



Using Feature Entropy to Guide Filter Pruning for Efficient Convolutional Networks

Yun Li[✉], Luyang Wang, Sifan Peng, Aakash Kumar, and Baoqun Yin^(✉)

Department of Automation, University of Science and Technology of China,
Hefei, China

{yli001,ly1105,sifan,akb}@mail.ustc.edu.cn

bqyin@ustc.edu.cn

Abstract. The rapid development of convolutional neural networks (CNNs) is usually accompanied by an increase in model volume and computational cost. In this paper, we propose an entropy-based filter pruning (EFP) method to learn more efficient CNNs. Different from many existing filter pruning approaches, our proposed method prunes unimportant filters based on the amount of information carried by their corresponding feature maps. We employ entropy to measure the information contained in the feature maps and design features selection module to formulate pruning strategies. Pruning and fine-tuning are iterated several times, yielding thin and more compact models with comparable accuracy. We empirically demonstrate the effectiveness of our method with many advanced CNNs on several benchmark datasets. Notably, for VGG-16 on CIFAR-10, our EFP method prunes 92.9% parameters and reduces 76% float-point-operations (FLOPs) without accuracy loss, which has advanced the state-of-the-art.

Keywords: Convolutional neural networks · Filter pruning · Entropy · Features selection module

1 Introduction

In recent years, we have witnessed a rapid development of deep neural networks in many computer vision tasks such as image classification [6], semantic segmentation [16, 19] and object detection [3]. However, as the CNN architectures tend to be deeper and wider to get superior performance, the number of parameters and convolution operations also increase rapidly. For instance, Resnet-164 has nearly 2 million parameters and VGG-16 requires more than 500 MB storage space. These cumbersome models significantly exceed the computing limitation of current mobile devices.

Considerable research efforts have been devoted to compressing large CNN architectures. Pruning is an intuitive CNN compression strategy and it mostly focuses on removing unimportant network connections. The current pruning

methods usually include directly deleting weight values of filters [4, 22] and totally pruning some filters [7, 12, 15, 24]. The weight value pruning methods introduce non-structured sparsity in the parameter tensors and require dedicated sparse matrix operation libraries. In contrast, the filter level pruning is a naturally structured way of pruning without introducing sparsity and thus does not require sparse libraries or specialized hardware. Therefore, filter pruning attracts more attention in accelerating CNN architectures.

However, most of the previous researches on filter pruning only focus on the activation values or the scale factors weighted on the output of filters and fail to consider the amount of information carried by the feature maps. The feature maps are the most direct reflection of the usefulness of convolution filters. Previous works [12, 14] have shown that feature maps are sparse and a considerable number of feature maps output by the middle layers of CNN are most of zeros or zero matrices. Regardless of the given scale factor, the feature maps with all zero values cannot make contribution to the accuracy of the model. On the other hand, if given a small scale factor, a feature map containing a large amount of information will be pruned, which may lead to some important information loss.

In this paper, we propose an entropy-based filter pruning (EFP) method to address the above-mentioned problem. Our EFP selects unimportant filters based on the amount of information contained in their corresponding feature maps. We employ entropy [21] to measure the information carried by feature maps, since it plays a central role in information theory as measures of information and uncertainty. Some similar works proposed to prune the network based on entropy. [13] proposed to calculate the filter entropy, and failed to consider that the amount of information in the filter is unexplained compared with the feature map. [17] calculated the entropy of the global mean of the feature map, which is a rough measure and ignores the spatial information in the feature map. However, our proposed feature entropy method is different from them and can overcome their weakness, since we choose to expand the feature maps by row and calculate their spatial information entropy to weigh the effectiveness of the corresponding filters. Then, we design features selection module to extract the output of every filter and determine their entropy weights. These modules are placed between every two adjacent convolutional layers of a well-trained network, as shown in Fig. 1. Those filters whose output feature maps are given small weights will be pruned. After pruning, we fine-tune the compact model to restore performance and can even achieve a higher accuracy in many cases. Finally, the pruning and fine-tuning process are repeated for several times to get an even more compact network. Furthermore, we also research the correspondence between the entropy pruning ratio and the number of filters to explore the distribution of information in each convolutional layer of a CNN architecture.

