In *Proceedings of NIPS Workshop on Efficient Methods for Deep Neural Networks*, Barcelona, Spain, December 2016.

Pruning Filters for Efficient ConvNets

Hao Li [*]
University of Maryland, College Park
haoli@cs.umd.edu

Asim Kadav
NEC Labs America
asim@nec-labs.com

Igor Durdanovic
NEC Labs America
igord@nec-labs.com

Hanan Samet
University of Maryland, College Park
hjs@cs.umd.edu

Hans Peter Graf
NEC Labs America
hpg@nec-labs.com

Abstract

We present an acceleration method for CNNs, where we prune filters from convolutional layers that are identified as having a small effect on the output accuracy. By removing whole filters in the network, together with their connecting feature maps, the computational costs are reduced significantly. In contrast to weights pruning, this approach does not result in sparse connectivity patterns. Hence, it does not need the support of sparse convolution libraries and can work with the most efficient BLAS libraries for dense matrix multiplications. We show that even simple filter pruning methods can reduce inference costs for VGG-16 by up to 34% and ResNet-110 by upto 38% while regaining close to the original accuracy by retraining the networks.

1 Introduction

The success of CNNs in various applications is accompanied by a significant increase in the the computation and parameter storage costs [1–3]. These high capacity networks have significant inference costs especially when used with embedded sensors or mobile devices where computational and power resources can be limited. For these applications, in addition to accuracy, computational efficiency and small network sizes are crucial enabling factors [4].

There has been a significant amount of work on reducing the storage and computational cost by parameter pruning [5–10]. Recently Han et al. [8, 9] report 90% compression rates on AlexNet [1] and VGGNet [2] by pruning weights with small magnitudes, followed by retraining without hurting the accuracy. This creates a sparsely populated network of parameters across all layers of CNNs. Though they demonstrate that the convolutional layers can be compressed and accelerated, it additionally requires sparse BLAS libraries or even specialized hardware.

Compared to pruning weights across the network, pruning filters is a naturally structured way of weights pruning without introducing sparsity and therefore does not require using sparse libraries or specialized hardware. Feature map pruning has been previously explored in [11, 12]. However, both the methods need to examine the importance of each feature map or the combination of selected feature mpas before pruning the connected filters, which could be infeasible for pruning CNNs with large number of filters.

To address these challenges, we propose to prune the whole convolutional filters based on a simple criteria from the trained CNNs instead of removing its weights across the network. We then restore the accuracy of the pruned network by retraining. This reduces the computational overheads from convolution layers and does not involve using any sparse libraries or any specialized hardware.

[*] Work done at NEC Labs

In *Proceedings of NIPS Workshop on Efficient Methods for Deep Neural Networks*, Barcelona, Spain, December 2016.

2 Pruning Filters and Feature Maps

Let n_i denote the number of input channels for the ith convolutional layer and h_i/w_i be the height/width of the input feature map. The convolutional layer transforms the input feature maps $\mathbf{x}_i \in \mathbb{R}^{n_i \times h_i \times w_i}$ into the output feature maps $\mathbf{x}_{i+1} \in \mathbb{R}^{n_{i+1} \times h_{i+1} \times w_{i+1}}$, which is used as input feature maps for the next convolutional layer. This is achieved by applying n_{i+1} filters $\mathcal{F}_{i,j} \in \mathbb{R}^{n_i \times k \times k}$ on the n_i input channels, in which one filter generates one feature map. Each filter is composed by n_i two-dimensional kernels $\mathcal{K} \in \mathbb{R}^{k \times k}$ (e.g., 3×3). All the filters, together, constitute the kernel matrix $\mathcal{F}_i \in \mathbb{R}^{n_i \times n_{i+1} \times k \times k}$. The operations of the convolutional layer is $n_{i+1} n_i k^2 h_{i+1} w_{i+1}$. As shown in Figure 1, when a filter $\mathcal{F}_{i,j}$ is pruned, its corresponding feature map $\mathbf{x}_{i+1,j}$ is removed, which reduces $n_i k^2 h_{i+1} w_{i+1}$ operations. The kernels that apply on the removed feature maps from the next convolutional layer are also removed, which saves an additional $n_{i+2} k^2 h_{i+2} w_{i+2}$ operations.

