Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Accelerating Convolutional Networks via Global & Dynamic Filter Pruning

Shaohui Lin[1,2], Rongrong Ji[1,2*], Yuchao Li[1,2], Yongjian Wu[3], Feiyue Huang[3], Baochang Zhang[4]
[1]Fujian Key Laboratory of Sensing and Computing for Smart City, Xiamen University, China
[2]School of Information Science and Engineering, Xiamen University, China
[3]BestImage, Tencent Technology (Shanghai) Co.,Ltd, China
[4]School of Automation Science and Electrical Engineering, Beihang University, China
Contact author: rrji@xmu.edu.cn

Abstract

Accelerating convolutional neural networks has recently received ever-increasing research focus. Among various approaches proposed in the literature, filter pruning has been regarded as a promising solution, which is due to its advantage in significant speedup and memory reduction of both network model and intermediate feature maps. To this end, most approaches tend to prune filters in a layer-wise fixed manner, which is incapable to dynamically recover the previously removed filter, as well as jointly optimize the pruned network across layers. In this paper, we propose a novel global & dynamic pruning (GDP) scheme to prune redundant filters for CNN acceleration. In particular, GDP first **globally** prunes the unsalient filters across all layers by proposing a global discriminative function based on prior knowledge of each filter. Second, it **dynamically** updates the filter saliency all over the pruned sparse network, and then recovers the mistakenly pruned filter, followed by a retraining phase to improve the model accuracy. Specially, we effectively solve the corresponding non-convex optimization problem of the proposed GDP via stochastic gradient descent with greedy alternative updating. Extensive experiments show that the proposed approach achieves superior performance to accelerate several cutting-edge CNNs on the ILSVRC 2012 benchmark, comparing to the state-of-the-art filter pruning methods.

1 Introduction

Convolutional neural networks (CNNs) have achieved remarkable success in various applications such as image classification [He *et al.*, 2016; Krizhevsky *et al.*, 2012; Simonyan and Zisserman, 2014], object detection [Girshick *et al.*, 2014] and semantic segmentation [Long *et al.*, 2015]. However, the promising performance is accompanied by significant computation cost, which raises huge difficulty to deploy these CNNs in real-time applications without the support of highly-efficient Graphic Processing Units (GPUs). As a result, the acceleration of convolutional networks has become emerging.

[*]Corresponding author

Recent works in convolutional neural network acceleration can be categorized into three groups, *i.e.*, low-rank decomposition, parameter quantization, and network pruning. Among them, network pruning has received ever-increasing research focus, which merits in limited memory footprints due to the small amount of filter parameters and intermediate activation, which is highly required for memory-light online inference. Methods in network pruning can be further categorized into either non-structured or structured pruning. Non-structured pruning [LeCun *et al.*, 1989; Hassibi and Stork, 1993; Han *et al.*, 2015a; 2015b] targets at directly pruning parameters independently in each layer, which will cause irregular memory access that adversely impacts the efficiency of online inference. Under such a circumstance, specialized hardware [Han *et al.*, 2016] or software [Liu *et al.*, 2015; Park *et al.*, 2017] designs are required to further speedup the pruned unstructured CNNs. Instead, structured pruning [Anwar *et al.*, 2015; Lebedev and Lempitsky, 2016; Wen *et al.*, 2016; Li *et al.*, 2016; Luo *et al.*, 2017; Molchanov *et al.*, 2017; Hu *et al.*, 2016] aims at directly removing filters as a whole, which is far more efficient and independent to specialized hardware/software platforms. For instance, Anwar *et al.* [Anwar *et al.*, 2015] introduced the structured sparsity to either filter-wise or channel-wise convolutional filter selection, based on which pruned filters with regularity by using particle filtering. Luo *et al.* [Luo *et al.*, 2017] implicitly associated the convolutional filter in the current layer with the input channel in the next layer, based on which pruned filters in the current layer via input channel selection of the next layer.