We evaluate our method on several benchmark datasets and different CNN architectures. For VGG-16 on CIFAR-10, we achieve 92.9% of parameters pruning and 76% float-point-operations (FLOPs) reduction with 0.04% accuracy improvement, which has advanced the state-of-the-art. For the less redundant

ResNet-56 and ResNet-164, we also gain 50.8% and 52.9% parameters reduction, respectively, without notable accuracy loss.

2 Related Work

Most previous works on deep CNN compression can be roughly divided into 4 categories, matrix decomposition, weight quantization, architecture learning and model pruning.

Matrix decomposition was proposed to approximate weight matrix of deep CNN tensor with sparse decomposition and low-rank matrix [23], using techniques like Singular Value Bounding [10]. While these methods can reduce the computational cost, the compression of the parameters is very limited.

Some works [1, 2, 4] proposed to quantize the filter weights. The network weights were quantized to several groups and all the weights in the same group shared the same value, only the effective weights and indices need to be stored. This method can achieve a large compression ratio in terms of parameters. However, the FLOPs of the network cannot be reduced, since shared weights need to be restored to the original positions during the process of calculation.

Some other works [8, 25] proposed to learn the CNN architecture automatically. [25] trained RNN with reinforcement learning to maximize the expected accuracy of the generated architectures on a validation set. [8] designed AutoML for Model Compression which leverages reinforcement learning to sample the model design space and achieves the model compression. The search space of these strategies is extremely large and they need to train models for a long time to determine the best strategy.

Pruning is an intuitive model compression method. [4] proposed an iterative connection pruning method by pruning unimportant connections whose weights are below the threshold. [22] regularized the structures by group Lasso penalty leading to a compact structure. However, pruning weights always bring unstructured models which are not implementation friendly and the FLOPs reduction is very limited. To overcome these limitations, some filter pruning methods have been explored. [9, 12, 15] leveraged L1-based methods to select unimportant filters and channels. [18] used statistics information from next layer to evaluate and prune filters. [24] utilized LSTM to select convolutional layers and then evaluated the filters of selected layers. [7] proposed a soft filter pruning method which updates the filters to be pruned after each training epoch. These methods usually require less dedicated libraries or hardwares, as they pay attention to pruning the network structures instead of individual connection of filters. Our proposed entropy-based method also falls into this category, achieving not only parameters reduction but also FLOPs saving without special libraries designed.

3 Method

In this section, we will give a detailed description of our entropy-based filter pruning method. First, we introduce how to determine the entropy weights of

feature maps in Sect. 3.1. Next, our feature maps selection and filter pruning strategies are presented in Sect. 3.2. Finally, the analysis of computational cost compression is illustrated in Sect. 3.3.

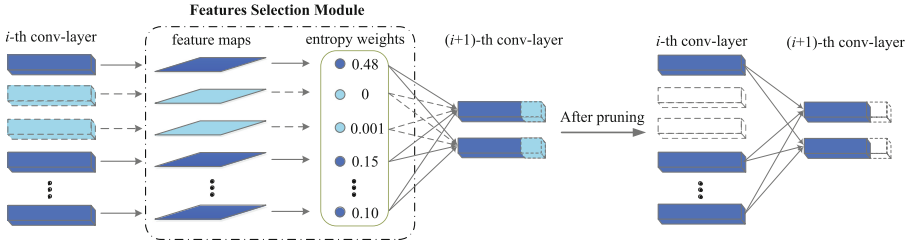


Fig. 1. Illustration of our proposed method which prunes filters based on entropy. We insert features selection module between each two adjacent convolutional layers of the well-trained network (left side). For the i -th layer, the output feature maps of convolution filters are extracted and input to the entropy module to determine their entropy weights. Then those feature maps with smaller weights indicate that they contain less information and the corresponding filters will be pruned (right side). Meanwhile, the corresponding channels of each filter in $(i + 1)$ -th layer will also be pruned to be consistent with input. All the convolutional layers are pruned in parallel.

3.1 Determining Entropy Weights of Feature Maps

Most of the previous works [9, 12, 15] determine the importance of filters by L1 sparsity or scale factors and ignore the amount of information carried by the feature maps. Some previous works [12, 14] have shown that quite a number of feature maps output by the intermediate layers of CNN are zero matrices or most of zeros, which reveals that not all the filters in the model are useful.

