–	53.50
SSR-GR	−0.01	–	–	55.10
HRel	−0.03	9.10	64.40	66.42
LFPC	−0.02	–	–	60.80
ThinNet	−0.06	8.66	66.07	71.27
CSHE	−0.05	13.08	48.70	65.10
ASFRP	−0.15	–	–	41.80
APSSF-2	+0.36	11.69	54.26	84.64
APSSF-4	+0.45	5.16	79.80	86.58

4.3.2 Pruning ResNet-34 on CIFAR100

In addition to the experiments conducted on the CIFAR10, we also evaluated our proposed method by setting δ to 2 (APSSF-2) and 4 (APSSF-4) on the CIFAR100. The results revealed that the accuracy of the APSSF-4 increased by 0.02. Concurrently, the number of parameters decreased from 17.80 million to 5.44 million, representing a reduction of 74.53%. Furthermore, the FLOPs also underwent a significant decrease of 98.68%.

To provide a comprehensive comparison, the results of the experiments on the CIFAR100 are displayed in Table 6. These additional findings further validate the effectiveness of our proposed method in achieving higher accuracy and substantial reductions in the number of parameters and FLOPs.

In ResNet-34, the selection of pruning standard convolutional layers can have a significant impact on the overall model architecture and performance. Below, we will detail how this selection affects the model architecture and performance, and discuss how dimensionality reduction techniques can enhance overall efficiency.

Architecture impact: Pruning standard convolutional layers leads to changes in the modelk architecture. Through pruning, some of the convolutional layer filters are pruned, thereby reducing the model's parameter and computational load. The pruned model architecture may become sparser, meaning that many positions in the output feature maps of certain convolutional layers are zero. This sparsity can offer computational and storage advantages as calculations for zero-value positions can be skipped.

Performance impact: The selection of pruning standard convolutional layers can impact model performance. Pruning may result in a decrease in model accuracy as some important filters are pruned. Therefore, careful selection of the convolutional layers to prune is necessary during the pruning process to maintain model performance. To mitigate the impact of pruning on performance, fine-tuning techniques can be employed to retrain the pruned model. Fine-tuning can aid in restoring or improving performance by adjusting filters through further training on the pruned model.

Table 8 Performance of indicator network

Model	$\delta=2$			$\delta=4$		
	Acc (%)	Parameter (M)	FLOPs	Acc (%)	Parameter (M)	FLOPs
Pruned VGG-16	84.26	2.39	1.19×10^7	83.23	1.19	0.59×10^7
Same_StructureNet	82.75	2.39	4.41×10^7	79.68	1.19	2.01×10^7
Same_LayerNet	83.43	2.68	4.55×10^7	76.86	1.18	2.00×10^7
Same_ParameterNet	10.00	2.39	4.43×10^7	10.00	1.18	2.00×10^7
Same_FilterNet	86.62	7.43	12.61×10^7	82.52	9.96	16.92×10^7

Computational complexity: The computational complexity of pruning standard convolutional layers depends on the extent of pruning and the resulting model architecture. Pruning can reduce computational load as the pruned filters no longer participate in calculations. However, pruning also introduces sparsity, which may require additional computations to handle sparse matrix multiplication. Dimensionality reduction techniques can enhance overall efficiency. For instance, dimensionality reduction techniques for convolutional layers (e.g., 1×1 convolutions) can reduce the number of channels in feature maps, thereby decreasing computational load and storage requirements. Dimensionality reduction techniques can be applied before or after pruning to further enhance overall efficiency.

In conclusion, the selection of pruning standard convolutional layers can impact the architecture and performance of ResNet-34. Pruning can reduce computational and parameter load but may also lead to performance degradation. By carefully selecting the convolutional layers to prune and employing fine-tuning techniques, pruning can be achieved while maintaining performance. Dimensionality reduction techniques can further enhance overall efficiency by reducing computational complexity and storage requirements.

4.3.3 Pruning ResNet-50 on ImageNet

The evaluation of APSSF was conducted on the ImageNet dataset using the ResNet-50 architecture. During the training process, a batch size of 32 was employed, and the network was trained for 300 epochs with a lr of 0.001. The cross-entropy loss function was adopted to calculate the loss. In the pruning process, we specifically selected two standard convolutional layers in each residual block for pruning.