However, the existing structured pruning schemes prune the convolutional neural network in a layer-by-layer fixed manner, which is less adaptive, less efficient, and less effective. First, in local pruning, iterative layer-wise pruning and local fine-tuning are required, which is computational intensive. Second, mistaken pruning of the salient filter is irretrievable, which is inadaptive and the pruned network cannot achieve an optimal performance.

In this paper, we present a novel global & dynamic pruning (GDP) scheme to prune redundant filters to address above

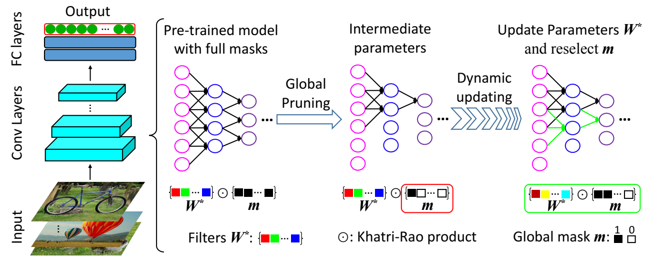


Figure 1: An illustration of GDP. Each rectangle with color (*e.g.*, red rectangle) is a filter in the filter set $\mathbf{W}^*$, while a global mask $\mathbf{m}$ with binary values determines the saliency of filters (*i.e.*, ■ indicates the corresponding filter is salient, and □ is unsalient). First, a pre-trained model and a full global mask are employed to initialize the network. Then, the redundant filters are globally pruned across all layers by masking out the corresponding value as 0. Finally, iteratively dynamic updating of filters and global mask is done to improve the accuracy of the pruned network. (Best viewed in color.)

two issues, which can largely accelerate the pruned networks while reducing the networks accuracy loss. Unlike the previous schemes of layer-by-layer fixed filter pruning, our key innovation lies in evaluating the importance/saliency of individual filter *globally* across all network layers, upon which *dynamically* and iteratively prune and tune the network, with the mechanism to recall filters that are mistakenly pruned in the previous iterations. Figure 1 demonstrates the flowchart of the proposed framework. In particular, we first initialize a pre-trained convolutional network and globally mask all filters to be equal to 1 (*i.e.*, an external switch which determines whether the filter is pruned). Then, we design a global discriminative function to determine the saliency scores of individual filters. Such scores guide us to globally prune the unsalient filters across all layers, which equivalently masks out unsalient filters as 0. Finally, we iteratively tune the sparse network and dynamically update the filter saliency in a top-down manner. By such operations, filters that are previously masked out is possible to recalled, which significantly improves the accuracy of the pruned network. In terms of optimization, GDP can be described as a non-convex optimization problem, which is then effectively solved via the stochastic gradient descent with greedy alternative updating.

The proposed GDP is evaluated on the ImageNet 2012 dataset [Russakovsky *et al.*, 2015] and implemented on the widely-used AlexNet [Krizhevsky *et al.*, 2012], VGG-16 [Simonyan and Zisserman, 2014] and ResNet-50 [He *et al.*, 2016]. Comparing to the state-of-the-art filter pruning methods [Wen *et al.*, 2016; Li *et al.*, 2016; Luo *et al.*, 2017; Molchanov *et al.*, 2017; Hu *et al.*, 2016], the proposed GDP scheme achieves the superior performance by a factor of $2.12\times$ GPU speedup with 1.15% Top-5 accuracy loss on AlexNet, $2.17\times$ CPU speedup with 1.45% Top-5 accuracy loss on VGG-16, and $1.93\times$ CPU speedup with 2.16% Top-5 accuracy loss on ResNet-50.

