From Rule-based to Learning-based Network Pruning

Ceyuan Yang, Yinghao Xu
The Chinese University of Hong Kong
{yc019,xy119}@ie.cuhk.edu.hk

Abstract

The powerful CNNs are known for their huge number of parameters, and tremendous computing resource consumptions, which leads to a rise in the requirements for smaller neural networks for on-the-go applications. We present a learning-based pruning method to simplify the network different from previous work which focuses on designing hand-crafted criteria for network slimming. In our paper, the pruning problem is regarding as a discrete search problem and we solve it by integrating Reinforcement Learning techniques. We train an agent to prune network by a data-driven way. We empirically demonstrate the effectiveness of our approach applied on several models, including VGG, and ResNet-18, on various image classification datasets. Code is available at https://github.com/justimyhxu/RLP.git.

1 Introduction

Deep neural networks (DNNs), especially deep convolution neural networks (DCNN) [5, 12, 9], have made remarkable strides during the last decade. These modern deep neural networks have millions of weights, rendering them both memory-intensive and computationally expensive. To reduce the computational cost, the research into network acceleration and compression emerges as an active field.

A family of popular compression methods are the DNN pruning algorithms, which are not only efficient in both memory and speed, but also enjoy the relatively simple procedure and intuition. This line of research is motivated by the theoretical analysis and empirical discovery that redundancy does exist in both human brains and several deep models [3]. We can categorize existing researches according to the level of the object, such as connection (weights)-level pruning, unit/channel/filter-level pruning, and layer-level pruning [14]. Connection-level pruning is the most widely studied approach, which produces sparse networks whose weights are stored as sparse tensors. Although both the footprint memory and the I/O consumption are reduced [4], such methods are often not helpful towards the goal of computation acceleration unless specifically-designed hardware is leveraged. This is because the dimensions of the weight tensor remain unchanged, though many entries are zeroed-out. As a well-known fact, the MAC operations on random structured sparse matrices are generally not too much faster than the dense ones of the same dimension. In contrast, structural pruning techniques [4], such as unit/channel/filter-level pruning, are more hardware-friendly, since they aim to produce tensors of reduced dimensions or having specific structures. Using these techniques, it is possible to achieve both computation acceleration and memory compression on general hardware and is common for deep learning frameworks.

However, previous works whether connection pruning or structure pruning, depend on a certain hand-crafted criterion or rule-based method. It is non-trivial to control the trade-off between network performance and scale during pruning. We formulate network pruning as a discrete search problem for the filter with good representation, which inspires us to adopt Reinforcement Learning [7] to solve it. Specifically, a pruning agent is designed to make a binary decision for every filter. We propose a

novel reward function to control the trade-off between the pruning ratio and network performance. Due to the non-differentiability of the reward function, we estimate the gradient by the policy gradient method [13] to obtain the expectation of gradient observation.

Compared with previous methods, we design a totally automatic pruning method by integrating RL to make a binary decision. Our method can also control the trade-off between network scale and performance, which cannot be easily handled by previous techniques.

1.1 Related Work

To address the issues in connection-level pruning, researchers proposed to increase the group-sparsity by applying sparse constraints to the channels, filters, and even layers [4, 1, 2]. [6] used LASSO constraints and reconstruction loss to guide network channel selection. [10] introduced L1-Norm rank to prune filters, which reduces redundancy and preserves the relatively important filters using a greedy policy. [11] leverages a scaling factor from batch normalization to pruning channels.

2 Methodology

Let $F^i \in R^{N_i \times H_i \times W_i}$ denote input feature maps of the i-th layer for the model f where N_i, H_i, and W_i are the number of the input feature maps, height, and width of the activation map, respectively. Kernel weights $W^i \in R^{N_{i+1} \times N_i \times K_{i+1} \times K_{i+1}}$ in this layer are convolved with the input feature map F^i into output feature map F^{i+1}. We decouple the weight matrix W^i represented by $\{w_1^i, w_2^i ... w_{N_i}^i\}$ at i-th layer. The goal of pruning is to remove certain filters in $\{w_1^i, w_2^i ... w_{N_i}^i\}$. If a filter is removed, the corresponding output feature map of this layer (which is also the input feature map of the next layer) will be removed, too.

