The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Auto-Balanced Filter Pruning for Efficient Convolutional Neural Networks*

Xiaohan Ding,[1] Guiguang Ding,[1] Jungong Han,[2] Sheng Tang[3]
[1]School of Software, Tsinghua University, Beijing 100084, China
[2]School of Computing and Communications, Lancaster University, Lancaster, LA1 4YW, UK
[3]Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
dxh17@mails.tsinghua.edu.cn, dinggg@tsinghua.edu.cn, jungonghan77@gmail.com, ts@ict.ac.cn

Abstract

In recent years considerable research efforts have been devoted to compression techniques of convolutional neural networks (CNNs). Many works so far have focused on CNN connection pruning methods which produce sparse parameter tensors in convolutional or fully-connected layers. It has been demonstrated in several studies that even simple methods can effectively eliminate connections of a CNN. However, since these methods make parameter tensors just sparser but no smaller, the compression may not transfer directly to acceleration without support from specially designed hardware. In this paper, we propose an iterative approach named Auto-balanced Filter Pruning, where we pre-train the network in an innovative auto-balanced way to transfer the representational capacity of its convolutional layers to a fraction of the filters, prune the redundant ones, then re-train it to restore the accuracy. In this way, a smaller version of the original network is learned and the floating-point operations (FLOPs) are reduced. By applying this method on several common CNNs, we show that a large portion of the filters can be discarded without obvious accuracy drop, leading to significant reduction of computational burdens. Concretely, we reduce the inference cost of LeNet-5 on MNIST, VGG-16 and ResNet-56 on CIFAR-10 by 95.1%, 79.7% and 60.9%, respectively.

Introduction

The past few years have witnessed rapid developments of convolutional neural networks (CNNs) in the areas of computer vision, natural language processing, etc. However, due to their nature of computational intensity, as CNNs grow wider and deeper, their computational burdens have increased dramatically, making them difficult to deploy on embedded systems. Therefore, this research community is soliciting the solutions that are able to simplify the CNNs but without losing too much accuracy.

Recent researches on CNN compression methods have attracted much attention. Some excellent works, such as (Han et al. 2015) and (Guo, Yao, and Chen 2016), have explored connection-wise pruning techniques. However, these methods make the parameter tensors of convolutional or fully-connected layers just sparser, but no smaller, hence the compression may not transfer directly to acceleration without support from specially designed hardware (Molchanov et al. 2016) (Han et al. 2016).

Some other works focus on filter-wise pruning approaches and have made promising achievements, which can be divided into two categories. One category of works tends to impose sparse constraints on the model and produce a compact network by training, where the representatives include group sparsity regularizer (Lebedev and Lempitsky 2016), group Lasso regularization (Wen et al. 2016) (Alvarez and Salzmann 2016), tensor low rank constraints (Zhou, Alvarez, and Porikli 2016) and group sparse constraints (Zhou, Alvarez, and Porikli 2016). Another category of works iteratively discards unimportant filters from a well-trained network and usually requires re-training to avoid severe accuracy drop. (Abbasi-Asl and Yu 2017) measures a filter's importance by the classification accuracy reduction of the network after pruning it, and eliminates one single filter at each iteration based on this metric. (Molchanov et al. 2016) and (Li et al. 2016) prune filters in a similar iterative way but by different metrics. (Hu et al. 2016) evaluates a filter's importance by its output on a subset of the training data and prunes a fraction of the filters in several layers simultaneously at one single iteration.

In this paper, we aim to combine the advantages of the two paradigms. On the one hand, we are inspired by the idea that the network can be trained under a certain constraint to introduce structured sparsity in the parameters. On the other hand, a progressive approach consisting of alternate pruning and re-training is more suitable for pursuing the highest compression rate, since we can easily roll back to a previous state when the network is irreversibly damaged. Taking both points into account, we seek for an approach where we fine-tune a well-trained model to make it robust to pruning, then alternately prune and re-train it to produce a significantly smaller network. The rationale behind our idea is that we can certainly pre-train the network as a prevention of the structural damage at the very beginning, instead of making up for the performance reduction after pruning.

There is thereby an urgent need but it is still a significant challenge to find a powerful pre-training method before the

*This research was supported by the National Natural Science Foundation of China (Grant No. 61571269) and the Royal Society Newton Mobility Grant (IE150997). Corresponding author: Guiguang Ding.

first pruning which prepares the network for the upcoming pruning process as well as maintains the accuracy. As a candidate approach, reducing the absolute value of parameters is most natural: considering that the parameters are set to zero during pruning, the closer to zero they used to be, the less structural damage pruning does to the network (Han et al. 2015). Though (Han et al. 2015) uses ℓ_1 and ℓ_2 regularizations to zero out parameters in the connection-wise iterative pruning pipeline and gets encouraging results, it must be pointed out that filter-wise pruning is something different: it's hard to push all parameters in one filter close to zero simultaneously. If applying ℓ_1 or ℓ_2 regularization is expected to zero out a whole filter, the regularization factor must be much larger. Unfortunately, an overly strong regularization puts another problem on the table: when the ℓ_1 or ℓ_2 regularization term added to the objective function becomes larger, the accuracy loss is considered less important due to the nature of back-propagation algorithm, thus degrading the network performance. When we use usual ℓ_1 or ℓ_2 regularization in the proposed pipeline, we encounter a dilemma. If the regularization factor is small, filters cannot be effectively zeroed out, thus the pruning operation causes severe accuracy reduction. However, if the factor is large, the network's performance gets badly harmed during the pre-training process.