2 Related Work

LeCun *et al.* [LeCun *et al.*, 1989] and Hassibi *et al.* [Hassibi and Stork, 1993] proposed a saliency measurement to remove unimportant weights, which is determined by the second-order derivative matrix of the loss function with respect to the parameters. Recently, Han *et al.* [Han *et al.*, 2015a; 2015b] proposed to prune parameters by using iterative thresholding to remove unsalient weights with small absolute values. Guo *et al.* [Guo *et al.*, 2016] pruned weights by using connection splicing to avoid incorrect pruning. However, such scheme can only be worked in a local manner layer-by-layer. Different from connection splicing, the proposed dynamic updating is conducted in a global manner, which can restore important filters that were mistakenly removed across all layers.

In line with our work, a few methods have been proposed for filter-level/channel pruning (*i.e.*, structured pruning), which can reduce both network size and inference speed. Li *et al.* [Li *et al.*, 2016] measured the importance of each filter by calculating the ℓ_1-norm to prune unsalient filters together with their corresponding feature maps. Hu *et al.* [Hu *et al.*, 2016] computed the Average Percentage of Zeros (APoZ) of each filter, *i.e.*, the percentage of zero values in the output feature map associated with the corresponding filter, which serves as its score to guide pruning. Lebedev *et al.* [Lebedev and Lempitsky, 2016] and Wen *et al.* [Wen *et al.*, 2016] utilized group sparsity regularization to prune convolutional filters in a group-wise fashion during the training, which is however less efficient since only stochastic gradient descend is used. Recently, a new criterion based on Taylor expansion has been introduced in [Molchanov *et al.*, 2017] to globally prune one filter and then fine-tune the pruned network. However, it was prohibitively costly when applying to deep networks, as time-consuming fine-tuning has to be done after pruning each filter. Our method is different to all above methods, in terms of globally removing unsalient filters across all layers, as well as dynamically restoring salient filters that were previously mislabeled removed.

Orthogonal methods to our work include low-rank decomposition [Denton *et al.*, 2014; Jaderberg *et al.*, 2014; Lin *et al.*, 2016; 2017; Lebedev *et al.*, 2014; Kim *et al.*, 2015], parameter quantization [Gong *et al.*, 2014; Wu *et al.*, 2016; Courbariaux *et al.*, 2015; Courbariaux and Bengio, 2016; Rastegari *et al.*, 2016], which have also widely used to accelerate convolutional networks. Low-rank decomposition [Denton *et al.*, 2014; Jaderberg *et al.*, 2014; Lin *et al.*, 2017; Lebedev *et al.*, 2014; Kim *et al.*, 2015] typically decomposed convolutional filters into a sequence of tensor based convolutions with fewer parameters. For parameter quantization, Gong *et al.* [Gong *et al.*, 2014] and Wu *et al.* [Wu *et al.*, 2016] employed product quantization over parameters to reduce the redundancy in the parameter space. Recently, directly predicting the model with binary weights has been proposed in [Courbariaux *et al.*, 2015; Courbariaux and Bengio, 2016; Rastegari *et al.*, 2016]. It is worth to note that, our scheme can be integrated with the above orthogonal methods to further accelerate the pruned network.

3 Globally Dynamic Pruning

3.1 Notations

CNN can be viewed as a feed-forward multi-layer architecture that maps the input image to a certain output vector. In

CNN, the convolutional layers are most time-consuming. Let us denote a set of image feature maps in the l-th layer by $\mathcal{Z}_l \in \mathbb{R}^{H_l \times W_l \times C_l}$ with size $H_l \times W_l$ and individual maps (or channels) C_l. The feature maps can either be the input of the network $\mathcal{Z}_0$, or the output feature maps $\mathcal{Z}_l$ with $l \in [1, 2, \cdots, L]$. In addition, we denote individual feature map by $\mathbf{Z}_l^{(k)} \in \mathbb{R}^{H_l \times W_l}$ with $k \in [1, 2, \cdots, C_l]$. The individual output feature map of the l-th convolutional layer $\mathbf{Z}_l^{(k)}$ is obtained by applying the convolutional operator $(*)$ to a set of input feature maps with filters parameterized by $\mathcal{W}_l^{(k)} \in \mathbb{R}^{d \times d \times C_{l-1}}$[1], *i.e.*,

$$\mathbf{Z}_l^{(k)} = f(\mathcal{Z}_{l-1} * \mathcal{W}_l^{(k)}), \tag{1}$$

where $f(\cdot)$ is a non-linear activation function, *e.g.*, rectifier linear unit (ReLU).