When training the ResNet-50 model on the ImageNet dataset, data preprocessing and initialization steps are performed.

Preprocessing:

Image resizing: Images in the ImageNet dataset come in various sizes and need to be resized to a uniform size of 224×224 for input into the ResNet-50 model.

Mean rormalization: Mean normalization is applied to each channel of the images, which involves subtracting the mean of each channel to bring the data mean closer to 0. This helps accelerate the model's convergence process.

Standardization: Each channel of the images undergoes standardization, where the value of each channel is divided by its standard deviation, aiming to bring the data's standard deviation close to 1.

Initialization:

When training the ResNet-50 model, pre-trained weights are used as initialization parameters. These weights are obtained from training on the ImageNet dataset and can aid the model in converging faster and achieving better performance.

The results demonstrated that the parameters of the APSSF-4 achieved a reduction of 79.80%, while the FLOPs experienced a significant drop of 86.58%. These outcomes indicate that our pruning method outperforms other techniques, such as GAL, FPGM [11], SSR-GR [26], ThiNet [13], HRel [27], LFPC [28], CSHE, and ASFRP[29], as displayed in Table 7. As can be seen from the Table 7, the APSSF method has great advantages in improving the accuracy and speed (FLOPs).

The APSSF method is compared with other methods on the ResNet-50 image dataset in the following aspects:

Accuracy improvement: According to the provided data, the accuracy improvement of the APSSF-2 method on the ResNet-50 image dataset is $+0.36$, while the accuracy improvement of the APSSF-4 method is $+0.45$. These two values are significantly higher than those of other methods, indicating that the APSSF method can maintain a high level of accuracy after pruning, and even achieve significant improvement.

Reduction in the number of parameters: The reduction in the number of parameters for the APSSF-2 method is 54.26%, while for the APSSF-4 method, it is 79.80%. These two values are also significantly higher than those of other methods, indicating that the APSSF method can achieve a substantial reduction in the number of parameters after pruning.

Reduction in FLOPs: The reduction in floating-point operations for the APSSF-2 method is 84.64%, while for the APSSF-4 method, it is 86.58%. Similarly, these two values are significantly higher than those of other methods, indicating that the APSSF method can achieve a substantial reduction in floating-point operations after pruning.

Based on these numerical comparisons, we can conclude that the APSSF method outperforms other methods on the ResNet-50 image dataset, primarily in terms of accuracy improvement, reduction in the number of parameters, and reduction in floating-point operations. These numerical comparisons clearly demonstrate that the APSSF method can comprehensively consider multiple performance indicators during the pruning process and achieve a substantial reduction in parameters and computational workload while maintaining efficient performance.

4.4 The Performance of Indicator Network

APSSF has the ability to generate subnets of varying sizes and accuracies. To evaluate whether these subnets possess a performance advantage over directly building models with the same structure, we conducted experiments to compare the performance of indicator networks with that of models having similar structures. In these experiments, we

constructed four CNNs from scratch: Same_StructureNet, Same_LayerNet, Same_ParameterNet, and Same_FilterNet. The performance of these models is presented in Table 8. We build the model based on the structure of the pruned VGG-16, using the CIFAR10 dataset with a training epoch set to 300, a batch size of 128, and a lr of 0.01.

- Same_StructureNet

This model is built with the same structure as the pruned VGG-16. It is trained from scratch and the same parameter settings employed during the pruning process.

- Same_LayerNet

The number of layers in this model is the same as the pruned VGG-16, but the filters are equally distributed across each layer.

- Same_ParameterNet

This model has the same total number of parameters as the pruned VGG-16. It consists of four convolutional layers, with the first and second layers having 64 filters each, the third layer having 128 filters, and the fourth layer having 142 filters.

- Same_FilterNet

A network with the same total number of filters as the pruned VGG-16. Same_FilterNet has nine convolutional layers. Each of the first three layers has 64 filters, layers 4 and 5 have 128 filters each, layers 6, 7 and 8 have 256 filters each, and layer 9 has 512 filters.