As our first contribution, we propose an approach called Auto-balanced Regularization to address this problem, where we apply ℓ_2 regularization with positive factors on unimportant filters and negative factors on important ones. The intuition is that we dynamically adjust the negative factors such that the sum of positive and negative regularization terms is always kept zero during training, meaning that as weak filters become weaker, the strong ones grow even stronger accordingly. Training with this regularization is able to transfer the model's representational capacity to a fraction of its filters, thus minimizing performance drop in the forthcoming pruning stage.

As our second contribution, we introduce a pre-training stage to common iterative pruning methods, as demonstrated in Figure 1. To be more specific, we propose a filter-wise pruning pipeline called Auto-balanced Filter Pruning (AFP), which removes redundant filters from a well-trained CNN to produce a smaller one, significantly reducing the computational cost. In this pipeline, the network is pre-trained at the very beginning, then iteratively pruned and re-trained to reach a satisfactory compression rate. In the pre-training process, we fine-tune the network with Auto-balanced Regularization to change the filter-wise distribution of parameters. In the pruning stage, the unimportant filters are discarded along with the corresponding feature maps. After pruning, the model is re-trained to restore its accuracy. Note that re-training is also done with our proposed regularization, hence each re-training stage can also be viewed as the pre-training stage of the next iteration. Besides, instead of pruning layer by layer, we prune filters from all convolutional layers simultaneously. To this end, we propose a novel iteration strategy named Abreast Advancing to decide which filters to prune at each iteration. By performing experiments on several common CNNs, we show that a large portion of the filters can be discarded without obvious accuracy drop, leading to significant reduction of computational burdens. Concretely, we reduce the inference cost of LeNet-5 on MNIST, VGG-16 and ResNet-56 on CIFAR-10 by 95.1%, 79.7% and 60.9%, respectively.

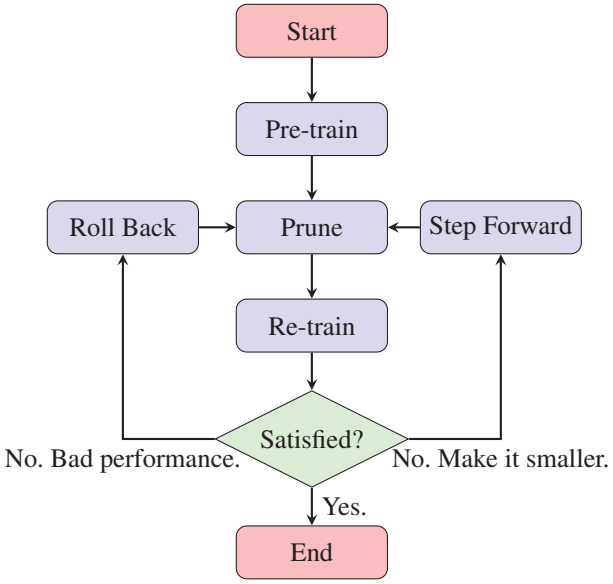


Figure 1: Auto-balanced Filter Pruning Pipeline

Related Work

Connection-wise CNN Pruning

A natural way to compress a CNN is removing some of its unimportant parameters. Nevertheless, defining what is important is a non-trivial work. Optimal Brain Damage (LeCun, Denker, and Solla 1990) and Optimal Brain Surgeon (Hassibi and Stork 1993) use second-order Taylor expansion to calculate the metric for importance of the parameters. However, these two methods require computation of the second derivatives, which is costly for today's deep CNNs. (Han et al. 2015) proposes a simple but effective approach where the importance of a parameter is measured by its absolute value. Based on that, the CNN is iteratively pruned and fine-tuned. In the pruning stage, a layer's unimportant parameters, i.e. the parameters with absolute values below a certain threshold, are set to zero, hence the network's accuracy drops due to the structural damage. Later in the re-training stage, the model recovers from the damage and restores its accuracy. After every iteration, the threshold for each layer is raised, thus producing increasingly sparser weight parameters. In the experiments, LeNet-5 (LeCun et al. 1998), AlexNet (Krizhevsky, Sutskever, and Hinton 2012) and VGG-16 (Simonyan and Zisserman 2014) are successfully compressed by an order of magnitude. However, the compression effects mainly come from fully-connected layers and the pruning of convolutional layers remains challenging.

