Machine Vision and Applications (2021) 32:20
https://doi.org/10.1007/s00138-020-01148-4

ORIGINAL PAPER

A pruning method based on the measurement of feature extraction ability

Honggang Wu[1] · Yi Tang[2] · Xiang Zhang[2]

Received: 28 September 2019 / Revised: 17 October 2020 / Accepted: 26 October 2020 / Published online: 21 November 2020

Abstract
As the network structure of convolutional neural network (CNN) becomes deeper and wider, network optimization, such as pruning, has received ever-increasing research focus. This paper propose a new pruning strategy based on Feature Extraction Ability Measurement (FEAM), which is a novel index of the feature extraction ability from both theoretical analysis and practical operation. Firstly, FEAM is computed as the product of the the kernel sparsity and feature dispersion. Kernel sparsity describes the ability of feature extraction in theory, and feature dispersion represents the feature extraction ability in practical operation. Secondly, FEAMs of all filters in the network are normalized so that the pruning operation can be applied to cross-layer filters. Finally, filters with weak FEAM are pruned to obtain a compact CNN model. In addition, fine-tuning is adopted to restore the generalization ability. Experiments on CAFAR-10 and CUB-200-2011 demonstrate the effectiveness of our method.

Keywords Pruning · Kernel sparsity · Feature dispersion · Feature extraction ability

1 Introduction

It is well known that the convolutional neural network (CNN) has achieved a great success in various computer vision tasks [1], including object detection [2–4], object classification [5,6], semantic segmentation [7,8], and many others. However, with the deepening and widening of CNN convolution layer, higher computational overhead and larger memory are required, so it is difficult to deploy CNN model to resource-limited devices [9]. For instance, AlexNet [10] network contains about 6×10^6 parameters, while some better networks like VGG [11] contain about 1.38×10^8 parameters. For less complex tasks, such as simple image recognition, the VGG network still require more than 500MB memory and 1.56×10^{10} Float Point Operations (FLOPs). The over-parameterization of deep learning is a major obstacle to the application of CNN [12] .

Thus, network compression has drawn a significant amount of interests from both academia and industry. In recent years, numerous efficient compression methods have been proposed, including low-rank approximation [12,13], parameter quantization [14,15] and binarization [16]. Among them, network pruning [17–20] has excellent performance in reducing redundancy of CNNs, and it has better model deployment ability compared with other methods. Network pruning resorts to removing unimportant connections from a well-trained network with negligible impact on network performance.

In this paper, a new pruning strategy based on Feature Extraction Ability Measurement (FEAM) is proposed. Based on this fact that the quality of output features is the criterion to judge the importance of a filter [21], we develop FEAM to measure the feature extraction ability from both theoretical analysis and practical operation, and use FEAM to guide the pruning process. As shown in Fig. 1b, FEAM is computed as the product of kernel sparsity and feature dispersion. Kernel sparsity is based on the given network structure without considering input data, which means it is an index from theoretical analysis. On the contrary, feature dispersion is based

✉ Yi Tang
tangyi1994@uestc.edu.cn

Honggang Wu
wu-honggang@hotmail.com

Xiang Zhang
uestchero@uestc.edu.cn

1 Civil Aviation Administration of China, Chengdu 610041, China

2 University of Electronic Science and Technology of China, Chengdu 611731, China

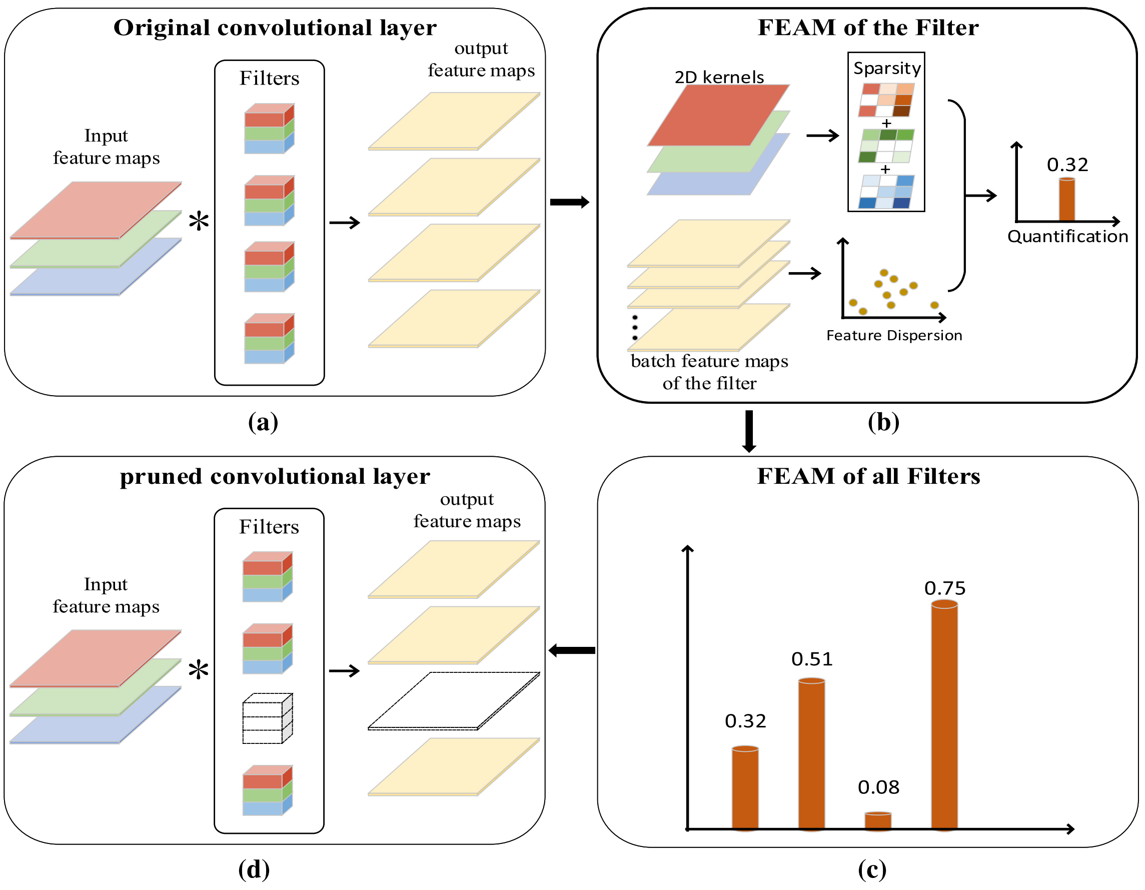

Fig. 1 The framework of the proposed method. Based on the original convolutional layer (**a**), the sparsity of filters is computed with L1-norm, the dispersion of feature maps is computed as standard deviation, and the FEAM is the product of sparsity and dispersion (**b**). FEAMs of all filters are computed and normalized (**c**), and finally filters with weak FEAM are pruned to generate a more compact network (**d**)