In many deep learning frameworks like Caffe [Jia *et al.*, 2014] and Tensorflow [Abadi *et al.*, 2016], the tensor-based convolution operator is reformulated as a matrix-by-matrix multiplication by lowering the input and reshaping the filters, such as:

$$\mathbf{Z}_l^* = f(\mathbf{Z}_{l-1}^* \times \mathbf{W}_l^*), \tag{2}$$

where each row of the matrix $\mathbf{Z}_{l-1}^* \in \mathbb{R}^{H_l W_l \times d^2 C_{l-1}}$ is related to the spatial position of the output tensor transformed from the input tensor $\mathcal{Z}_{l-1}$, and the matrix $\mathbf{W}_l^* \in \mathbb{R}^{d^2 C_{l-1} \times C_l}$ is reshaped from filter $\mathcal{W}_l$[2].

3.2 The Proposed Pruning Scheme

Our goal is to globally prune redundant filters. To that effect, a large network can be directly converted into a compact one without repeatedly evaluating each filter saliency and fine-tuning the pruned network layer-by-layer. We introduce a global mask to temporally mask out unsalient filters in each iteration during training. Therefore, Eq. 2 can be rewritten as:

$$\mathbf{Z}_l^* = f\Big(\mathbf{Z}_{l-1}^* \times (\mathbf{W}_l^* \odot \mathbf{m}_l)\Big), \quad s.t.\ l = 1, 2, \cdots, L, \tag{3}$$

where $\mathbf{m}_l = \{0,1\}^{C_l}$ is a mask with binary values. $m_l^k = 1$ if the k-th filter is salient in the l-th layer, and 0 otherwise. $\odot$ denotes the Khatri-Rao product operator.

As we argued, pruning the filters in an irretrievable/fixed way is inflexible and ineffective in practice, which will cause severe performance loss. Note that the filter saliency may change dramatically after pruning a certain layer, as there exists complex interconnections among filters [Guo *et al.*, 2016]. Therefore, dynamic pruning, *i.e.*, enabling the roll-back of masked filters in a global perspective, is highly desired to improve the discriminability of the pruned network.

To better describe the objective function for the proposed GDP, we denote filters of the entire network as $\mathcal{W}^* = \{\mathbf{W}_1^{1*}, \mathbf{W}_1^{2*}, \cdots, \mathbf{W}_L^{C_L*}\}$ and a global mask as $\mathbf{m} = \{0,1\}^{\sum_{l=1}^L C_l}$. We further give a set of training examples $\mathcal{D} = \Big\{\mathcal{X} = \{\mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_N\}, \mathcal{Y} = \{\mathbf{Y}_1, \mathbf{Y}_2, \cdots, \mathbf{Y}_N\}\Big\}$, where $\mathbf{X}_i$ and $\mathbf{Y}_i$ represent an input and a target output, respectively. We propose to solve the following optimization problem:

$$\begin{aligned} \min \quad & \mathcal{L}\Big(\mathcal{Y}, g\big(\mathcal{X}; \mathcal{W}^*, \mathbf{m}\big)\Big) \\ s.t. \quad & \mathbf{m} = h(\mathcal{W}^*) \\ & \|\mathbf{m}\|_0 \le \beta \textstyle\sum_{l=1}^L C_l, \end{aligned} \tag{4}$$

where $\mathcal{L}(\cdot)$ is a loss function for the pruned network, *e.g.*, cross-entropy loss. $g\big(\mathcal{X}; \mathcal{W}^*, \mathbf{m}\big)$ takes the input $\mathcal{X}$, the filters $\mathcal{W}^*$ and the global mask $\mathbf{m}$ to map to an s-dimensional output (s is the number of classes). $h(\cdot)$ is a global discriminative function to determine the saliency values of filters, which depends on the prior knowledge of $\mathcal{W}^*$. The output entry of function $h(\cdot)$ is binary, *i.e.*, to be 1 if the corresponding filter is salient, and 0 otherwise.