To judge the effectiveness of filters, we employ entropy to measure the information in feature maps. Entropy plays a central role in information theory as measures of information and uncertainty and it is proportional to the amount of information [17]. Considering that the outputs of different convolutional layers have significant differences in the amount of information, the weights of feature maps are determined in each layer independently. Specially, to avoid the contingent result of a single image, we randomly select a large number of images from the training dataset to calculate the average entropy weights of filters. Let H_i/W_i denote the height/width of the output feature maps and m_i be the number of filters of the i -th convolutional layer, in which one filter generates one feature map. N denotes the number of images randomly fed into the network. For n -th image, let $X_{i,k}^{(n)}$ be the k -th output feature map matrix of layer i , expanded by row and forms a feature map vector:

$$\hat{X}_{i,k}^{(n)} = \left(x_{i,k,1}^{(n)}, x_{i,k,2}^{(n)}, \dots, x_{i,k,L_i}^{(n)} \right), \quad (1)$$

in which $L_i = H_i \times W_i$. Normalize $\hat{X}_{i,k}^{(n)}$ by Eq. (2), we gain $P_{i,k}^{(n)}$.

$$p_{i,k,l}^{(n)} = \frac{x_{i,k,l}^{(n)} - \min_l \{x_{i,k,l}^{(n)}\}}{\max_l \{x_{i,k,l}^{(n)}\} - \min_l \{x_{i,k,l}^{(n)}\}} \quad (2)$$

$$P_{i,k}^{(n)} = \left(p_{i,k,1}^{(n)}, p_{i,k,2}^{(n)}, \dots, p_{i,k,L_i}^{(n)} \right) \quad (3)$$

Next, for n -th image and i -th convolutional layer, the entropy of the the k -th feature map vector is defined as:

$$E_{i,k}^{(n)} = -q_i \sum_{l=1, f_{i,k,l}^{(n)} > 0}^{L_i} f_{i,k,l}^{(n)} \ln f_{i,k,l}^{(n)}, k = 1, 2, \dots, m_i, \quad (4)$$

in which $f_{i,k,l}^{(n)} = p_{i,k,l}^{(n)} / \sum_{l=1}^{L_i} p_{i,k,l}^{(n)}$, $q_i = 1 / \ln L_i$.

Then, for the output of the i -th convolutional layer, the entropy weight of the k -th feature map can be defined as:

$$w_{i,k}^{(n)} = \frac{E_{i,k}^{(n)}}{\sum_{k=1}^{m_i} E_{i,k}^{(n)}}, \quad (5)$$

in which $0 \leq w_{i,k}^{(n)} \leq 1$, $\sum_{k=1}^{m_i} w_{i,k}^{(n)} = 1$.

Afterwards, we get the average entropy weight of the the k -th feature map:

$$w_{i,k} = \frac{\sum_{n=1}^N w_{i,k}^{(n)}}{N}, \quad (6)$$

in which $0 \leq w_{i,k} \leq 1$, $\sum_{k=1}^{m_i} w_{i,k} = 1$.

Eventually, using the the algorithm described above, we can determine all the entropy weights of feature maps in each convolutional layer.

3.2 Filter Pruning Strategies

In order to identify the less useful filters from a well-trained model, the features selection module is designed and inserted between each two adjacent convolutional layers of the model. As shown in Fig. 1, the output of i -th convolutional layer is fed into entropy weights module to determine the weights of every feature map via the algorithm described in Sect. 3.1. The low weights indicate there is less information in these feature maps and the corresponding filters of i -th convolutional layer are less useful. Then we can prune the feature maps with low weights by removing all their incoming and outgoing connections. By doing so, all the less important filters of the i -th layer and feature maps fed into the $(i+1)$ -th layer are pruned, as well as the corresponding channels of each filter in the $(i+1)$ -th layer.

Determining Pruning Thresholds. In each convolution layer, it is essential to determine the pruning threshold based on the given entropy pruning ratio. Firstly, entropy weights of feature maps in each layer are sorted in ascending order. Then they are accumulated from the smallest weights until the given entropy pruning ratio is exceeded. The last superimposed weight is used as the threshold of the corresponding layer, and all the feature maps and corresponding filters whose entropy weights are lower than the threshold will be pruned. After that, we obtain a more compact network with fewer parameters, less storage and less run time consumption.