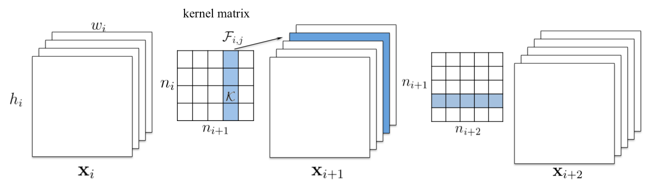
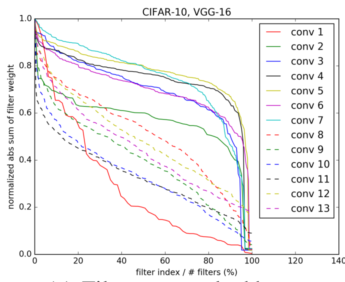


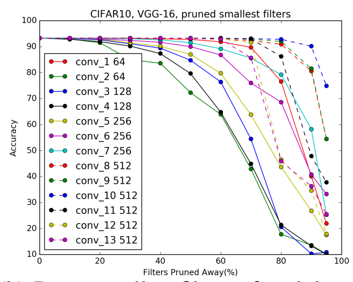
Figure 1: Pruning a filter and its corresponding feature map.

Determining which filters to prune We measure the importance of a filter in each layer by calculating its absolute weight sum $\sum |\mathcal{F}_{i,j}|$. Since the number of input channels, n_i, is the same across filters, $\sum |\mathcal{F}_{i,j}|$ also represents the average magnitude of its kernel weights. This value gives an expectation of the magnitude of the output feature map. Filters with smaller kernel weights tends to produce feature maps with weak activations as compared to the other filters in that layer. Figure 2(a) illustrates the distribution of this filter weight sum for each layer in a VGG-16 network trained using the CIFAR-10 dataset. The filters in a layer are ordered by its sum of kernel weights, which is divided by the maximum value $\max(s_i)$.

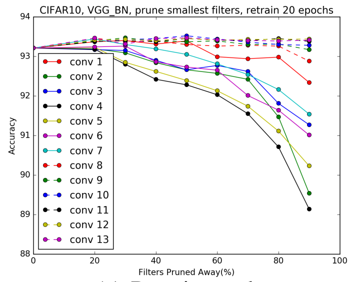
To prune m filters from the ith convolutional layer, we first calculate the absolute sum of its kernel weights $s_j = \sum_{l=1}^{n_i} \sum |\mathcal{K}_l|$ for each filter $\mathcal{F}_{i,j}$. We then sort the filters by s_j and prune m filters with smallest sum values and their corresponding feature maps. The kernels in the next convolutional layer corresponding to the pruned feature maps are also removed. Finally, a new kernel matrix is created for both ith and $i+1$th layer, and the remaining kernels are copied to the new one.



(a) Filters are ranked by s_j



(b) Prune smallest filters of each layer



(c) Retrain result

Figure 2: (a) Sorted filter weight sum for each layer of VGG-16 on CIFAR-10. The x-axis is the filter index divided by the total number of filters; The y-axis is the filter weight sum divided by the max sum value among filters in that layer. (b) Pruning filters with lowest absolute weights sum and their corresponding test accuracy on CIFAR-10. (c) Prune and retrain for each single layer of VGG-16 on CIFAR-10. Some layers are sensitive and it can be harder to recover accuracy after pruning them.

Determining single layer's sensitivity to pruning To understand the sensitivity of each layer, we prune each layer independently and test the resulting pruned network's accuracy on the validation set. Figure 2(b) shows the layers that maintain their accuracy as the filters are pruned away and they correspond to filters with larger slopes in Figure 2(a). On the contrary, layers with relatively flat slopes (as in Figure 2(a)) are more sensitive to pruning. We empirically determine the number of filters to prune for each layer based on their sensitivity to pruning. For layers that are sensitive to pruning, we prune a smaller percentage of these layers or completely skip pruning them.

Pruning filters across multiple layers To prune filters across multiple layers, a naive approach is to determine filters to be pruned at each layer independently. We consider a *greedy* pruning strategy that does not consider the kernels corresponding to the previously pruned feature maps (Figure 3(a)). We find this greedy approach results in pruned networks with higher accuracy especially when large number of filters are used. For more complex network architectures such as Residual networks [3], pruning filters may not be straightforward. We show the filter pruning for residual blocks with projection mapping in Figure 3(b). Here, the filters of the first layer in the residual block can be arbitrarily pruned, as it does not change the number of output feature maps of the block. To prune the second convolutional layer of the residual block, the corresponding projected feature maps must also be pruned. Since the identical feature maps are more important than the added residual maps, the feature map to be pruned should be determined by the pruning results of the shortcut layer. To determine which identity feature maps are to be pruned, we use the same selection criterion based on the filters of the shortcut convolutional layers (with 1×1 kernels). The second layer of the residual block is pruned with the same filter index as selected by the pruning of shortcut layer.