Filter-wise CNN Pruning

In the past few years, several fully-convolutional neural networks (Long, Shelhamer, and Darrell 2015) (Redmon and Farhadi 2016) have made satisfactory achievements in the areas of semantic segmentation, object detection, etc. Besides, a substitute for fully-connected layers named global average pooling has been applied in some main-stream CNNs (Lin, Chen, and Yan 2013) (He et al. 2016). In this context, compression techniques for convolutional layers are attracting much attention.

Recently the application of sparsity constraints on CNN compression has been intensively investigated (Zhou, Alvarez, and Porikli 2016) (Wen et al. 2016) (Alvarez and Salzmann 2016) (Lebedev and Lempitsky 2016). These works use group sparsity based constraints to penalize unimportant parameters and learn a small compact network from a big redundant one.

Another way of filter pruning is iteratively deleting unimportant filters from a well-trained network. (Abbasi-Asl and Yu 2017), (Molchanov et al. 2016), (Guo, Yao, and Chen 2016), and (Li et al. 2016) have discussed different iteration strategies as well as various metrics to measure the filter importance. Apparently, the number of filters pruned at each iteration is a sort of trade-off to be solved: the fewer filters are discarded once, the less damage is done to the network structure, which means the less retraining time is required for the network to restore the accuracy; but more iterations are needed to reach a satisfactory compression rate.

Other Methods

Apart from pruning, there are some other CNN compression methods such as parameter quantization (Han, Mao, and Dally 2015), distillation (Hinton, Vinyals, and Dean 2015) and binarization (Rastegari et al. 2016) (Courbariaux et al. 2016), which are complementary to ours. It's important to point out that since our method simply learns a small and compact full-precision network with no sparsity or custom structure, it's promising to combine it with other methods and achieve even higher compression rate.

The Proposed Method

Filter-wise Pruning

In contrast to previous connection-wise pruning methods, we prune a CNN at the filter level. Every time a CNN is pruned, some 3-D filters along with the corresponding feature maps are deleted, resulting in a structural change in the network. It must be mentioned that when several convolutional layers stacked together are pruned, the FLOPs of the entire network are reduced quadratically. Imagine a simple CNN with n convolutional layers and m fully-connected layers on top, let u_i be the FLOPs of convolutional layer i and v_i be the FLOPs of fully-connected layer i, then the original inference cost can be represented as

$$C_o = \sum_{i=1}^{n} u_i + \sum_{i=1}^{m} v_i \,. \tag{1}$$

Let o_i be the original number of filters in the convolutional layer i and r_i be the number of remaining filters after pruning. Then the FLOPs of the pruned network is

$$C_p = \frac{r_1}{o_1} u_1 + \sum_{i=2}^{n} \frac{r_{i-1} r_i}{o_{i-1} o_i} u_i + \frac{r_n}{o_n} v_1 + \sum_{i=2}^{m} v_i \,. \tag{2}$$

Metric for Filter Importance

It's natural to measure a filter's importance by observing the performance drop of the network if it is deleted (Abbasi-Asl and Yu 2017). However, calculating the filter's importance in the way that remeasures the whole network's performance after discarding filters one by one is computationally impractical. Therefore, how to approximate a filter's importance without bringing in too much computational burdens becomes an inevitable topic. Recently many researchers have focused on this topic and proposed some enlightening metrics.

(Li et al. 2016) uses ℓ_1 norm to evaluate the importance of a filter for the ease of the calculation. This metric is naive but natural, for a parameter with a bigger absolute value is more likely to have a strong influence on the network's output. Inspired by the early works of (LeCun, Denker, and Solla 1990) (Hassibi and Stork 1993), some researchers choose to approximate a filter's importance in a Taylor expansion based way (Molchanov et al. 2016) (Wolfe et al. 2017). Instead of simply summing up filter parameters, an evaluation process is performed on a subset of the training set to calculate the metrics. Though costly, these methods usually outperform the naive ways.

In our experiments, we choose ℓ_1 norm as the filter importance metric to minimize the time cost as well as produce experimental results comparable to (Li et al. 2016). Formally, let $M_{i,j}$ be the importance metric of the j-th filter in the i-th convolutional layer, $F_{i,j}$ be the corresponding 3-D parameter tensor, and vec be the function that flattens a 3-D tensor to obtain a vector, we have

$$M_{i,j} = ||vec(F_{i,j})||_1 \,. \tag{3}$$

Auto-balanced Training

(Han et al. 2015) has demonstrated that ℓ_2 norm gives better experimental results than ℓ_1 norm as a means of regularization for iterative pruning. Considering that, we apply a customized ℓ_2 regularization to penalize weak filters and stimulate strong ones to grow stronger. Specifically, we set a hyper-parameter r_i for each layer i to represent the target number of remaining filters after pruning. For each layer i, we pick r_i filters with the biggest importance metric, i.e. ℓ_1 norm, to form a remaining set denoted as $\boldsymbol{R_i}$ and the others to make up a to-be-pruned set denoted as $\boldsymbol{P_i}$. We define a threshold value θ_i as the r_i-th largest importance metric among all the filters in layer i. Then we have