Iterative Pruning and Fine-Tuning. There may be some temporary accuracy loss after pruning, but it can be largely compensated by the following fine-tuning process. After that, we can even achieve a higher accuracy than the original one. For the whole network pruning, previous works usually prune and fine-tune the filters layer by layer [4, 18], or retrain the network after each pruning and fine-tuning process [15]. Considering that these strategies are quite time-consuming, our method prunes all layers in parallel, followed by fine-tuning to compensate any loss of accuracy. Moreover, we prune and fine-tune the network iteratively and there is no need to retrain the network from scratch again. The experiments on VGGNet indicate that this strategy is effective. With several iterations, we can achieve a large degree of compression and even lead to a better result.

Adjustment Strategy for Residual Architectures. The proposed filter pruning method can be easily applied to plain CNN architectures such as VGGNet and AlexNet. However, some adjustment strategies are required when it is used to prune complex architectures with cross layer connections such as residual networks [6]. For these architectures, the output of the building block’s last convolutional layer and the identity mapping must be same in size and number of feature maps, which makes them difficult to be pruned. As can be seen from part b of Fig. 2, our features selection modules are placed after the first and second convolutional layers. For the third convolutional layer of the bottleneck block, we only prune the channels of each filter to make them consistent with the input feature maps and do not reduce the number of filters, since the output of it must match the identity maps and there are fewer parameters contained in these 1×1 filters.

3.3 Analysis of Computational Cost Compression

According to Sect. 3.1, the i th convolutional layer takes as input a $W_{i-1} \times H_{i-1} \times m_{i-1}$ tensor of feature maps and produces a $W_i \times H_i \times m_i$ tensor, where m_{i-1} and m_i are the numbers of feature maps. Let us assume the i th convolutional layer is parameterized by $K_i \times K_i \times m_{i-1} \times m_i$, where K_i is the spatial dimension of every filter. Standard convolutions have the computational cost of $K_i \times K_i \times m_{i-1} \times m_i \times W_i \times H_i$. Let r_i denote the entropy pruning rate of i -th layer and \hat{r}_i be the corresponding filter pruning rate. Then the number of filters of the i -th layer will be reduced from m_i to $m_i(1 - \hat{r}_i)$, and the channels of filters in

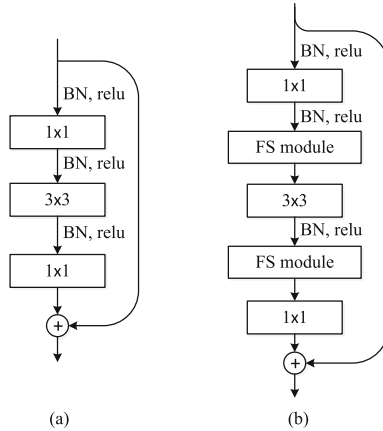


Fig. 2. Illustration of the strategy to prune residual networks. (a) Original bottleneck block of ResNet, (b) bottleneck block with features selection (FS) modules. The BN is placed before each convolutional layer, and FS module is placed after ReLU.

this layer are reduced from m_{i-1} to $m_{i-1}(1 - \hat{r}_{i-1})$. Afterwards, we can get the compression ratio in computational cost for this pruned layer:

$$\begin{aligned}
 & 1 - \frac{K_i^2 \times m_{i-1}(1 - \hat{r}_{i-1}) \times m_i(1 - \hat{r}_i) \times W_i \times H_i}{K_i^2 \times m_{i-1} \times m_i \times W_i \times H_i} \\
 & = 1 - (1 - \hat{r}_{i-1})(1 - \hat{r}_i).
 \end{aligned} \tag{7}$$

4 Experiments

We evaluate our proposed EFP on several benchmark datasets and networks: VGG-16 on CIFAR10 and CIFAR100, ResNet-56 on CIFAR-10, ResNet-164 on CIFAR-100. Both CIFAR datasets [11] contain 50000 training images and 10000 test images. The CIFAR-10 dataset is categorized into 10 classes, and the CIFAR-100 is categorized into 100 classes. All the experiments are implemented with PyTorch [20] framework on NVIDIA GTX TITAN Xp GPU. Moreover, our method is compared with several state-of-the-art methods [7, 9, 12, 15].