on the output of filters. That is to say, it is data-driven and relevant to practical applications. Therefore, as the combination of kernel sparsity and feature dispersion, FEAM provides a comprehensive description of the feature extraction ability. Furthermore, we normalize the FEAMs of all filters, so that the pruning operation can be applied to cross-layer filters, thus avoiding layer-by-layer threshold sensitivity analysis. Finally, filters with weak FEAMs are pruned to generate a compact CNN model with only a slight performance degradation. In addition, fine-tuning is employed to restore the generalization capabilities.

The proposed pruning method is tested within four commonly used CNN structures: VGG-16 [22], Resnet-110 [23], the lightweight models MobileNet_V1 [29] and MobileNet_V2 [28]. Two representative datasets, CIFAR10 and CUB_200_2011, are used as benchmark. For CIFAR10, our method achieves 4.9× compression and reduces FLOPs by 89.4% on VGG-16 with about 0.3% top-1 accuracy drop, and achieves 2.4× compression on Resnet-110 with 0.9% top-1 accuracy drop. For CUB_ 200_2011, our method reduces FLOPs by 78.7% on VGG-16 with roughly 0.6% top-1 accuracy drop, reduces FLOPs by 55.9% on MobileNet_V1 with about 0.4% top-1 accuracy drop, and reduces FLOPs by 17.5% on MobileNet_V2 with about 0.7% top-1 accuracy drop. This is better than most similar pruning methods.

2 Related work

In this section, we briefly review some popular network pruning methods, which can be divided into two classes, unstructured pruning and structured pruning.

Unstructured pruning is to zero the weight values below a certain threshold. Among them, what is impressive is that Han et al. [17] proposed to prune the weights having small magnitudes on AlexNet and VGG, then retrained without affecting the overall accuracy but effectively reduced the number of parameters. However, this pruning operation generates an unstructured sparse model that requires sparse BLAS libraries or even specialized hardware for achieving acceleration.

Structured pruning reduces computational complexity and memory overhead by directly removing structured parts, such as kernels, filters, or layers. It is well supported by a variety of off-the-shelf deep learning platforms. For instance, One pruning criterion is sparsity activated by non-linear ReLu mappings. Hu et al. [18] proposed a data-driven neuron pruning approach to remove unimportant neurons. They argued that if most of these activated feature maps are zero, it is not important for neurons to have a high probability. The criterion measures the importance of neurons by calculating the average percentage of zeros (APoZ) in the activated feature map. However, the APoZ pruning criterion requires the introduction of different threshold parameters for each convolutional layer, which are difficult to accurately determine. Li et al. [24] proposed to remove unimportant filters based on the L1-norm. Molchanov et al. [19] calculated the influence of filters on the loss function based on Taylor expansion. According to the criterion, if the filter has little influence on the loss function, the filter can be safely removed. So they use Taylor expansion to approximate the change in loss. He et al. [25] proposed a channel selection method based on LASSO (Least absolute shrinkage and selection operator) regression, which uses least squares reconstruction to eliminate redundant filters. Similar to our study, Luo et al. [20] proposed a method to calculate entropy of filters to measure the information richness of the convolution kernel. However, the method only consider the information richness of the filter, and the strategy can only compare the entropy value among the same convolutional layers. Most of these methods need to accurately obtain the pruning threshold of each convolutional layer, but this is difficult to achieve. If fixed compression rate is used for pruning, it may lead to irreparable accuracy reduction.

Except for the network pruning methods, some other CNN compression methods are developed, such as designing a more compact architecture. For example, it is known that most parameters of the CNN model exist in the fully connected layers, so the global average pooling is proposed to replace the full connection layer in the Network-In-Network [26]. Son et al. [27] reconstructed the network by unified representation of similar convolutions to achieve effective compression of the network. However, this method has some limitations. It is only effective for 3×3 convolution kernels. Sandler et al. [28] proposed the use of depthwise separable convolution to build a lightweight network, which has also been widely used in mobile devices.

3 Proposed method

Figure 1 illustrates the framework of the proposed method. In original convolution operation (Fig. 1a), output feature maps are obtained after filtering the input feataps with kernels. In our method, a new pruning strategy is embedded into the original convolution operation. First, the kernel sparsity of each filter kernel and the dispersion of output feature maps are computed and combined to get FEAM (Fig. 1b). Then all FEAMs are normalized for cross-layer operation (Fig. 1c). Finally, filters with weak FEAMS are removed from the network to get a compact model (Fig. 1d). In addition, the network is fine-tuned to restore its generalization performance. Next each component are described in detail.

First we give some notations used in this paper. Each convolution layer has a total number of J 3D filters (kernels), and $W_j \epsilon \mathbb{R}^{C_1 \times C_2 \times C_3}$ is the weight matrix of the jth filter, where C_1, C_2 and C_3 represent the dimension, height and width of kernels, respectively.

3.1 Kernel sparsity

Kernel sparsity is based on the kernel weights without considering applications and training data. We compute the sparsity of a filter kernel with the simple L1-norm, which is the sum of the absolute values of all kernel weights:

$$k_j = \sum_{u=1}^{C_1} \sum_{v=1}^{C_2} \sum_{w=1}^{C_3} |C_j(u, v, w)|, \quad j = 1, 2, \ldots, J \tag{1}$$

Theoretically speaking, if most weights in a kernel are close to zero and the absolute values of nonzero weights are small, this kernel should be sparse and does little contribution to the network, and it can be removed from network structure. This case can be reflected by Eq. (1), and hence it is a reasonable definition for kernel sparsity. However, since the practical application varies greatly, there are sometimes exceptions like that an important kernel outputs significant feature maps but has a small kernel sparsity under a specific application scenario. This case can be figured out by the following feature dispersion.