$$M_{i,j} < \theta_i \quad \forall F_{i,j} \in \boldsymbol{P_i} \,, \tag{4}$$

$$M_{i,j} \geq \theta_i \quad \forall F_{i,j} \in \boldsymbol{R_i} \,. \tag{5}$$

An ℓ_2 regularization term is added on every filter with different factors. The factors are positive for the filters in $\boldsymbol{P_i}$

and negative for those in $\boldsymbol{R_i}$. Furthermore, for the filters in $\boldsymbol{P_i}$, the weaker they are, the harder they are penalized. And for those in $\boldsymbol{R_i}$, the stronger they are, the more intensely they are stimulated. Since the magnitude of the parameters in different layers may vary considerably, a filter's importance is only comparable to the other ones in the same layer. In view of that, we define the factors based on θ_i as follows,

$$\lambda_{i,j} = \begin{cases} 1 + \log \dfrac{\theta_i}{M_{i,j} + \epsilon} & \text{if } F_{i,j} \in \boldsymbol{P_i}\,, \\ -1 - \log \dfrac{M_{i,j}}{\theta_i + \epsilon} & \text{if } F_{i,j} \in \boldsymbol{R_i}\,. \end{cases} \tag{6}$$

Note that these factors are calculated when a training stage starts and remain unchanged until the end of the training. This design brings about another benefit: when each retraining stage starts and $M_{i,j}$ for each filter j in convolutional layer i is updated, if one filter $F_{i,j}$ has been penalized at the last iteration but not yet pruned, $\lambda_{i,j}$ becomes larger, since θ_i is not likely to have significantly decreased. In this way, the penalization of $F_{i,j}$ becomes harder at the current iteration so that the training process is accelerated. We use logarithm here because the parameters may vary in magnitude but we don't want their regularization factors to differ that enormously. ϵ is set to 1×10^{-12} to avoid division by zero. Then we have the positive regularization term for $\boldsymbol{P_i}$ and the negative term for $\boldsymbol{R_i}$ denoted by $S(\boldsymbol{P_i})$ and $S(\boldsymbol{R_i})$,

$$S(\boldsymbol{P_i}) = \sum_{F_{i,j} \in \boldsymbol{P_i}} \lambda_{i,j} ||vec(F_{i,j})||_2^2\,, \tag{7}$$

$$S(\boldsymbol{R_i}) = \sum_{F_{i,j} \in \boldsymbol{R_i}} \lambda_{i,j} ||vec(F_{i,j})||_2^2\,. \tag{8}$$

We denote the union of all to-be-pruned sets by $\boldsymbol{P}$ and the union of all remaining sets by $\boldsymbol{R}$, thus we have,

$$S(\boldsymbol{P}) = \sum_{i=1}^{n} S(\boldsymbol{P_i})\,, \tag{9}$$

$$S(\boldsymbol{R}) = \sum_{i=1}^{n} S(\boldsymbol{R_i})\,. \tag{10}$$

The positive regularization terms of all convolutional layers are added to the network's original objective function by a factor α. Similarly, all negative terms are added by a factor τ. Let L_o denote the original objective function including common regularization terms, if any, then the new optimization objective of training is to minimize

$$L = L_o + \alpha S(\boldsymbol{P}) + \tau S(\boldsymbol{R})\,. \tag{11}$$

Therefore, the gradients of filters are changed as

$$\frac{\partial L}{\partial F_{i,j}} = \begin{cases} \dfrac{\partial L_o}{\partial F_{i,j}} + 2\alpha\lambda_{i,j}F_{i,j} & \text{if } F_{i,j} \in \boldsymbol{P_i}\,, \\ \dfrac{\partial L_o}{\partial F_{i,j}} + 2\tau\lambda_{i,j}F_{i,j} & \text{if } F_{i,j} \in \boldsymbol{R_i}\,. \end{cases} \tag{12}$$

Since $\lambda_{i,j} > 0\ \forall F_{i,j} \in \boldsymbol{P_i}$, $\lambda_{i,j} < 0\ \forall F_{i,j} \in \boldsymbol{R_i}$, the weak filters are penalized and the strong ones are stimulated, respectively. Notably, τ is not a hyper-parameter but calculated and reset before every batch of data is fed into the network during training (Eq. 13). Therefore, our method requires two hyper-parameters, namely the n-dimensional vector $\boldsymbol{r}$ which represents the target number of remaining filters in each layer, and the regularization factor α.

$$\tau = -\alpha \frac{S(\boldsymbol{P})}{S(\boldsymbol{R})} \tag{13}$$

The word auto-balanced includes two meanings. On the one hand, according to Eq. 13, the intensity of stimulation on strong filters varies with the weak ones. When the weak filters are zeroed out, the stimulation automatically stops and the training converges. On the other hand, as the weak filters in a certain layer are weakened and the strong ones are stimulated, the representational capacity of the weak part is gradually transferred to the strong part, keeping the whole layer's representational capacity unharmed.