Eq. 4 is the core function in our GDP framework, which is non-convex and whose solver will be introduced in Sec. 3.3. $\beta \in (0, 1]$ is a threshold to determine the sparsity of the pruned network. The problem Eq. 4 is NP-hard, because of the $\|\cdot\|_0$ operator. We simplify this NP-hard problem by bounding $\mathbf{m}$ on prior knowledge of $\mathcal{W}^*$. Then, it is solved by greedy and alternatively updating $\mathcal{W}^*$ and $\mathbf{m}$ by using the stochastic gradient descent, which will be introduced in detail in Sec. 3.3.

3.3 The Solver

We first investigate the constraint in Eq. 4, which can be relaxed by greedily selecting an amount of $\beta \sum_{l=1}^L C_l$ most important filters, which determines the global discriminative function h based on the prior knowledge of $\mathcal{W}^*$. Then, we only need to solve the objective function in Eq. 4 through the stochastic gradient descent. Since every filter has a mask, we update $\mathcal{W}^*$ as below:

$$\mathbf{W}_l^* = \mathbf{W}_l^* - \eta \frac{\partial \mathcal{L}\big(\mathcal{Y}, g(\mathcal{X}; \mathcal{W}^*, \mathbf{m})\big)}{\partial(\mathbf{W}_l^* \odot \mathbf{m}_l)},\ l = 1, \cdots, L, \tag{5}$$

where $\mathbf{W}_l^* \in \mathbb{R}^{d^2 C_{l-1} \times C_l}$ has C_l filters, η is the learning rate. The global mask $\mathbf{m}$ and the filters $\mathcal{W}^*$ are updated iteratively to dynamically adapt to the pruned network. Algorithm 1 presents the detailed optimization algorithm.

In Eq. 5, we employ back-propagation to calculate the partial $\mathcal{L}$ with respect to $\mathbf{W}_l^* \odot \mathbf{m}_l$, instead of filters $\mathbf{W}_l^*$. In the framework of greedy alternative updating, $\mathbf{m}$ depends on the knowledge of $\mathcal{W}^*$, and is implemented by the global discriminative function h, which can be constructed by sorting the importance of each filter and signing all entries with 0 or 1. After that, all filters $\mathcal{W}^*$ are then masked by the global mask $\mathbf{m}$ to be updated to adapt to a newly pruned network.

Comparing to the existing solvers in layer-wised pruning, the above solver has two following advantages:

1. The saliency evaluation of filters are global (*i.e.*, across all layers), and the corresponding pruning is conducted only one time, rather than layer-by-layer.
2. We enable a dynamic updating of filters that are incorrectly masked out, which constitutes a closed circular procedure to improve the accuracy and flexibility of the pruned network.

[1]For simplicity, we discuss the problem without the bias term.

[2]These efficient implementations can take advantage of highly optimized linear algebra packages, such as Intel MKL and BLAS.