2.1 Individual Pruning

We borrow the idea of RL to simplify the network. We train a agent π_i to take the filter $\{w_1^i, w_2^i ... w_{N_i}^i\}$ at i-th layer and output the binary decision action for each filter $A_i = \{a_1^i, a_2^i .. a_{N_i}^i\}$ to decide whether keep or remove the corresponding filter. Here $a_k^i \in \{0, 1\}$. Our goal is to maximize the reward $R(A_i)$ on the validation set. Firstly, to guarantee the performance of each action, we design an accuracy term $\phi(A_i, a, b)$ which formulated as the following:

$$\phi(A_i, a, b) = 1 - \frac{a - a^*}{b}, \tag{1}$$

where a and a^* are the performance of the pruned model and the original model on validation set and b is the performance drop bound which control the performance of pruned model fluctuate within a range as well as the trade-off between efficiency and accuracy.

To make the network to be slimming, we also design an efficiency term to encourage the agent pruning more filter. It can be formulated as:

$$\psi(A_i) = \frac{N_i}{C(A_i)}, \tag{2}$$

where $C(A_i)$ denotes the sum of the action set which meaning the number of preserved filter. A small $C(A_i)$ means that less filter are kept and the network is more efficient. It also contributes the ψA^i

By integrating $\phi(A_i, a, b)$ and $\psi(A_i)$, the fully objective reward function $R(A_i)$ is :

$$R(A_i) = \phi(A_i, a, b) * \psi(A_i). \tag{3}$$

A challenge that the reward function $R(A_i)$ is not differentiable is raised for training the agent π_i. Inspired by the policy gradient [13], we sample K times from the output distribution and obtain the expectation to estimate the gradient. The gradient can be formulated as:

$$\nabla_{\theta_i} \mathcal{L} = \mathbb{E}[R(\mathbf{A}_i) * \nabla_{\theta_i}(\pi_i(\mathbf{A}_i | W^i, \theta^i))]. \tag{4}$$

By designing the novel reward function and solve the non-differentiable objective function, the agent can be trained in an end2end manner and achieve the goal to control the network scale and its performance.

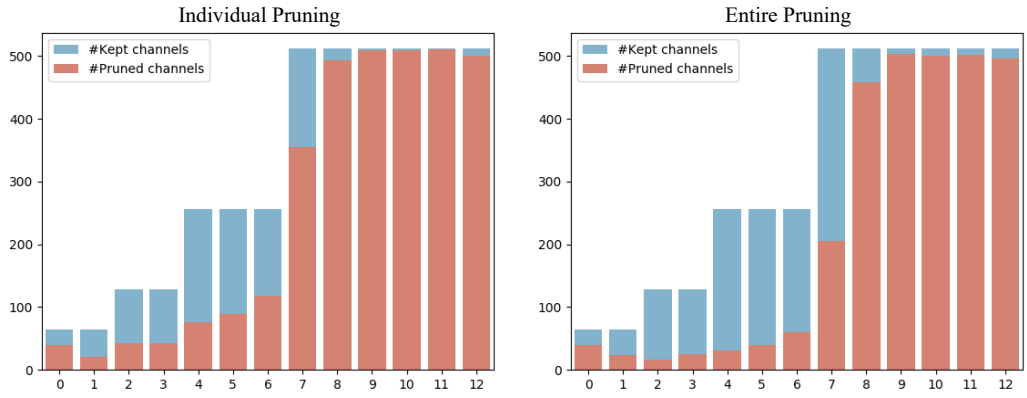

Figure 1: **Comparison between individual and entire pruning.**

2.2 Entire Pruning

Sec. 2.1 introduces our pruning algorithm for individual layer. It can easily generalize to the entire network pruning. We apply the individual pruning method from the lower layer and proceeds higher layer by layer. After finishing the pruning of one layer, it finetunes the entire network for some epochs to maintain the performance. Specifically, by training a set of agents $\{\pi_1, \pi_2..\pi_l\}$, out method can prune the entire network with a loop mode. The reason why we choose low to high training strategy is the pruning ratio than other strategies with the same drop bound.