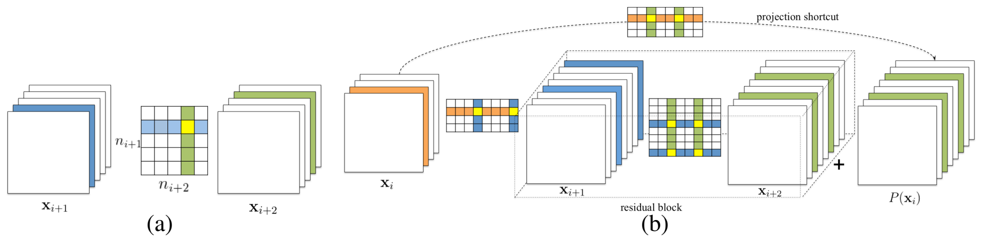


Figure 3: Pruning filters across consecutive layers. a) The greedy pruning strategy does not count kernels for the already pruned feature maps (shown in yellow). b) Pruning residual blocks with projection shortcut. The filters to be pruned for the second layer of the residual blocks (marked as green) is determined by the pruning result of the shortcut projection.

Retraining pruned networks to regain accuracy After pruning the filters, the performance degradation can be compensated by retraining the network. We find that for the layers that are resilient to pruning, the prune and retrain once strategy can prune away significant portions of the network and loss in accuracy can be regained by retraining for a short period of time (less than the original training times). However, when some sensitive layers are pruned away or a very large portions of the networks are pruned away, it is hard to recover the original accuracy.

3 Experiments

We prune two types of networks: simple CNNs (VGG-16 on CIFAR-10) and Residual networks (ResNet-56/110 on CIFAR-10 and ResNet-34 on ImageNet). We implement our filter pruning method in Torch. When filters are pruned, a new model with fewer filters is created and the remaining parameters of the modified layers as well as unaffected layers are copied into the new model. Furthermore, if a convolutional layer is pruned, the weights of the subsequent batch normalization layer are also removed. To get the baseline accuracies for each network, we train each model from scratch and follow the same pre-processing and hyper-parameters as ResNet [3]. For retraining, we use a constant learning rate 0.001 and retrain 40 epochs for CIFAR-10 and 20 epochs for ImageNet, which represents one-fourth of the original training epochs.

VGG-16 on CIFAR-10 VGG-16 [2] is a large capacity network originally designed for the ImageNet dataset. Recent work [13] has found that a slightly modified version produces state of the art results on the CIFAR-10 dataset. We use the model described in [13] but add Batch Normalization [14] after each convolutional layer and the first linear layer, without using Dropout [15]. Note that when the last convolutional layer is pruned, the input to the linear layer is changed and the connections are also removed. As seen in Figure 2(c), almost 90% of the filters of the convolutional layers with 512 feature maps can be safely removed after retraining. One explanation is that these filters operate on 4×4 or 2×2 feature maps, which may have no meaningful spatial connections in

Table 1: Overall results. The best test/validation accuracy during the retraining process is reported.

Model	Error(%)	FLOP	Pruned %	Parameters	Pruned %
VGG-16	6.75	3.13×10^8		1.5×10^7	
VGG-16-pruned-A	**6.60**	2.06×10^8	34.2%	5.4×10^6	64.0%
ResNet-56	6.96	1.25×10^8		8.5×10^5	
ResNet-56-pruned-A	6.90	1.12×10^8	10.4%	7.7×10^5	9.4%
ResNet-56-pruned-B	**6.94**	9.09×10^7	27.6%	7.3×10^5	13.7%
ResNet-110	6.47	2.53×10^8		1.72×10^6	
ResNet-110-pruned-A	**6.45**	2.13×10^8	15.9%	1.68×10^6	2.3%
ResNet-110-pruned-B	6.70	1.55×10^8	38.6%	1.16×10^6	32.4%
ResNet-34	26.77	3.64×10^9		2.16×10^7	
ResNet-34-pruned-A	27.44	3.08×10^9	15.5%	1.99×10^7	7.6%
ResNet-34-pruned-B	27.83	2.76×10^9	24.2%	1.93×10^7	10.8%
ResNet-34-pruned-C	27.52	3.37×10^9	7.5%	2.01×10^7	7.2%

such small dimensions. Unlike previous work [16, 8], we find that the first layer is quite robust to pruning as compared to the next few layers. With 50% of the filters being pruned in layer 1 and from 8 to 13, we achieve 34% FLOP reduction for the same original accuracy.