4.1 Implementation Details

Experimental Setting. In the experiments, the initial models are trained from scratch to calculate the accuracies as their baselines. During the training process, all images are cropped randomly into 32×32 with four paddings and horizontal flip is also applied. We use mini-batch size 100 to train and mini-batch size 1000 to test VGGNet, and use mini-batch size 64 to train and mini-batch size 256 to test ResNet. All the models are trained and fine-tuned using SGD for 180 epochs on two datasets. During training and fine-tuning processes, the initial learning

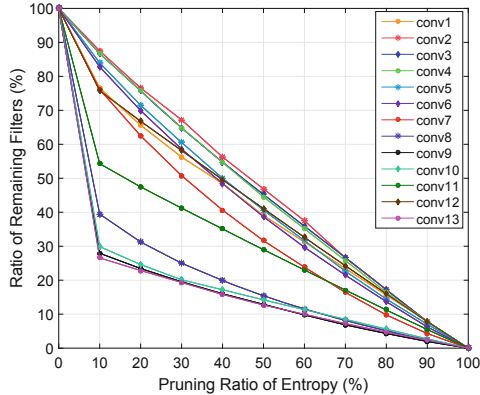


Fig. 3. The pruning results of VGG-16 on CIFAR-10 with entropy pruning rate from 10% to 100%.

rate is set to 0.1, and is divided by 10 at 50% and 75% of the total number of epochs, following the setting in [15]. Besides, the weight initialization described by [5] is also applied.

Entropy Pruning Rate and Filter Pruning Rate. We use our entropy-based method to prune less important filters. In our experiments, 1000 images randomly selected from the datasets are fed into the pre-trained network to gain the average entropy weights of feature maps. Then, we give entropy pruning rates for every convolutional layer to calculate the pruning thresholds via the method described in Sect. 3.2. Figure 3 shows the filter pruning results of VGG-16 on CIFAR-10 with entropy pruning rate from 10% to 100%. The pruning results reveal that information distribution varies between different convolution layers. For some convolutional layers of VGG-16, pruning only 10% information entropy can lead to more than 70% filters reduction.

4.2 Pruning VGGNet

To evaluate the effectiveness of our proposed method on VGG-16, we test it on CIFAR-10 and CIFAR-100 datasets.

VGG-16 on CIFAR-10. Considering that convolutional layers of VGGNet are different in robustness [12] and information concentration, we give each convolution layer a separate pruning ratio. According to the number of filters and the results shown in Fig. 3, the layers of VGG-16 are divided into 3 levels, the layers with less than 128 filters, the layers with 256 filters and the layers with 512 filters. For a given entropy pruning rate r , the rates are set to $0.5r$, r and $1.5r$ for the three levels, respectively. We set $r = 10\%$ and prune VGG-16 model iteratively. As shown in Table 1(a), our proposed EFP achieves a better performance than the other filter pruning methods. With only one iteration, our

Table 1. Pruning results of VGGNet on CIFAR-10 and CIFAR-100. “Baseline” and “Param. baseline” denote the normal accuracy and number of parameters of the original model, respectively. In “Method” column, “iter-1” declares the first iteration of pruning. The pruned ratio of parameters and FLOPs are also shown in column-7&8.

(a) Pruning results of VGG-16 on CIFAR-10						
Method	Baseline	Accuracy	Param. baseline	Parameters	Pruned	Flops saved
Li <i>et al.</i> [12]	93.25%	93.30%	1.5×10^7	5.4×10^6	64.0%	34.2%
Slimming [15]	93.66%	93.80%	–	–	88.5%	51.0%
Ours (iter-1)	93.72%	93.97%	1.5×10^7	3.5×10^5	76.4%	49.5%
Ours (iter-4)	93.72%	93.76%	1.5×10^7	1.0×10^5	92.9%	76.0%
(b) Pruning results of VGG-16 on CIFAR-100						
Method	Baseline	Accuracy	Param. baseline	Parameters	Pruned	Flops saved
Slimming [15]	73.26%	73.48%	–	–	75.1%	37.1%
Ours (iter-1)	73.60%	73.82%	1.5×10^7	9.6×10^6	34.4%	26.2%
Ours (iter-2)	73.60%	73.61%	1.5×10^7	6.4×10^6	56.4%	44.0%

method can prune 50% parameters and even achieve 0.25% accuracy improvement. After four iterations, the parameters saving can be up to 92.9% and the FLOP reduction is 76% with 0.04% accuracy improvement, which has advanced the state-of-the-art.

VGG-16 on CIFAR-100. We use the same setting on VGG-16 to evaluate our method on CIFAR-100. As can be seen from Table 1(b), our model can achieve 44% FLOPs reduction with only two iterations, which is better than the result of [15]. The pruning ratio is not as high as it in CIFAR-10. It is possibly due to the fact that CIFAR-100 contains more classes and it needs more information to classify targets.