3.2 Feature dispersion

Feature dispersion measures the feature validity of the outputs of a filter, which means it is data-driven and it varies with different training data. We develop feature dispersion to compensate the defect of kernel disparity, the ignorance of the influence of real applications on the feature extraction ability.

Feature dispersion is based on Global Average Pooling (GAP) [26]. As shown in Fig. 2, given a training sample, the output feature of a convolution layer is a $J \times H_{out} \times W_{out}$ tensor, where J is the number of filters in this layer. Then this tensor is converted to a J-dimensional transverse vector

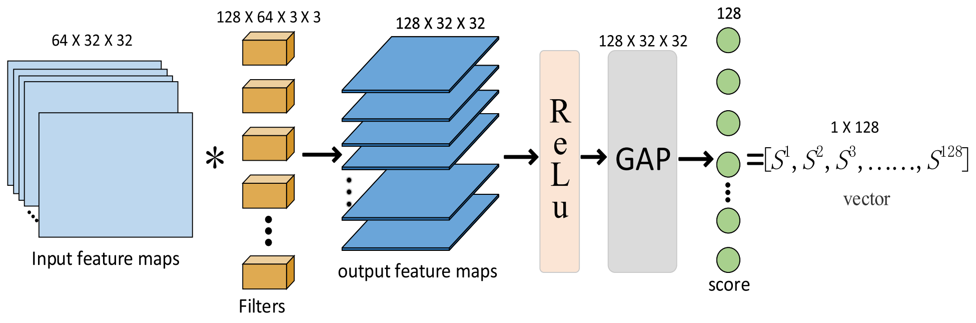

Fig. 2 Illustration of global average pooling

$\mathbf{S} = [s_1, s_2, \ldots, s_J]$ by GAP, where s_j corresponds to the jth filter. It can be seen that this vector $\mathbf{S}$ after GAP will vary with training samples, and hence it is indeed data-driven. Then we repeat this operation shown in Fig. 2 with different training samples, e.g. B number of samples, and we will get a feature matrix $\mathbf{M} = \mathbf{S}^1; \mathbf{S}^2; \ldots; \mathbf{S}^B$, where $\mathbf{S}^b$ corresponds to the GAP output of the bth training sample. Sometimes, we omit the superscript or subscript of $\mathbf{S}$ in case of no confusion.

The feature matrix $\mathbf{M}$ is the basis of feature dispersion. We have known that each transverse vector of $\mathbf{M}$ corresponds to a training sample. On the contrary, each column vector of $\mathbf{M}$ corresponds to a filter, e.g., $\mathbf{M}_{:,j}$ represents the output GAP values of the jth filter. Let μ_j be the mean value of $\mathbf{M}_{:,j}$, the feature dispersion of the jth filter is defined as follows:

$$d_j = \begin{cases} 0, & \mu_j = 0 \\ \sqrt{\frac{\sum_{i=0}^{B}(M_{i,j}-\mu_j)^2}{B}}, & \text{otherwise} \end{cases} \tag{2}$$

The above formula shows that in fact the feature dispersion is the standard deviation, which reflects the fluctuation of samples around the mean value. For a filter, if its feature dispersion is small, which means the output of this filter cannot highlight the difference of input samples, this filter should be judged as a weak filter with poor feature extraction ability. On the contrary, a filter with large feature dispersion means it is able to extract distinguishable features, and hence, it is more likely to be retained.

3.3 FEAM

As shown in above sections, the computation of kernel sparsity is based on kernel weights, and the feature dispersion is based on the output feature maps without considering kernel parameters. That's to say, kernel sparsity is such an index from theoretical analysis, while feature dispersion is from real operation. Whichever indicator is used alone will not give a complete evaluation of the feature extraction ability. As a result, we integrate the two indicators into a new one, FEAM, to comprehensively measure the feature extraction ability of the jth filter:

$$FEAM_j = k_j \times d_j \tag{3}$$

The above equation shows that, the FEAM value is small if both kernel sparsity and feature dispersion are small, and it is large if both kernel sparsity and feature dispersion are large. When one is large and the other is small, the two indicators will competitively measure the feature extraction ability in FEAM.

FEAM is defined as the product of kernel sparsity and feature dispersion in Eq. (3). In this paper, k_j is the quantitative index of kernel sparsity, and d_j is the quantitative index of feature dispersion. We believe that k_j and d_j are relatively robust pruning indexes, and the combination of them can remedy for the defect of single index. Therefore, in the design process of the algorithm, we tried a variety of different combinations, including product and weight sum. According to the experimental results, the product of the two is chosen as the combination method, which is more robust and sensitive. Similarly, we also tried the method of weight sum, and the formula is as follows:

$$FEAM_j = \sqrt{\alpha k_j + d_j} \tag{4}$$

Since the value of k_j is much larger than the value of d_j, we have added a weight α to scale k_j. α can control the proportion of kernel sparsity and feature dispersion in FEAM, and the value of α is 0.25 in the experiment. In Table 2 and Table 4, we added the experimental results (in red font) of FEAM using weight sum method, and we can see that the experimental results of this method are slightly worse than those using product method. The main reason for our analysis is that the value of a affects the final results of FEAM. We also tried a variety of values of α in the experiment and found that it will be a little bit troublesome to get a better weight value. So we think that the product method is relatively robust.