3 Experiments

In order to evaluate our pruning strategy, we conduct quantitative experiments on visual classification task with various backbones. Besides, we also investigate the effect of different hyper-parameters (drop bound) to confirm our motivation that our method could control the trade-off between performance and network scale. The comparison between pruning for individual and entire layer could observe the reasonable results.

3.1 Main Results

We prune VGGNet[12] and ResNet-18[5] on CIFAR-10 dataset [8] respectively. Table 1 illustrates the performance of baseline and the pruned models. Compared with the original VGGNet [12] and ResNet-18 [5], our pruning method shows the consistent decrease of both the number of parameters and the computational complexity. Obviously, the larger drop bound tends to give the fewer parameters, fewer FLOPs (which is about computational complexity) and larger accuracy drop. Specifically, when the drop bound is set as 3, the number of parameters for VGGNet [12] and ResNet-18 [5] is decreased to 1.1MB and 2.4MB respectively. Importantly, Table 1 also demonstrates our motivation that we could control the trade-off between performances and network scale. Therefore, we could prune a given network as many parameters as possible while maintaining the performance at the desired level.

(a) **VGGNet.** Results of different drop bounds.

Models	#Params	GFLOPs	Top-1
Baseline	15M	3.1	92.7
Ours(b=0.5)	2.5M	1.7	92.2
Ours(b=1)	2.6M	1.4	91.4
Ours(b=2)	2.0M	1.1	90.5
Ours(b=3)	1.1M	0.6	89.2

(b) **ResNet-18.** Results of different drop bounds.

Models	#Params	GFLOPs	Top-1
Baseline	11M	1.3	93.5
Ours(b=0.5)	8.0M	1.0	93.5
Ours(b=1)	6.9M	0.8	92.2
Ours(b=2)	3.5M	0.5	91.6
Ours(b=3)	2.4M	0.3	90.2

Table 1: **Main results of pruning.** GFLOPs denotes the measurement of computational complexity.

3.2 Comparison Between Individual And Entire Pruning

Taking VGGNet [12] as an example, we also compare the individual (Sec.2.1) and entire (Sec.2.2) pruning. Figure 1 presents the results of two pruning methods. Specifically, the left figure is the results of individual pruning. The blue and orange part means the kept and pruned channels of each layer. Both of them suggest that higher layers tend to contain more unnecessary filters. Such conclusion is also pointed out in other work. Besides, given the same drop bound, the pruning ratio of entire pruning is higher than that of the individual. It is reasonable since the individual always prunes single layer.

4 Conclusion

In this work, a method of learning-based network pruning is proposed to decrease the redundant filters as many as possible while maintaining the performance of the pruned model at the desired level. Besides, due to the novel reward function, our pruning strategy allows to control the trade-off between the performance and network scale. Experimental results also demonstrate the effectiveness of our method.

References

[1] J. Alvarez and M. Salzmann. Learning the number of neurons in deep networks. *NIPS*, 2016.

[2] S. Anwar, K. Hwang, and W. Sung. Structured Pruning of Deep Convolutional Neural Networks. *arXiv Preprint*, 2015.

[3] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. d. Freitas. Predicting parameters in deep learning. *NIPS*, 2013.

[4] S. Han. Learning both weights and connections for efficient neural networks. *NIPS*, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[6] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017.

[7] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *JALR*, 1996.

[8] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[10] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning Filters for Efficient ConvNets. *ICLR*, 2016.

[11] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017.

[12] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[13] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 2000.

[14] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. *NIPS*, 2016.