The results presented in Table 8 indicate that the accuracy and FLOPs of Same_StructureNet, Same_LayerNet, and Same_ParameterNet are lower than those of the pruned VGG-16. Although the accuracy of Same_FilterNet reaches 82.52%, which is only 0.68% lower compared to the pruned VGG-16, both the number of parameters and FLOPs are considerably higher than those of the pruned VGG-16. This observation suggests that redundancy in parameters is necessary in the initial stages of model training. Training a small-scale model from scratch leads to significantly lower overall performance compared to pruning a larger model that already exhibits high accuracy.

Large-scale models typically achieve high accuracy and satisfactory performance. However, using APSSF, it is feasible to obtain small-scale models with similar levels of accuracy. This illustrates the effectiveness of the APSSF method in producing small-scale models that possess the same level of performance as large-scale models.

5 Conclusions

Pruning is widely acknowledged as an effective technique for compressing CNN. This paper introduces the APSSF pruning method, which offers several notable advantages, including a high compression rate, minimal accuracy loss, efficient computational speed, and straightforward implementation. The experimental results on CIFAR10/100 and ImageNet datasets demonstrate that APSSF achieves state-of-the-art performance. Through systematic experimentation and analysis on these three benchmark datasets, we observed that deep convolutional layers in CNN often contain a significant amount of redundant parameters. By selectively removing these redundant parameters, the pruned model can even outperform the original model. Moving forward, our future research will focus on exploring more efficient pruning methods to further optimize the performance of compressed models.

Author Contributions The study is written by all authors. All authors reviewed the results and approved the final version of the manuscript.

Funding This work is supported by the Key Research and Development Plan of Shanxi Provence No. 201903D421007 and 202302010101004.

Data Availability The datasets used in the experiments are publicly available.

Declarations

Conflict of Interest The authors declare no conflicts of interest relevant to this article.

Ethical Approval and Consent to Participate All authors have read and agree to participate in this paper. In addition, the authors confirm that this manuscript has not been submitted to any other journal for simultaneous consideration.

Consent for Publication All authors appeared in this paper agreed to publication in this journal.