4.3 Pruning ResNet

For ResNet architectures, two models ResNet-56 and ResNet-164 with bottleneck structure are utilized to evaluate the proposed method. In bottleneck blocks, the BN layer and ReLU are placed before each convolutional layer. Considering that there are skip connections in the ResNet structures and the information can be shared across the network, we use the same entropy pruning ratio to prune all the layers. Moreover, the feature information can be shared across the network through skip connection, making ResNet structures less sensitive to large-scale pruning, so we prune ResNet-56 and ResNet-164 in a one-shot manner.

ResNet-56 on CIFAR-10. We first prune a medium depth network ResNet-56 on CIFAR-10. The result is compared with several state-of-the-art methods. As shown in Table 2(a), we can gain almost the equal accuracy with the original

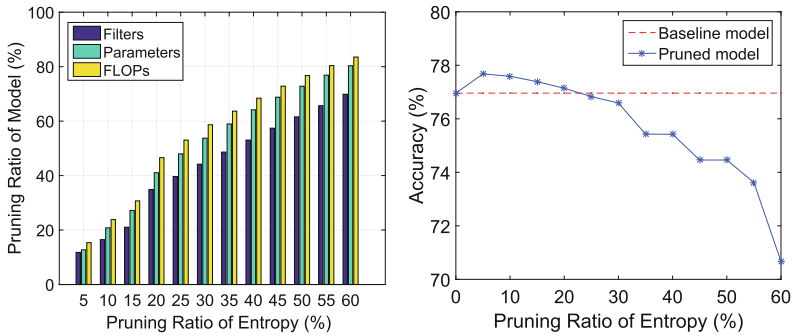
Table 2. Pruning results of ResNet on CIFAR-10 and CIFAR-100 datasets. In “Method” column, “10%” is the one-shot pruning ratio of entropy weights.

(a) Pruning results of Resnet-56 on CIFAR-10						
Method	Baseline	Accuracy	Param. baseline	Parameters	Pruned	Flops saved
Li <i>et al.</i> [12]	93.04%	93.06%	8.5×10^5	7.3×10^5	13.7%	27.6%
He <i>et al.</i> [9]	92.80%	91.80%	–	–	–	50.0%
He <i>et al.</i> [7]	93.59%	93.35%	–	–	–	52.6%
Ours (15%)	94.12%	94.11%	5.9×10^5	4.3×10^5	25.7%	29.4%
Ours (30%)	94.12%	93.31%	5.9×10^5	2.9×10^5	50.8%	53.9%
(b) Pruning results of Resnet-164 on CIFAR-100						
Method	Baseline	Accuracy	Param. baseline	Parameters	Pruned	Flops saved
Slimming1 [15]	76.63%	77.13%	–	–	15.5%	33.3%
Slimming2 [15]	76.63%	76.09%	–	–	29.7%	50.6%
Ours (10%)	76.96%	77.59%	1.7×10^6	1.3×10^6	23.5%	24.0%
Ours (20%)	76.96%	77.14%	1.7×10^6	1.0×10^6	41.2%	46.8%
Ours (30%)	76.96%	76.59%	1.7×10^6	8.0×10^5	52.9%	58.7%

model with about 26% parameters pruned and 30% FLOPs reduced. Moreover, our EFP can also prune 50.8% parameters and 53.9% FLOPs with only 0.79% accuracy drop. It can be observed that unlike VGGNet has a large number of parameters and FLOPs, the bottleneck designed ResNet has less redundancy in parameters and calculations.

ResNet-164 on CIFAR-100. For the deeper network ResNet-164, we adopt the same setting with ResNet-56. Table 2(b) shows that our method outperforms Network Slimming [15]. For ResNet-164, Network Slimming can prune 15% parameters without accuracy loss. When they prune about 30% parameters, there will be 0.54% accuracy drop. However, our proposed method can outperform the original model by 0.18% with 41.2% parameters pruned and 46.8% FLOPs reduced. When we prune 30% entropy weights, our method achieves 52.94% parameters pruned and 58.7% FLOPs saved with only 0.37% accuracy drop.

To comprehensively understand the impact of our proposed method on the model, we test model compression ratio and accuracy of different entropy pruning rates. As shown in Fig. 4, the accuracy of the pruned model first rises above the baseline model and then drops as the entropy pruning rate increases. When the entropy pruning ratio is under 25%, almost 50% parameters are pruned, and our method brings no accuracy loss and even achieves slight accuracy improvement. This result shows our proposed EFP can reduce redundant information and improve the effective expression of features.