Algorithm 1 The proposed global dynamic pruning scheme

```
Input: Training data $\mathcal{D} = \{\mathcal{X}, \mathcal{Y}\}$, reference model $\mathcal{W} = \{\mathbf{W}_1^1, \cdots, \mathbf{W}_L^{C_L}\}$, sparsity threshold $\beta$, learning rate $\eta$, threshold of updating mask $e$, maximum iterations $T$.
Output: The updated parameters and their binary masks $\mathcal{W}^* = \{\mathbf{W}_l^{1*}, \cdots, \mathbf{W}_L^{C_L *}\}, \mathbf{m} = \{0,1\}^{\sum_{l=1}^L C_l}$.
1: Initialize $\mathcal{W}^*$ by $\mathcal{W}$, $\mathbf{m} = \mathbf{1}$, and $t = 0$.
2: repeat
3:     Forward Pass:
       Choose a minibatch from $\mathcal{D}$, conduct forward propagation and loss computation with $\mathcal{W}^*, \mathbf{m}$ via Eq. 3.
4:     Backward Pass:
       Compute the gradient of filter $\nabla \mathbf{W}_l^*$ by $\frac{\partial \mathcal{L}\left(\mathcal{Y}, g(\mathcal{X}; \mathcal{W}^*, \mathbf{m})\right)}{\partial (\mathbf{W}_l^* \odot \mathbf{m}_l)}$.
5:     Update:
       if Mod$((t+1), e) == 0$ then update $\mathbf{m}$ by function $h(\cdot)$ based on the current $\mathcal{W}^*$;
       Update $\mathcal{W}^*$ via Eq. 5 and the current gradient $\nabla \mathcal{W}_l^*$.
6:     $t := t + 1$.
7: until convergence or $t$ reach maximum iterations $T$.
```

To accelerate the convergence of Algorithm 1, we set a low frequency for the global mask updating, which is controlled by the threshold e. And the global mask is not updated when the network is in the warm-up phase (*i.e.*, right after finishing the mask updating). To explain, with a large loss of the network in the unstable warm-up phase, frequently updating the global mask cannot provide useful information to guide the network pruning. Therefore, we set e to be a large value in the warm-up phase, which aims to slowing down the updating frequency of the global mask. After the warm-up phase, we decrease the value e to accelerate the updating of both the global mask and the filter weights. For different networks, the detailed setups of the threshold e are discussed in our experiments subsequently.

3.4 The Global Mask

To obtain the global mask $\mathbf{m}$, a global discriminative function is further required to evaluate the importance of each filter. We introduce a criterion to measure the contribution of filters based on the Taylor expansion, termed TE.

Taylor expansion (TE). We propose a criterion based on Taylor expansion, which identifies and removes redundant filters whose removal has a limited impact to the loss function. Let $\mathbf{W}_l^{k*}$ be the k-th filter from the l-th layer, as presented in Sec. 3.2. For notation convenience, we consider the global output function, which has a global mask with all entries equal to 1, *i.e.*, $g(\mathcal{X}; \mathbf{W}^*, \mathbf{m} = \mathbf{1}) = g(\mathcal{X}; \mathbf{W}^*)$. To consider all filters with a probability to be selected as salient filters, all entries in the global mask are first set to be 1, we have:

$$\begin{aligned}\left|\Delta\mathcal{L}(\mathcal{Y}, g(\mathcal{X}; \mathbf{W}_l^{k*}))\right| = &\Big|\mathcal{L}\big(\mathcal{Y}, g(\mathcal{X}; \mathbf{W}_l^{k*} = \mathbf{0})\big) \\ &- \mathcal{L}\big(\mathcal{Y}, g(\mathcal{X}; \mathbf{W}^*)\big)\Big|,\end{aligned} \quad (6)$$

where $\mathcal{L}\big(\mathcal{Y}, g(\mathcal{X}; \mathbf{W}_l^{k*} = \mathbf{0})\big)$ evaluates the loss in the case that the k-th filter from the l-th layer is pruned, while $\mathcal{L}\big(\mathcal{Y}, g(\mathcal{X}; \mathbf{W}^*)\big)$ evaluates the loss when keeping all filters.