Table 1 Overall performance of the proposed method

Layer	Feature map size	FLOPs			Parameters		
		Original	Pruned	Percentage (%)	Original	Pruned	Percentage (%)
Conv1-1	128 × 128	29.36M	21.56M	73.4	1.79K	1.32K	73.5
Conv1-2	128 × 128	605.02M	305.65M	50.5	36.92K	18.65K	46.7
Conv2-1	64 × 64	302.51M	37.4M	12.4	73.85K	9.13K	12.4
Conv2-2	64 × 64	604.50M	18.74M	3.1	147.58K	4.58K	3.1
Conv3-1	32 × 32	302.25M	5.91M	2.0	295.16K	5.77K	1.9
Conv3-2	32 × 32	604.24M	9.39M	1.5	590.08K	9.17K	1.6
Conv3-3	32 × 32	604.24M	16.83M	2.8	590.08K	16.43K	2.8
Conv4-1	16 × 16	302.12M	10.69M	3.5	1.18M	41.74K	3.5
Conv4-2	16 × 16	604.11M	24.84M	4.1	2.36M	97.04K	4.1
Conv4-3	16 × 16	604.11M	32.37M	5.3	2.36M	126.44K	5.4
Conv5-1	8 × 8	151.02M	9.63M	6.4	2.36M	150.48K	6.4
Conv5-2	8 × 8	151.02M	16.68M	6.9	2.36M	260.69K	11.1
Conv5-3	8 × 8	151.02M	20.62M	13.6	2.36M	322.18K	13.7
FC1	1	8.39M	2.92M	34.8	8.39M	2.92M	34.8
FC2	1	1.05M	1.05M	100.0	1.05M	1.05M	100.0
FC3	1	10.24K	10.24K	100.0	10.24K	10.24K	100.0
Total	-	5.02B	534.29M	10.6	24.17M	5.03M	20.9

This experiment is based on the VGG-16 model and CIFAR10 dataset

3.4 Normalization

One of the innovations of this paper is that the proposed pruning method based on FEAM can compare the feature extraction ability of different layers of convolution filters after normalization. The advantage of this method is that it does not need to set the pruning rate of each convolution layer separately, only need to set the overall pruning rate of the model. In this way, it can avoid the permanent decrease of model accuracy caused by the inaccurate pruning rate of a certain layer. As shown in Fig. 7 of the paper.

For each convolution layer, we compute the FEAM of all filters and get a vector $\Theta = [FEAM_1, \ldots, FEAM_J]$. Most previous pruning methods are realized by compare the elements in such a vector like Θ. However, the scale inconsistency problem may occur when this strategy is applied to the cross-layers. In our method, we normalize the Θ of each layer to the same range [0, 1], and then the pruning operation can be applied to cross-layers. Normalization is defined as:

$$\hat{\Theta} = \frac{\Theta}{\sqrt{\sum_{j=0}^{J}(FEAM_j)^2}} \tag{5}$$

where $\hat{\Theta}$ is the normalized vector of Θ. After the normalization of each layer, we can prune and tune the network.

3.5 Pruning and fine-tuning

The implementation details of pruning and tuning vary with network structures. In this paper, we consider VGG, Resnet and MobileNet, which are typical representatives of traditional convolution and fully connected architectures. As shown in Table 1, more than 39% parameters exist in the fully connected layers of VGG-16. Some papers, e.g. [20], use global average pooling instead of the full connection layer. Although this method greatly reduce the number of parameters, it also reduces the convergence speed of the model, and make it difficult to train the model back to the original accuracy. Therefore, we consider pruning the filter of the last convolution layer to reduce parameters.

For ResNet, there are two kinds of residual modules. One is that two convolution layers with 3×3 kernels are grouped as one residual module, and the other is three convolutional layers with 1×1, 3×3, 1×1 kernels respectively are grouped as a residual module. Our pruning method for the two residual modules are shown in Fig. 3. Please note that there is a restriction in the pruning process of ResNet, identity shortcut connection, which requires that the number of output channels in the same group is consistent. We can see that our pruning method in Fig. 3 meets this requirement.

Our pruning method for MobileNet is shown in Fig. 4. For MobileNet V1 [29], there is not shortcut connection in the simple superimposed block. Because each input channel of the depth-wise convolution layer corresponds to a filter

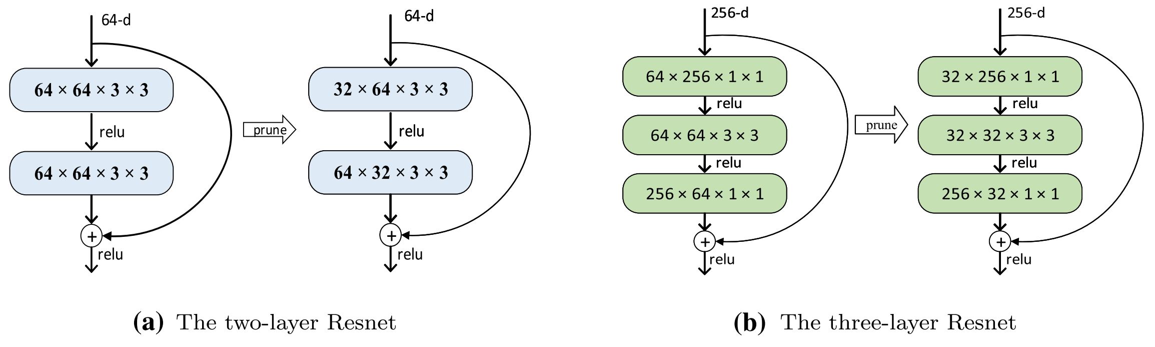

Fig. 3 Our pruning strategy for ResNet. The previous layer of the last layer is pruned

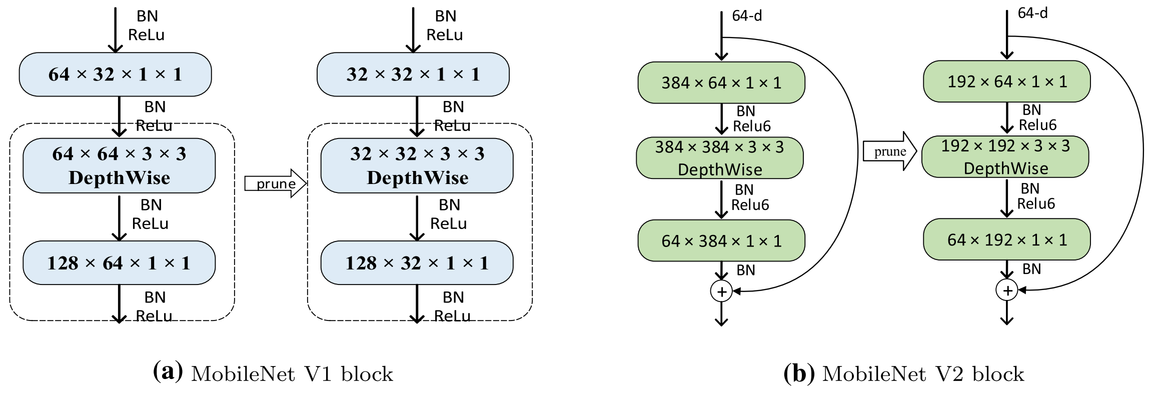

Fig. 4 Our pruning strategy for MobileNet V1 and V2. For each block, we only prune the deep-wise convolution layer of 3×3

separately, the input channel will be pruned at the same time when we prune the filters of the depth-wise convolution layer, thus indirectly prune the 1×1 convolution layer. The pruning method for MobileNet V2 [28] is similar to that for Resnet.