(a) Compression ratio of filters, parameters and FLOPs (b) Accuracy of the pruned model

Fig. 4. Pruning results of ResNet-164 on CIFAR-100 regarding different entropy pruning ratios.

5 Conclusion

In this paper, we propose a simple yet effective method, which evaluates the usefulness of convolution filters based on the information contained in the feature maps output by these filters. Our method introduces entropy to measure the amount of information carried by feature maps and evaluate the importance of corresponding convolution filters. Features selection module is designed to formulate pruning strategies. To fit different network structures, some pruning strategies are proposed and address the problem of the dimension mismatch in resnets during pruning. Moreover, the distribution of information in every convolutional layer of CNNs is also discussed and the results demonstrate that in some layers most filters make limited contribution to the performance of the model. Extensive experiments show the superiority of our approach compared to the existing methods. Notably, for VGG-16 on CIFAR-10, our proposed method can prune 92.9% parameters and meanwhile lead to 76% FLOPs reduction without accuracy loss, and this performance has advanced recent state-of-the-art methods.

Acknowledgments. This work was supported by the Equipment Pre-Research Foundation of China under grant No. 61403120201.

References

1. Chen, W., Wilson, J., Tyree, S., Weinberger, K., Chen, Y.: Compressing neural networks with the hashing trick. In: ICML, pp. 2285–2294 (2015)
2. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or -1. arXiv preprint [arXiv:1602.02830](https://arxiv.org/abs/1602.02830) (2016)
3. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: CVPR, pp. 580–587 (2014)

4. Han, S., Mao, H., Dally, W.J.: Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. In: ICLR (2016)
5. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: ICCV, pp. 1026–1034 (2015)
6. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR, pp. 770–778 (2016)
7. He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y.: Soft filter pruning for accelerating deep convolutional neural networks. In: IJCAI (2018)
8. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., Han, S.: AMC: AutoML for model compression and acceleration on mobile devices. In: Ferrari, V., Hebert, M., Sminchisescu, C., Weiss, Y. (eds.) ECCV 2018. LNCS, vol. 11211, pp. 815–832. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01234-2_48
9. He, Y., Zhang, X., Sun, J.: Channel pruning for accelerating very deep neural networks. In: ICCV (2017)
10. Jia, K., Tao, D., Gao, S., Xu, X.: Improving training of deep neural networks via singular value bounding. In: CVPR 2017, pp. 3994–4002 (2017)
11. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Technical report, Citeseer (2009)
12. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. In: ICLR (2017)
13. Li, Y., et al.: Exploiting kernel sparsity and entropy for interpretable CNN compression. In: CVPR (2019)
14. Liu, B., Wang, M., Foroosh, H., Tappen, M., Pensky, M.: Sparse convolutional neural networks. In: CVPR, pp. 806–814 (2015)
15. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: ICCV, pp. 2755–2763 (2017)
16. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: CVPR, pp. 3431–3440 (2015)
17. Luo, J.H., Wu, J.: An entropy-based pruning method for CNN compression. arXiv preprint [arXiv:1706.05791](https://arxiv.org/abs/1706.05791) (2017)
18. Luo, J.H., Wu, J., Lin, W.: ThiNet: a filter level pruning method for deep neural network compression. In: ICCV (2017)
19. Noh, H., Hong, S., Han, B.: Learning deconvolution network for semantic segmentation. In: ICCV, pp. 1520–1528 (2015)
20. Paszke, A., Gross, S., Chintala, S., Chanan, G.: Pytorch: tensors and dynamic neural networks in Python with strong GPU acceleration (2017)
21. Shannon, C.E.: A mathematical theory of communication. *Bell Syst. Tech. J.* **27**(3), 379–423 (1948)
22. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: Advances in Neural Information Processing Systems, pp. 2074–2082 (2016)
23. Yu, X., Liu, T., Wang, X., Tao, D.: On compressing deep models by low rank and sparse decomposition. In: CVPR, pp. 7370–7379 (2017)
24. Zhong, J., Ding, G., Guo, Y., Han, J., Wang, B.: Where to prune: using LSTM to guide end-to-end pruning. In: IJCAI, pp. 3205–3211 (2018)
25. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint [arXiv:1611.01578](https://arxiv.org/abs/1611.01578) (2016)