Abreast Advancing Iterative Pruning

Algorithm 1 Abreast Advancing Iterative Pruning

Require: Original network N_o; Target number of remaining filters vector $\boldsymbol{r}$; Schedule vector $\boldsymbol{s}$
1: Initialize: Index $i = 1$; Current progress $p = 0$
2: Build network N by imposing Auto-balanced Regularization to N_o
3: Pre-train N
4: **repeat**
5: $p = s_i$
6: Calculate vector $\boldsymbol{z}$ according to $\boldsymbol{r}$ and p, which denotes the number of filters to be pruned in each layer at current iteration
7: For each layer i, remove z_i filters with the smallest M value
8: Re-construct Auto-balanced Regularization of N
9: Re-train N
10: $i = i + 1$
11: **until** $p == 1$
12: **return** N

For deep CNNs, such as VGG-16 (Simonyan and Zisserman 2014) and ResNet-56 (He et al. 2016), pre-training once is not enough for transferring the representational capacity of the weak filters to the strong ones entirely. Considering that, an iterative process of pruning and re-training is applied for these deep networks. In the pre-training stage, the network polarizes its filters and gets prepared for the pruning. In every pruning iteration, the network is pruned and re-trained. Concretely, in the pruning stage, some of the weak filters are discarded and the remaining ones are re-organized to make up a smaller network. In the following re-training stage, the new network is re-trained to restore its accuracy and further polarize the remaining filters. So essentially, every re-training stage can also be seen as the pre-training stage of the next iteration.

The iteration strategy that decides which filters to prune at each iteration plays an important role in the pipeline. In order to preserve the holistic structure of the original

model during training as well as reduce the time cost of the pipeline, we do not prune the network layer by layer, which is the most conservative and popular style. Instead, we propose a novel iteration strategy named Abreast Advancing, which means that the pruning tasks for all convolutional layers are carried out synchronously. Concretely, we pre-set the target number of pruned filters for each layer, and the ratios of the already pruned filters and the target numbers are kept the same among all convolutional layers at each iteration. For simplicity, we use a vector $\boldsymbol{s}$ named schedule vector to denote the pruning progress at each iteration. The elements in $\boldsymbol{s}$ must be strictly increasing and the last one must be 1. For example, $\boldsymbol{s} = (0.5, 0.75, 1)$ means that 50%, 75% and 100% of the target filters in every convolutional layer are synchronously removed at each iteration, respectively, thus the whole pruning process is finished within 3 iterations. We specify the algorithm in Algorithm 1.

Experiment

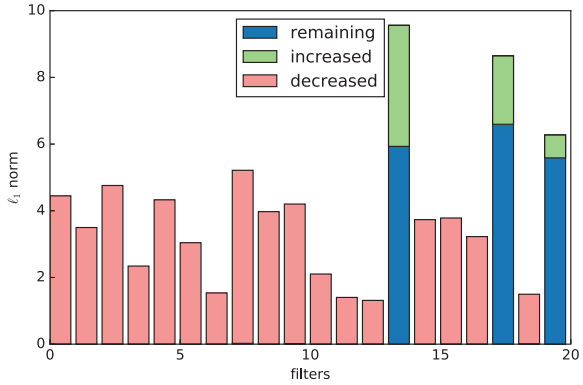
We evaluate the effectiveness of our proposed method (AFP) by pruning LeNet-5 (LeCun et al. 1998) on MNIST, VGG-16 (Simonyan and Zisserman 2014) and ResNet-56 (He et al. 2016) on CIFAR-10. We perform our experiments with tensorflow (Abadi et al. 2016) 1.01 on one NVIDIA TITAN X GPU. Every pruning experiment starts from a well-trained model and produces a smaller network with dramatically reduced FLOPs and comparative accuracy.

LeNet-5 on MNIST

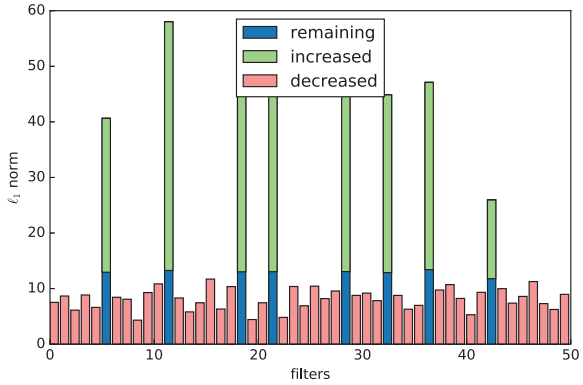
MNIST is a well-known database of handwritten digits containing 60,000 images for training and 10,000 for testing. We perform experiments on a version of LeNet-5 defined in (Yangqing 2014), which consists of two convolutional and two fully-connected layers. Since the two convolutional layers comprise 20 and 50 filters, respectively, we use a tuple (20, 50) to denote the network structure for simplicity. We train LeNet-5 from scratch on MNIST using the same training parameters as (Yangqing 2014). The trained network is tested on MNIST test set and achieves an error rate of 0.83%.