The final problem is how to use fine-tuning during the pruning process. We find that most methods prune and tune the network sequentially, e.g., tuning the network after the pruning is completely finished. There is a problem for this strategy: the network structure may have been damaged before tuning, e.g., the pruning rate is too high. This problem will become more apparent for more complex networks. Based on this fact, we consider pruning and tuning the network iteratively. In such cases, the advantages of fine-tuning can be better played, and the destruction of the network structure may also be avoided.

First, we preset a pruning ratio α for the whole network, assuming that α is equal to 0.3. There are altogether L filters in the whole network, if L is equal to 3968, then 3968×0.3 filters need to be pruned eventually. Instead of pruning all 3968×0.3 filters at once, we pruned them in batches. In each pruning, we prune β filters, e.g. 256 filters, from the network, which selected the filters with bottom β FEAM scores. After pruning we conduct fine-tuning a few epochs to recover the performance slightly. Then the pruning followed by tuning operation is repeated again and again. The total number of this iteration is $(L \times \alpha)/\beta$ times.

4 Experiments

This section consists of two parts: quantitative analysis and qualitative analysis. In quantitative analysis, we verify the effectiveness of FEAM by visualizing feature maps. In qualitative analysis, we test the proposed method on two popular datasets.

4.1 Quantitative analysis

To quantitatively evaluate the proposed method, we visually observe the output feature maps and corresponding FEAM values. Generally speaking, if a visually significant feature map is accompanied by a large FEAM, the method is valid.

Given a single training sample (No. 1 Sample), Fig. 3 shows output feature maps and their FEAM values. Each feature map in Fig. 3 corresponds to a filter in the second convolution layer of VGG16 network. We can see that the proposed FEAM is indeed valid: the more significant the output feature map, the larger the FEAM value. However, there are also anomalous examples, such as the feature map

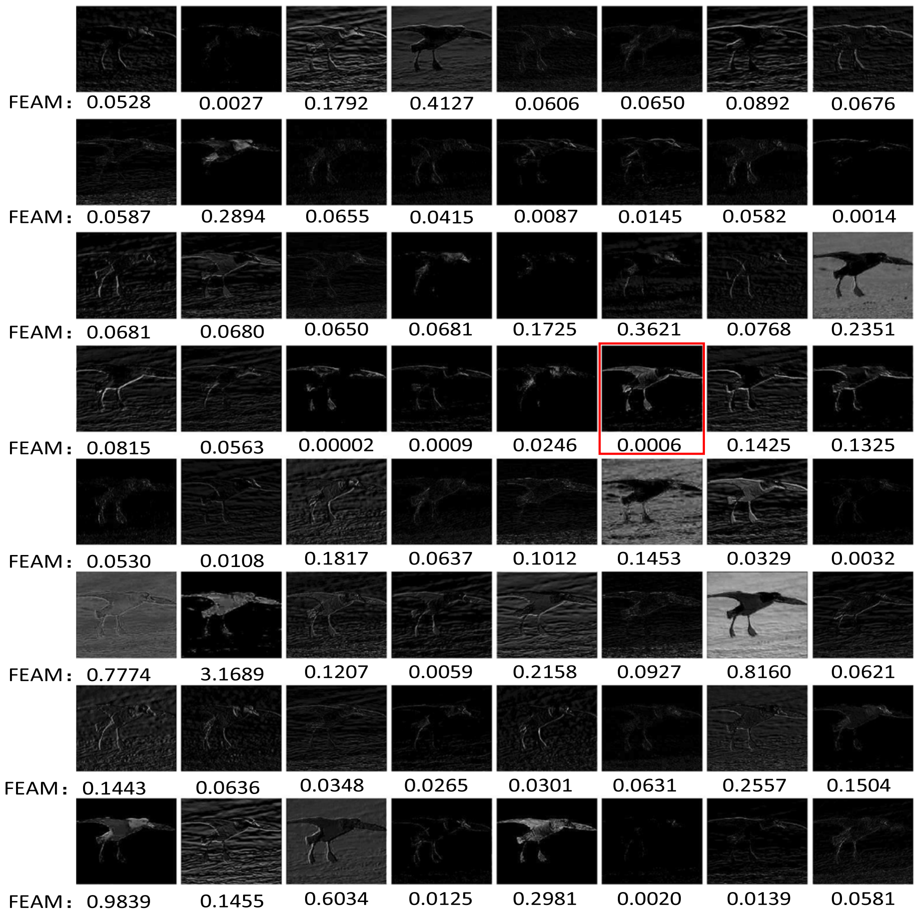
FEAM: 0.0528 0.0027 0.1792 0.4127 0.0606 0.0650 0.0892 0.0676
FEAM: 0.0587 0.2894 0.0655 0.0415 0.0087 0.0145 0.0582 0.0014
FEAM: 0.0681 0.0680 0.0650 0.0681 0.1725 0.3621 0.0768 0.2351
FEAM: 0.0815 0.0563 0.00002 0.0009 0.0246 0.0006 0.1425 0.1325
FEAM: 0.0530 0.0108 0.1817 0.0637 0.1012 0.1453 0.0329 0.0032
FEAM: 0.7774 3.1689 0.1207 0.0059 0.2158 0.0927 0.8160 0.0621
FEAM: 0.1443 0.0636 0.0348 0.0265 0.0301 0.0631 0.2557 0.1504
FEAM: 0.9839 0.1455 0.6034 0.0125 0.2981 0.0020 0.0139 0.0581

Fig. 5 The feature maps and corresponding FEAM values of the second convolution layer of VGG16 network

marked with red rectangle in Fig. 3. This feature map is relatively significant, but the FEAM value is small (Fig. 5).