References

1. Simonyan, K, Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (ICLR), pp. 1–14 (2015)
2. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, pp. 770–778 (2016). https://doi.org/10.1109/CVPR.2016.90.
3. Chollet, F.: Xception: Deep learning with depthwise separable convolutions. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, pp. 1800–1807 (2017). https://doi.org/10.1109/CVPR.2017.195.
4. Xie, S., Girshick, R., Dollár, P., Tu, Z., He, K.: Aggregated residual transformations for deep neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), vol. 1, no. 2 pp. 1492–1500 (2017)
5. Zagoruyko, S., Komodakis, N.: Wide residual networks. arXiv preprint arXiv:1605.07146 (2016)
6. Zhang, Q., Shi, Y., Zhang, L., Wang, Y., Tian, Y.: Learning compact networks via similarity-aware channel pruning. In: 2020 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR), Shenzhen, China, pp. 145–148. https://doi.org/10.1109/MIPR49039.2020.00037 (2020)
7. Han, S., Pool, J., Tran, J., Dally, W. L.: Learning both weights and connections for efficient neural networks. In: Proceedings of the 28th International Conference on Neural Information Processing Systems, vol. 1, December 2015, pp.1135–1143 (2015)
8. Carreira-Perpinan, M. A., Idelbayev, Y.: "Learning-Compression" algorithms for neural net pruning. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, pp. 8532–8541. https://doi.org/10.1109/CVPR.2018.00890 (2018)
9. Ghimire, D., Kim, S.H.: Magnitude and similarity based variable rate filter pruning for efficient convolution neural networks. Appl. Sci. **13**(1), 316 (2023). https://doi.org/10.3390/app13010316
10. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H. P.: Pruning filters for efficient convnets. In: Proceedings of International Conference on Learning Representations, pp. 1–13 (2017)
11. He, Y., Liu, P., Wang, Z., Hu, Z., Yang, Y.: Filter pruning via geometric median for deep convolutional neural networks acceleration. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (Long Beach, CA), pp. 4340–4349. https://doi.org/10.1109/CVPR.2019.00447 (2019)
12. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: Proceedings of the IEEE International Conference on Computer Vision (Venice), pp. 2736–2744. https://doi.org/10.1109/ICCV.2017.298 (2017)
13. Luo, J. H., Wu, J., Lin, W.: ThiNet: a filter level pruning method for deep neural network compression. In: 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, pp. 5068–5076. https://doi.org/10.1109/ICCV.2017.541 (2017)
14. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient inference. In: International Conference on Learning Representations (ICLR), pp. 1–17 (2017)
15. Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J.: Importance estimation for neural network pruning. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, pp. 11256–11264. https://doi.org/10.1109/CVPR.2019.01152 (2019)
16. Yang, T.-J., Chen, Y.-H., Sze, V.: Designing energy-efficient convolutional neural networks using energy-aware pruning. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, pp. 6071–6079. https://doi.org/10.1109/CVPR.2017.643 (2017)
17. Chu, C., Chen, L., Gao, Z.: Similarity based filter pruning for efficient super-resolution models. In: 2020 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB), Paris, France, pp. 1–7. https://doi.org/10.1109/BMSB49480.2020.9379712 (2020)
18. Shao, M., Dai, J., Wang, R., Kuang, J.D., Zuo, W.M.: CSHE: network pruning by using cluster similarity and matrix eigenvalues. Int. J. Mach. Learn. & Cyber. **13**, 371–382 (2022). https://doi.org/10.1007/s13042-021-01411-8
19. Li, L.Q., Xu, Y.H., Zhu, J.: Filter level pruning based on similar feature extraction for convolutional neural networks. IEICE Trans. Inf. Syst. **E101D**(4), 1203–1206 (2018)
20. Chang, J.F., Lu, Y., Xue, P., Xu, Y.Q., Wei, Z.: Iterative clustering pruning for convolutional neural networks. Knowl.-Based Syst. **265**, 8 (2023). https://doi.org/10.1016/j.knosys.110386
21. Ian, G., Yoshua, B., Aaron, C.: Deep Learning, pp. 201–202. The People's Posts and Telecommunications Press, Beijing (2024)
22. Krizhevsky, A.: Learning multiple layers of features from tiny images. Technical Report, Computer Science Department, University of Toronto, http://www.cs.toronto.edu/~kriz/cifar-10-binary.tar.gz (2009)
23. Krizhevsky, A.: Learning multiple layers of features from tiny images. Technical Report, Computer Science Department, University of Toronto. http://www.cs.toronto.edu/~kriz/cifar-100-binary.tar.gz. (2009)
24. Jia, D., Wei, D., Socher, R., Li, L. J., Kai, L., Li, F. F.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition, pp. 248–255 (2009)
25. Lin, S. et al.: Towards Optimal Structured CNN Pruning via Generative Adversarial Learning. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, pp. 2785–2794. https://doi.org/10.1109/CVPR.2019.00290. (2019)
26. Wang, Z., Li, C., Wang, X.: Convolutional Neural Network Pruning with Structural Redundancy Reduction. In: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Nashville, TN, USA, pp. 14908–14917, https://doi.org/10.1109/CVPR46437.2021.01467 (2021)
27. Sarvani, C., Ghorai, M., Dubey, S.R., Basha, S.S.: Hrel: filter pruning based on high relevance between activation maps and class labels. Neural Netw. **147**, 186–197 (2022)
28. He, Y., Ding, Y., Liu, P., Zhu, L., Zhang, H., Yang, Y.: Learning Filter Pruning Criteria for Deep Convolutional Neural Networks Acceleration. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, pp. 2006–2015. https://doi.org/10.1109/CVPR42600.2020.00208 (2020)
29. Cai, L., An, Z., Yang, C., Xu, Y.: Softer pruning, incremental regularization. In: 2020 25th International Conference on Pattern Recognition (ICPR), Milan, Italy, pp. 224–230, https://doi.org/10.1109/ICPR48806.2021.9412993 (2021)