Since LeNet-5 is shallow, there is no need for pruning it in a progressive way. We simply pre-train the network with Auto-balanced Regularization and the filters are polarized perfectly. According to our most satisfactory result labeled as AFP-C, the error rate even further decreases to 0.76%. In the end, the weak filters are discarded and the error rate remains unchanged. Some experimental results of our method and another comparable work (Wen et al. 2016) are demonstrated in Table 1.

To demonstrate the effectiveness of Auto-balanced Regularization, we visualize the ℓ_1 norm of all the filters of the result AFP-C before and after pre-training in Figure 2. Obviously, all the filters except for the 3 and 8 strongest ones in two convolutional layers are zeroed out. It's noteworthy that, if pre-trained properly, the network even performs better with far fewer filters, which is consistent with the observation from (Han et al. 2015). A possible explanation is that eliminating parameters is an effective way of regularization which reduces overfitting of the network (Han et al. 2015).



(a) conv-1



(b) conv-2

Figure 2: The change of ℓ_1 norm of LeNet-5 filters

Table 1: LeNet-5 Experimental Results

Method	Filter	Error%	FLOPs	Pruned%
baseline	20,50	0.83	4.40×10^6	-
SSL-2	5,19	0.80	5.97×10^5	86.42%
SSL-3	3,12	1.00	2.89×10^5	93.42%
AFP-A	4,10	0.74	3.07×10^5	93.00%
AFP-B	4,5	0.96	1.95×10^5	95.56%
AFP-C	3,8	0.76	2.14×10^5	95.13%
AFP-D	3,5	2.21	1.58×10^5	96.41%

VGG-16 on CIFAR-10

CIFAR-10 is a widely used database comprising 50,000 RBG images of 32×32 pixels for training and 10,000 for testing. In our experiments, we use random horizontal flip and random shift for data augmentation.

The experiments in this section are based on a VGG-16 version same as (Li et al. 2016). Concretely, the overall convolutional architecture remains the same as (Simonyan and Zisserman 2014), while every convolutional layer is followed by a batch normalization (Ioffe and Szegedy 2015)

layer. Besides, the fully-connected part is re-designed: only one hidden layer with 512 neurons is applied without dropout (Srivastava et al. 2014). The network is trained from scratch for 160 epochs and eventually achieves 7.08% error rate.

We set the hyper-parameter $\boldsymbol{r}$ based on the layer-wise histogram of ℓ_1 norm of filters in the original network. Basically, the steeper the histogram looks, the more filters we prune from the layer. Note that if we define this hyper-parameter too conservatively and result in an unsatisfactory compression effect, we can adjust $\boldsymbol{r}$ and restart our pipeline with the previous output as initialization, thus no effort is wasted. Based on the histograms of the 13 layers, we prune 50% of the filters from layer 4-8 and 90% of the filters from layer 9-13. We only shift r_1, r_2 and r_3, namely $\boldsymbol{r} = (r_1, r_2, r_3, 64, 128, 128, 128, 256, 52, 52, 52, 52, 52)$. Considering that the original ℓ_2 regularization factor of the network is 5×10^{-4}, we set α to be an order larger i.e. 5×10^{-3}. Since VGG-16 is deep, an Abreast Advancing pipeline is applied with $\boldsymbol{s} = (0.5, 0.9, 1)$. We train the network with a learning rate of 3×10^{-4} and 3×10^{-5} for 50 epochs respectively in each training stage. We first perform experiments with no Auto-balanced Regularization but only Abreast Advancing (AA) iterative pruning and then apply the complete version of AFP. It can be seen from Table 2 that our method is able to reduce the FLOPs of VGG-16 on CIFAR-10 by 79.69% with acceptable accuracy drop using only Abreast Advancing. Better still, when AFP is applied, no accuracy loss is observed (AFP-E).

Table 2: VGG-16 Experimental Results

Method	r_1, r_2, r_3	Error	FLOPs	Pruned%
Li-base	-	6.75	3.13×10^8	-
Li-pruned	-	6.60	2.06×10^8	34.2%
AFP-base	-	7.08	3.13×10^8	-
AA-E	39,39,77	7.28	6.37×10^7	79.69%
AFP-E	39,39,77	7.06	6.37×10^7	79.69%
AA-F	64,26,52	7.56	5.83×10^7	81.39%
AFP-F	64,26,52	7.13	5.83×10^7	81.39%

ResNet-56 on CIFAR-10

We use the same ResNet-56 architecture as described in (He et al. 2016), which contains 3 stages of convolutional layers connected by projection mapping and followed by a global average pooling layer and one fully-connected layer. After trained on CIFAR-10 from scratch using the same training parameters as (He et al. 2016), the network achieves an error rate of 6.07%.