For the filter corresponding to the red rectangle in Fig. 3, the output feature maps of ten training samples are shown in Fig. 6. We can see that the output feature maps of most samples are not as significant as that of No. 1 Sample, which reflects this filter is over-fitted, and it is not a good filter. This case is reflected in GAP values and finally reflected in FEAM value. All GAP values in Fig. 6 are in a small range [0, 0.11], and hence the feature dispersion is small. Accordingly, the FEAM value of this filter is discounted by feature dispersion, as shown in in Fig. 3.

4.2 Qualitative analysis

The performance of the proposed method is evaluated within four typical CNN models: VGG-16, ResNet-50, MobileNet V1 and MobileNet V2. Two public datasets, CIFAR-10 and CUB_ 200_2011, are used as benchmark. The CIFAR-10 dataset consists of 10 categories, and each category includes 6000 images. The image resolution of this dataset is 32×32. Considering that the deceleration is not obvious for such small resolution due to pruning, we convert the resolution of CIFAR-10 to 128×128 before training. The Cub_200_2011 is a bird dataset for classification task, which contains 11788 images of 200 different bird species. All samples in this dataset are resized to 320×320 before training. After each round of pruning, we fine-tune the whole network in 8 epochs with learning rate varying from 10^{-3} to 10^{-5}. During the last pruning, the network is fine-tuned in 20 epochs with learning rate varying from 10^{-3} to 10^{-8}. All experiments are conducted on a computer equipped with Nvidia GTX 1080Ti GPU.

4.2.1 VGG-16 pruning

The detailed distribution of FLOPs and parameters in each layer of VGG-16 is shown in Table 1. As we have seen, convolutional layer from the 2nd to 12th contains 90% FLOPs. And we can see that our pruning method is also mainly aimed at layer 3–12. For the first two layers of convolutional layer, there is no large-scale pruning. We think that the first two layers of filters contain rich feature information, so they have stronger Feature Abstraction Capability than other filters in the layer. The side proves that our method has a certain degree of interpretability. The pruning rate we set is 80%, that is, 80% of the filters in the model are pruned off. Finally,

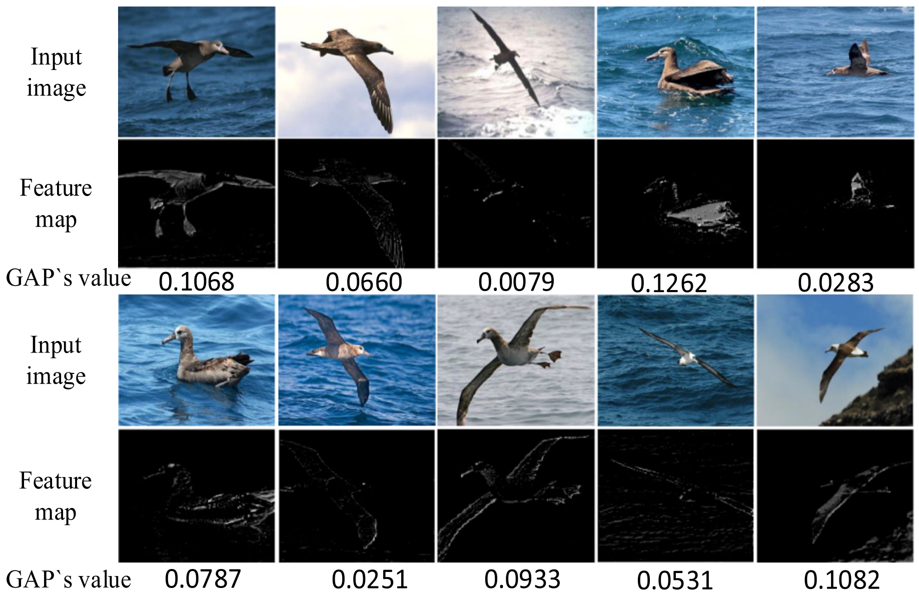

Fig. 6 For the filter corresponding to the red rectangle in Fig. 5, the output feature maps of ten training samples and corresponding GAP values

Table 2 Comparison of different model compression methods for VGG-16 network on CIFAR10

Method	Top-1 Acc	FLOPs (%)	Params (%)	Compression
Original	0.916	5.02B	24.17M	1.0×
APoZ-50%	0.912	1.27B (25.3%)	8.39M (34.7%)	2.7×
APoZ-80%	0.803	0.19B (3.8%)	3.34M (13.8%)	5.4×
Taylor-50%	0.922	1.42B (28.2%)	6.94M (28.71%)	3.8×
Taylor-80%	0.908	0.56B (11.2%)	1.03M (4.3%)	6.1×
Entropy_GAP	0.868	1.56B (31.1%)	4.22M (17.5%)	4.8×
FEAM-50% (product)	0.925	1.26B (25.1%)	11.09M (45.9%)	2.0×
FEAM-80% (product)	0.913	0.53B (10.6%)	5.03M (20.8%)	4.9×
FEAM-50% (sum)	0.918	1.46B (29.2%)	9.98M (41.3%)	2.8×
FEAM-80% (sum)	0.904	0.67B (13.4%)	4.21M (17.4%)	5.1×

we compare our method with the following baselines on the VGG-16 model:

Taylor expansion [19]: The effect of the filter on the network loss function is calculated based on the Taylor expansion method. According to this criterion, if the filter has little effect on the loss function, the filter can be safely removed.

APoZ [18]: The criterion measures the importance of filters by calculating the average percentage of zeros(APoZ) in the activated feature map.

Entropy [20]: The method calculates entropy of filters to measure the information richness of the convolution kernel.

As shown in Tables 2 and 3, we used different algorithms for pruning VGG-16 networks in CIFAR10 and CUB_200_2011 datasets, among which the APOZ method pruned the filter of each layer with a fixed prune rate. We can see that when the pruning rate reaches 80%, the accuracy of the model drops very seriously. In the Entropy method, Luo et al. used GAP instead of the fully connected layer, which greatly reduced the parameters of the model, but had a greater impact on the prediction accuracy of the model (which greatly increased the difficulty of convergence of the model). The Taylor method uses a pruning strategy similar to ours. It can be seen that the Taylor criterion has better performance on model size compression, but at the same pruning rate, our method has less precision loss and more excellent acceleration performance.