ResNet-56/110 on CIFAR-10 ResNets for CIFAR-10 have three stages of residual blocks for feature maps with the sizes of 32×32, 16×16 and 8×8. When the feature maps increase, the shortcut layer provides an identity mapping with an additional zero padding for the increased dimensions. Here, we only consider pruning the first layer of the residual block. We find that layers that are sensitive to pruning (layer 20, 38 and 54 for ResNet-56, layer 36, 38 and 74 for ResNet-110) lie at the residual blocks close to the layers where the number of feature maps change, e.g., the first and the last residual blocks for each stage. The retraining performance can be improved by just skipping pruning these sensitive layers. In addition, we find that the deeper layers are more sensitive to pruning than ones in the earlier stages of the network. Hence, we use a different pruning rate p_i for each stage. ResNet-56-pruned-A improves the performance by pruning 10% filters while skipping the sensitive layers 16, 20, 38 and 54. ResNet-56-pruned-B skips more layers (16, 18, 20, 34, 38, 54) and prunes layers with p_1=60%, p_2=30% and p_3=10%. ResNet-110-pruned-A gets a slightly better result with p_1=50% and layer 36 skipped. ResNet-110-pruned-B skips layer 36, 38, 74 with p_1=50%, p_2=40% and p_3=30. When there are more than two residual blocks at each stage, the middle residual blocks can be easily pruned. This might explain why ResNet-110 is easier to prune than ResNet-56.

ResNet-34 on ImageNet ResNets for ImageNet have four stages of residual blocks for feature maps of 56×56, 28×28, 14×14 and 7×7. ResNet-34 uses the projection shortcut when the feature maps are down-sampled. We first prune the first layer of each residual block. Similar to ResNet-56/110, the first and the last residual blocks of each stage are more sensitive to pruning than the intermediate blocks (i.e., layer 2, 8, 14, 16, 26, 28, 30, 32). We skip those layers and prune the remaining layers at each stage equally. We provide two options of pruning rates for the first three stages: (A) p_1=30%, p_2=30%, p_3=30%; (B) p_1=50%, p_2=60%, p_3=40%. Option-B provides 24% FLOP reduction with about 1% loss in accuracy. We also prune the identity feature maps and the second convolutional layer of the residual blocks. However, these layers are more sensitive to pruning Option-C prunes the third stage (p_3=20%) and only reduce 7.5% FLOP with 0.75% loss in accuracy. Therefore, pruning the first layer of the residual block is more effective at reducing the overall FLOP. This finding correlates with the bottleneck block design for deeper ResNets, which first reduce the dimension of input feature maps and then increase the dimension to match the identity mapping.

4 Conclusions

Modern CNNs are often over-capacity with large training and inference costs. We present a method to prune filters with relatively low weight magnitudes to reduce the inference costs without introducing irregular sparsity. Instead of layer-wise pruning and iterative retraining, we prune filters across multiple layers and retrain only once for simplicity and ease of implementation, which is critical for pruning very deep networks. We find that it can achieve about 30% reduction in FLOP for VGG-16 (on CIFAR-10) and deep ResNets without significant loss in the original accuracy.

References

[1] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet Classification with Deep Convolutional Neural Networks. In: NIPS. (2012)

[2] Simonyan, K., Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR (2015)

[3] He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. arXiv preprint arXiv:1512.03385 (2015)

[4] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the Inception Architecture for Computer Vision. arXiv preprint arXiv:1512.00567 (2015)

[5] Le Cun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: NIPS. (1989)

[6] Reed, R.: Pruning algorithms-a survey. IEEE transactions on Neural Networks **4**(5) (1993) 740–747

[7] Srinivas, S., Babu, R.V.: Data-free parameter pruning for deep neural networks. BMVC (2015)

[8] Han, S., Pool, J., Tran, J., Dally, W.: Learning both Weights and Connections for Efficient Neural Network. In: NIPS. (2015)

[9] Han, S., Mao, H., Dally, W.J.: Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. ICLR (2016)

[10] Mariet, Z., Sra, S.: Diversity Networks. In: ICLR. (2016)

[11] Polyak, A., Wolf, L.: Channel-Level Acceleration of Deep Face Representations. IEEE Access (2015)

[12] Anwar, S., Hwang, K., Sung, W.: Structured Pruning of Deep Convolutional Neural Networks. arXiv preprint arXiv:1512.08571 (2015)

[13] Zagoruyko, S.: 92.45% on CIFAR-10 in Torch. `http://torch.ch/blog/2015/07/30/cifar.html` (2015)

[14] Ioffe, S., Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv preprint arXiv:1502.03167 (2015)

[15] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. JMLR (2014)

[16] Zeiler, M.D., Fergus, R.: Visualizing and Understanding Convolutional Networks. In: ECCV. (2014)