For simplicity and consistency, we number the convolutional layers of ResNet-56 as follows: the first layer is numbered 1; from the shallower to the deeper in one stage, the number increases; when it comes to the junction of two adjacent stages, we number the projection layer before the two convolutional layers in the parallel residual block.

Due to the special structure of identity mapping, the pruned indexes of the input and output feature maps of one residual block must be identical (Li et al. 2016), or the residual mechanism goes wrong. With D_i denoting the set of discarded filter indexes of layer i, we have $D_1 = D_3 = D_5 = \ldots = D_{19}$, $D_{20} = D_{22} = D_{24} = \ldots = D_{38}$,$D_{39} = D_{41} = D_{43} = \ldots = D_{57}$ as constraints. Since we cannot expect that the important filters in the same stage are in the same positions, pruning this network is a real challenge,

Table 3: ResNet-56

Method	q	Error	FLOPs	Pruned%
Li-base	-	6.96	1.25×10^8	-
Li-A	-	6.90	1.12×10^8	10.4%
Li-B	-	6.94	9.04×10^7	27.6%
AFP-base	-	6.07	1.42×10^8	-
AFP-G	10,20,40	7.06	5.55×10^7	60.86%
AFP-H	12,20,32	7.41	5.72×10^7	59.65%
AFP-I	12,16,16	9.43	4.14×10^7	70.79%

(Li et al. 2016) performs experiments on ResNet-56 in a conservative way. To avoid changing the input and output feature maps of each residual block, (Li et al. 2016) only prunes filters from the first layers of each block. Namely, only filters from layer 2,4,6,...,18 in stage 1, layer 21,23,25,...,37 in stage 2 and layer 40,42,44,...,56 in stage 3 are pruned.

We perform our experiments far more aggressively by satisfying

$$\begin{aligned} D_i &= D_1 \quad \forall 1 < i \leq 19\,, \\ D_i &= D_{20} \quad \forall 20 < i \leq 38\,, \\ D_i &= D_{39} \quad \forall 39 < i \leq 57\,, \end{aligned}$$

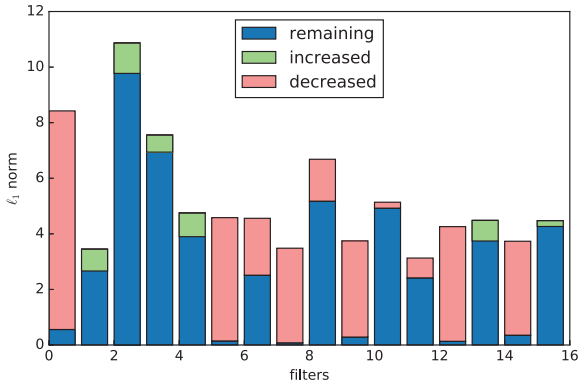
which means all convolutional layers in the same stage (if we group projection 1 into stage 2 and projection 2 into stage 3) are pruned following the same pattern. To this end, we pick 3 layers from the 3 stages respectively as the pacesetters and force every other convolutional layer to transfer its representational capacity exactly the same way as its corresponding pacesetter does. We use $M_i \coloneqq M_j$ as a simple notation for assigning $M_{i,k} \coloneqq M_{j,k} \quad \forall k \leq h$ where layer i and j both have h filters. With this notation, we assign $M_a \coloneqq M_b$ to force layer a to adjust its filters just like layer b does. In our experiments, we pick the first layer of stage 1, the second layer of stage 2 and the second layer of stage 3 as pacesetters. Note that since projection layers have 2×2 kernels instead of 3×3, we pick the first convolutional layer in the first residual block of stage 2 but not the projection layer as the pacesetter simply because it is more similar to other convolutional layers, and the same is true for stage 3. Formally, that is

$$\begin{aligned} M_i &\coloneqq M_1 \quad \forall 1 \leq i \leq 19\,, \\ M_i &\coloneqq M_{21} \quad \forall 20 \leq i \leq 38\,, \\ M_i &\coloneqq M_{40} \quad \forall 39 \leq i \leq 57\,. \end{aligned}$$