In the design process of the algorithm, we tried a variety of different combinations, including product and weight sum. According to the experimental results, the product of the two is chosen as the combination method, which is more robust and sensitive.As can be seen from the two tables, the larger the size of the input image, the greater the clipping acceleration. In the CUB_200_2011 dataset, the size of the input image is 320 × 320. Our method can achieve about 4.7 × reduction in FLOPs and parameters with only 0.006 decrease in accuracy. When the pruning rate is 50%, the accuracy of the pruned model is even higher than the original model, and

Table 3 Comparison of different model compression methods for VGG-16 network on CUB_200_2011

Method	Top-1 Acc	Top-5 Acc	FLOPs (%)	Params (%)	Compression
Original	0.764	0.939	31.64B	304.1M	1.0×
APoZ-50%	0.756	0.932	12.49B (39.5%)	104.3M (30.3%)	5.7×
APoZ-80%	0.556	0.833	1.40B (4.4%)	80.6M (26.5%)	7.2×
Taylor-50%	0.772	0.948	13.96B (44.1%)	91.8M (30.2%)	6.2×
Taylor-80%	0.744	0.921	7.47B (23.6%)	55.7M (18.4%)	11.3×
Entropy-50%	0.728	0.932	9.42B (29.8%)	64.6M (21.2%)	9.1×
FEAM-50% (product)	0.786	0.950	11.62B (36.7%)	122.6M (40.3%)	2.9×
FEAM-80% (product)	0.758	0.937	6.74B (21.3%)	66.6M (21.9%)	8.6×
FEAM-50% (sum)	0.763	0.941	12.38B (39.1%)	109.3M (35.9%)	3.1×
FEAM-80% (sum)	0.735	0.928	7.64B (24.2%)	64.1M (21.1%)	9.0×

Table 4 The pruned model for ResNet-110 on CIFAR-10 with different pruning rate

Model	Top-1 Acc	Speed-up		Compression	
		#FLOPs	FLOPs%	#Para.	Para. %
ResNet110	0.937	2.53×10^8	–	1.72×10^6	–
FEAM-20% (product)	0.933	1.46×10^8	57.7	1.02×10^6	59.3
FEAM-30% (product)	0.928	1.02×10^8	40.3	0.72×10^6	41.9
FEAM-20% (sum)	0.926	1.67×10^8	66.0	0.84×10^6	48.8
FEAM-30% (sum)	0.901	1.24×10^8	49.1	0.68×10^6	39.6

Table 5 The pruned model for MobileNet V1 and V2 on CUB_200_2011 with different pruning rate

Method	Top-1 Acc	Top-5 Acc	FLOPs (%)	Params (%)
MobileNet V1	0.736	0.915	1.26B (100%)	3.40M (100%)
FEAM-30%	0.771	0.926	0.804B (63.8%)	2.51M (73.82%)
FEAM-50%	0.732	0.910	0.555B (44.1%)	1.21M (35.59%)
FEAM-60%	0.688	0.878	0.492B (39.04%)	0.76M (28.23%)
MobileNet V2	0.743	0.939	1.07B (100%)	5.73M (100%)
FEAM-30%	0.736	0.833	0.883B (82.5%)	5.20M (90.75%)
FEAM-50%	0.649	0.948	0.796B (74.4%)	4.52M (78.89%)

higher than other pruning methods. Through these comparisons, we can see that our FAC-based pruning method has better overall performance.

4.2.2 ResNet-110 pruning

For the network structure of ResNet-110, it is divided into three hierarchies by the residual block, and the size of its corresponding feature maps are 32 × 32, 16 × 16, and 8 × 8, respectively. According to the process of pruning for ResNets in Sect. 3.4, the pruned model for ResNet-110 was obtained on CIFAR-10. During the training process, the images are randomly cropped to 32 × 32.

The overall performance of our method on pruning ResNet-110 is shown in Table 4, We prune this model with 2 different pruning rates (pruning 20%, 30% filters respectively). The best pruned model achieves 2.48× reduction in FLOPs and parameters with only 0.007 decrease in accuracy. Unlike traditional CNN architectures, ResNet is more compact. There is less redundancy than the VGG-16 model, so it seems more difficult to delete a large number of filters. However, when small pruning rate is adopted, our method can improve the performance of ResNet-110 to the maximum extent.

4.2.3 MobileNet pruning

As shown in Table 5, our pruning strategy still works well in MobileNet V1 and V2 lightweight networks. When the pruning rate is 30% for MobileNet V1, the Top1 accuracy of the network after pruning increases by 0.025, the FLOPs decreases by 35.2%, and the number of parameters decreases by 38.7%. When the pruning rate is 50%, the Top1 accuracy of pruning network is decreased by 0.004, while the FLOPs

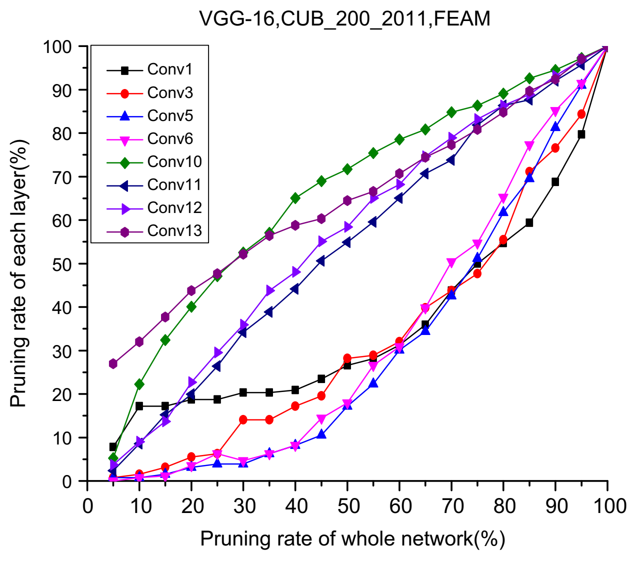

Fig. 7 The pruning rate curve of each convolutional layer under different overall network pruning rates

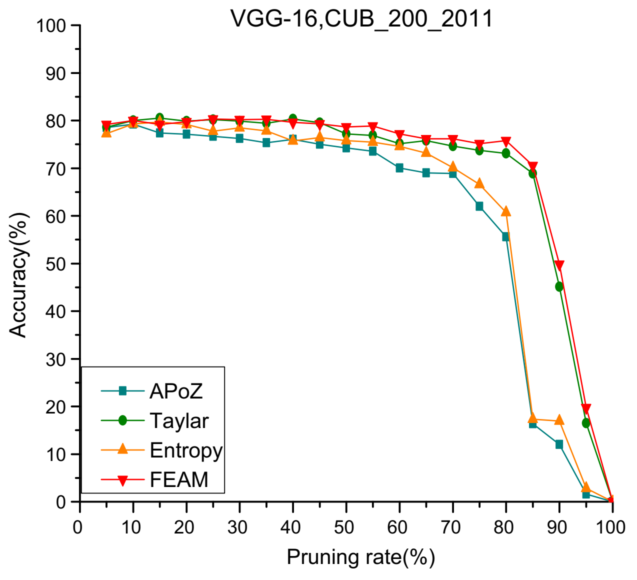

Fig. 8 The curve between pruning rate and network accuracy of different pruning methods on CUB_200_2011

is decreased by 56.3%, and the parameters are decreased by 73.1%.