To facilitate discussion, the notation in Eq. 6 is simplified as:

$$\left|\Delta\mathcal{L}(\mathbf{W}_l^{k*})\right| = \left|\mathcal{L}(\mathcal{D}, \mathbf{W}_l^{k*} = \mathbf{0}) - \mathcal{L}(\mathcal{D}, \mathbf{W}^*)\right|. \quad (7)$$

Therefore, we can estimate the change of the loss $\Delta\mathcal{L}\big(\mathbf{W}_l^{k*}\big)$ by approximating $\mathcal{L}\big(\mathcal{D}, \mathbf{W}^*\big)$ with the first-order Taylor expansion at $\mathbf{W}_l^{k*} = \mathbf{0}$:

$$\left|\Delta\mathcal{L}(\mathbf{W}_l^{k*})\right| \approx \left|\frac{\partial \mathcal{L}(\mathcal{D}, \mathbf{W}^*)}{\partial \mathbf{W}_l^{k*}} \mathbf{W}_l^{k*}\right|, \quad (8)$$

where the value $\frac{\partial \mathcal{L}(\mathcal{D}, \mathbf{W}^*)}{\partial \mathbf{W}_l^{k*}}$ is obtained via back-propagation. Since the filter $\mathbf{W}_l^{k*}$ is a $d^2 C_{l-1}$-dimensional vector, we calculate the change of the loss $\left|\Delta\mathcal{L}(\mathbf{W}_l^{k*})\right|$ by accumulating the product of the loss function's gradient and the own value of filter as below[3]:

$$\left|\Delta\mathcal{L}(\mathbf{W}_l^{k*})\right| \approx \left|\sum_{r=1}^{R} \frac{\partial \mathcal{L}(\mathcal{D}, \mathbf{W}^*)}{\partial \mathbf{W}_{l,r}^{k*}} \mathbf{W}_{l,r}^{k*}\right|, \quad (9)$$

where R is the dimension of a filter. Therefore, we construct a function to measure the saliency score of a filter, *i.e.*, $f_T : \mathbf{R}^{d^2 C_{l-1}} \to \mathbf{R}^+$ with

$$f_T(\mathbf{W}_l^{k*}) = \left|\sum_{r=1}^{d^2 C_{l-1}} \frac{\partial \mathcal{L}(\mathcal{D}, \mathbf{W}^*)}{\partial \mathbf{W}_{l,r}^{k*}} \mathbf{W}_{l,r}^{k*}\right|. \quad (10)$$

The global saliency scores of all filters Ind_T is constructed, which are sorted by a descending order, *i.e.*, $Ind_T = sort\Big(\big\{f_T(\mathbf{W}_1^{1*}), \cdots, f_T(\mathbf{W}_L^{C_L *})\big\}\Big)$. Therefore, each element $\mathbf{m}_i, i = 1, 2, \cdots, \sum_{l=1}^L C_l$ in the global mask can be constructed by taking the corresponding top-$\beta \sum_{l=1}^L C_l$ index in the set:

$$\mathbf{m}_i = h_i(\mathbf{W}^*) = \begin{cases} 1, & i \in Ind_T[1 : \beta \sum_{l=1}^{L} C_l], \\ 0, & otherwise. \end{cases} \quad (11)$$

4 Experiments

4.1 Experimental Settings

We evaluate the proposed GDP approach on AlexNet [Krizhevsky *et al.*, 2012], VGG-16 [Simonyan and Zisserman, 2014] and ResNet-50 [He *et al.*, 2016] in ImageNet 2012 dataset [Russakovsky *et al.*, 2015], which contains about 1.2M training images and 50K validation images of 1,000 classes. Training images in the ImageNet dataset are rescaled to the size of 256×256, with a 224×224 (227×227 for AlexNet) crop randomly sampled from each image and its horizontal flip. The accuracy is measured on the validation set using single-view testing (central patch only).

Implementation Details. We implement our global dynamic pruning in Tensorflow [Abadi *et al.*, 2016]. To get the baseline accuracy of each network, we train AlexNet,

[3]In practice, the entire training data is divided into M minibatch, we average the $\left|\Delta\mathcal{L}(\mathbf{W}_l^{k*})\right|$ over M.