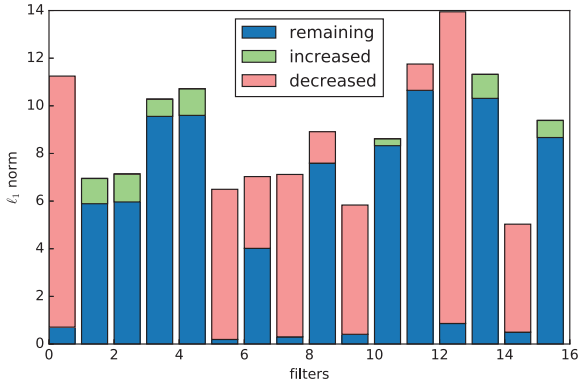
In practice, we simply calculate the ℓ_1 norms of layer 1, 21 and 40 as the importance metrics and apply them to calculate the λ values of all filters. Since we don't expect the

importance of all the filters in one stage to be of identical distribution, some important filters may be penalized and unimportant ones may be stimulated. Aggressive as it is, our method performs well. We use a triple q to denote the number of remaining filters in each convolutional layer within the 3 stages. Our results are demonstrated in Table 3 together with a comparable work (Li et al. 2016). We also visualize the ℓ_1 norm of filters in layer 4 and layer 14 after pre-training but before pruning. We choose these two layers simply because of their distinctly different distribution of parameters. It is observed that even if important filters are penalized and zeroed out to follow the pattern of the pacesetter i.e. layer 1, the network performs well.

Though we cannot learn a smaller network with an accuracy higher than our baseline, the experiment is still a success considering that ResNet-56 is an originally compact network with complex structure. Concretely, our method is able to reduce the FLOPs of ResNet-56 by 60.86% with an acceptable accuracy decrease of 0.99%. This experiment demonstrates that our method is able to not only utilize and amplify the inherent difference between filters, but also reshape the network drastically as we wish.



(a) conv-4



(b) conv-14

Figure 3: The change of ℓ_1 norm of ResNet-56 filters

Conclusion

To prune CNNs at the filter level, we propose a method named Auto-balanced Filter Pruning (AFP), which learns a small compact network from a big redundant one, hence significantly reduces the FLOPs of the network. We propose Auto-balanced Regularization, which penalize the weak filters as well as stimulate the strong ones, transferring the representational capacity of a whole convolutional layer to a fraction of its filters. Furthermore, we propose a pruning pipeline, which contains one pre-training stage before the iterative pruning and re-training process. In the pre-training stage, we train the network with Auto-balanced Regularization and polarize its filters. In the pruning stage, some of the weak filters are discarded along with the corresponding feature maps. In the re-training stage, the network is trained to restore its accuracy and prepare for the next pruning. Besides, instead of pruning layer by layer, we propose a novel iteration strategy named Abreast Advancing, which is aimed to preserve the original holistic network structure as well as reduce the time cost of the pipeline. By applying this method on a few common CNNs, we reduce the inference cost of LeNet-5 on MNIST, VGG-16 and ResNet-56 on CIFAR-10 by 95.1%, 79.7% and 60.9%, respectively.

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Abbasi-Asl, R., and Yu, B. 2017. Structural compression of convolutional neural networks based on greedy filter pruning. *arXiv preprint arXiv:1705.07356*.

Alvarez, J. M., and Salzmann, M. 2016. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, 2270–2278.

Courbariaux, M.; Hubara, I.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*.

Guo, Y.; Yao, A.; and Chen, Y. 2016. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, 1379–1387.

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, 1135–1143.

Han, S.; Liu, X.; Mao, H.; Pu, J.; Pedram, A.; Horowitz, M. A.; and Dally, W. J. 2016. Eie: efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture*, 243–254. IEEE Press.

Han, S.; Mao, H.; and Dally, W. J. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Hassibi, B., and Stork, D. G. 1993. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, 164–171.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hu, H.; Peng, R.; Tai, Y.-W.; and Tang, C.-K. 2016. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*.

Ioffe, S., and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 448–456.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Lebedev, V., and Lempitsky, V. 2016. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2554–2564.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11):2278–2324.

LeCun, Y.; Denker, J. S.; and Solla, S. A. 1990. Optimal brain damage. In Touretzky, D. S., ed., *Advances in Neural Information Processing Systems 2*. Morgan-Kaufmann. 598–605.

Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.

Lin, M.; Chen, Q.; and Yan, S. 2013. Network in network. *arXiv preprint arXiv:1312.4400*.

Long, J.; Shelhamer, E.; and Darrell, T. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3431–3440.

Molchanov, P.; Tyree, S.; Karras, T.; Aila, T.; and Kautz, J. 2016. Pruning convolutional neural networks for resource efficient inference.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, 525–542. Springer.

Redmon, J., and Farhadi, A. 2016. Yolo9000: better, faster, stronger. *arXiv preprint arXiv:1612.08242*.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Srivastava, N.; Hinton, G. E.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research* 15(1):1929–1958.

Wen, W.; Wu, C.; Wang, Y.; Chen, Y.; and Li, H. 2016. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, 2074–2082.

Wolfe, N.; Sharma, A.; Drude, L.; and Raj, B. 2017. The incredible shrinking neural network: New perspectives on learning representations through the lens of pruning. *arXiv preprint arXiv:1701.04465*.

Yangqing, J. 2014. Caffe LeNet-5. https://github.com/BVLC/caffe/tree/master/examples/mnist/.

Zhou, H.; Alvarez, J. M.; and Porikli, F. 2016. Less is more: Towards compact cnns. In *European Conference on Computer Vision*, 662–677. Springer.