4.2.4 Comparison

Figure 7 shows the pruning rate curve of some convolution layer under different overall pruning rates. We can see that under the same overall pruning rate, the pruning rate increases roughly with the network deepening. This is in line with common sense: the deeper layers are based on shallower layers in the whole network. In such cases, pruning in deeper layers has less impact on the network than pruning in shallower layers. The same phenomenon is also observed in Table 1.

Figure 8 compares the network accuracy of different pruning methods. When the pruning rate is low, the performance of the proposed method is roughly equivalent to Taylor, but when the pruning rate is high, the proposed method is slightly better than Taylor. At all pruning rates, the proposed method performs better than APoZ and Entropy.

5 Conclusion

A new pruning method based on FEAM is introduced in this work. FEAM is a novel index that measures the feature extraction ability of filters in deep neural networks. FEAM includes two components, kernel sparsity and feature dispersion, which evaluate the feature extraction ability from theoretical analysis and practical operation, respectively, and then competitively decide the final feature extraction ability of filters. Filters with weak FEAM values are pruned from the network. Experiments show that the proposed pruning method is effective for multiple networks. Future research direction is considering the proposed pruning method under specific application scenarios, e.g., segmentation.

References

1. Girshick, R.: Fast R-CNN. In: The IEEE International Conference on Computer Vision (ICCV) (2015)
2. Redmon, J., Farhadi, A.: YOLO9000: better, faster, stronger. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
3. Hu, H., Gu, J., Zhang, Z., et al.: Relation networks for object detection. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
4. He, K., Gkioxari, G., Dollr, P., et al.: Mask R-CNN. In: The IEEE International Conference on Computer Vision (ICCV) (2017)
5. He, K., Zhang, X., Ren, S., et al.: Deep residual learning for image recognition. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
6. Huang, G., Liu, Z., Van Der Maaten, L. et al.: Densely connected convolutional networks. The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
7. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015)
8. Xu, Y., Fu, T., Yang, H., et al.: Dynamic video segmentation network. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
9. Kuhn, M., Johnson, K.: An introduction to feature selection. In: Applied Predictive Modeling. Springer, Berlin (2013)
10. Krizhevsky, A., Sutskever, I., Hinton, G.E..: Imagenet classification with deep convolutional neural networks. In: International Conference on Neural Information Processing Systems (NIPS) (2012)
11. Simonyan, K., Zisserman, A.: Imagenet classification with deep convolutional neural networks. In: International Conference on Learning Representations (ICLR) (2015)

12. Lin, S., Ji, R., Chen, C., et al.: Holistic CNN compression via low-rank decomposition with knowledge transfer. In: IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) (2018)
13. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Coordinating filters for faster deep neural networks. In: The IEEE International Conference on Computer Vision (ICCV) (2017)
14. Wu, J., Leng, C., Wang, Y., et al.: Quantized convolutional neural networks for mobile devices. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
15. Jacob, B., Kligys, S., Chen, B., et al.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
16. Lin, X., Zhao, C., Pan, W.: Towards accurate binary convolutional neural network. In: International Conference on Neural Information Processing Systems (NIPS) (2017)
17. Han, S., Mao, H., Dally, W.J.: Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. In: International Conference on Learning Representations (ICLR) (2016)
18. Hu, H., Peng, R., Tai, Y.W., Tang, C.K.: Network trimming: a data-driven neuron pruning approach towards efficient deep architectures. arXiv preprint arXiv:1607.03250 (2016)
19. Molchanov, P., Tyree, S., Karras, T., et al.: Pruning convolutional neural networks for resource efficient inference. In: International Conference on Learning Representations (ICLR) (2017)
20. Luo, J., Wu, J.: An entropy-based pruning method for CNN compression. arXiv preprint arXiv:1706.05791 (2017)
21. Zhou, B., Sun, Y., Bau, D., et al: Revisiting the importance of individual units in CNNs via ablation. arXiv preprint arXiv:1806.02891 (2018)
22. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (ICLR) (2015)
23. He, K., Zhang, X., Ren, S., et al.: Deep residual learning for image recognition. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016)
24. Li, H., Kadav, A., Durdanovic, I., et al.: Pruning filters for efficient convnets. In: International Conference on Learning Representations (ICLR) (2016)
25. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: International Conference on Computer Vision (ICCV) (2017)
26. Lin, M., Chen,Q., Yan, S.: Network in network. arXiv preprint arXiv:1312.4400 (2013)
27. Son, S., Nah, S., Mu Lee, K.: Clustering convolutional kernels to compress deep neural networks. In: European Conference on Computer Vision (ECCV) (2018)
28. Sandler, M., Howard, A.G., Zhu, M., et al.: Mobilenetv2: inverted residuals and linear bottlenecks. In: Computer Vision and Pattern Recognition (CVPR) (2018)
29. Howard, A.G., Andrew, G., Zhu, M., et al.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)

H. Wu obtained his PhD degree from University of Electronic Science and Technology of China in 2007. He is the director of the Research and Development Center of the Second Institute of Civil Aviation Administration of China. His research interests are computer vision and intelligent aviation

Y. Tang is a master student in the School of Information and Communication at Electronic Science and Technology of China. His research interests are computer vision and intelligent aviation

X. Zhang obtained his PhD degree from Shanghai Jiaotong University of China in 2010. He is an Associate Professor in the School of Information and Communication at Electronic Science and Technology of China. His research interests are computer vision and intelligent aviation