Model	Layer	FLOPs	FLOPs% GDP-D	FLOPs% GDP
VGG-16	Conv1_1	89.91M	56.25%	56.25%
	Conv1_2	1.85B	33.44%	42.24%
	Conv2_1	926.45M	32.97%	41.63%
	Conv2_2	1.85B	54.21%	54.21%
	Conv3_1	925.65M	51.12%	51.50%
	Conv3_2	1.85B	51.55%	52.75%
	Conv3_3	1.85B	98.44%	98.05%
	Conv4_1	925.25M	58.79%	49.02%
	Conv4_2	1.85B	35.94%	12.60%
	Conv4_3	1.85B	38.69%	12.56%
	Conv5_1	462.52M	46.73%	42.52%
	Conv5_2	462.52M	55.23%	79.35%
	Conv5_3	462.52M	50.56%	87.52%
	Total	15.36B	51.16%	48.03%

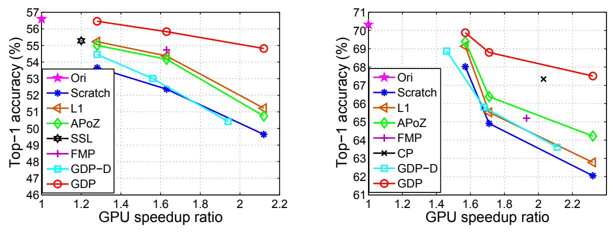
Table 1: FLOPs comparison of GDP and GDP-D, when β is set to be 0.7. FLOPs% is the percentage of the remaining FLOPs.

Method	Hy-P	FLOPs	Speedup (ms)		Top-1	Top-5
			CPU	GPU	Acc.	Acc.
AlexNet	-	729.7M	2,990	36	56.60%	80.12%
SSL	-	559.3M	2,493	30	55.28%	78.88%
FMP	-	434.8M	1,839	27	54.73%	78.53%
GDP	0.7	455.2M	2,252	28	56.46%	80.01%
	0.6	348.2M	1,760	22	55.83%	79.64%
	0.5	263.1M	1,629	17	54.82%	78.97%

Table 2: Comparing different pruning methods for accelerating AlexNet. Hy-P denotes the setting of hyper-parameter and batch size is 32 (the same in the following tables).

VGG-16 and ResNet-50 from scratch and follow the same pre-processing and hyper-parameter setting as Krizhevsky *et al.* [Krizhevsky *et al.*, 2012], Simonyan *et al* [Simonyan and Zisserman, 2014] and He *et al.* [He *et al.*, 2016], respectively. We achieve the results of each reference model as shown in Table 2, Table 3 and Table 4. We solve the optimization Eq. 4 by running on NVIDIA GTX 1080Ti GPU with 128GB of RAM. All models are trained for a total of 30 epochs with batch sizes of 128, 32 and 32 for AlexNet, VGG-16 and ResNet-50, respectively. The learning rate is set to an initial value of 0.001 and then scaled by 0.1 throughout 10 epochs. The weight decay is set to be 0.0005 and the momentum is set to be 0.9. To re-train the pruned network, we use an initial learning rate of 0.0001 for a total of 20 epochs, with a constant dropping factor of 10 throughout 10 epochs. In terms of threshold e, we use different values for different CNNs. More specifically, $e = 3$ for the first 10 epochs, $e = 2$ for the second 10 epochs and $e = 1$ for the remaining epochs is used for AlexNet. When training VGG-16 and ResNet-50, we use the same value of $e = 2$ for the first 20 epochs and $e = 1$ for the remaining 10 epochs. In terms of hyper-parameter β, we vary β in the set of $\{0.5, 0.6, 0.7\}$ with 3 values to select the best trade-off between speedup rate and classification accuracy.

Evaluation Protocols. The *Top-1* and *Top-5* classification accuracy on the validation set are employed as the evaluation protocol. We further measure the speedup ratio under the batch size of 32 to select the trade-off between speedup ratio and classification accuracy in a single-thread Intel Xeon E5-2620 CPU and NVIDIA GTX TITAN X GPU.



(a) AlexNet acceleration (b) VGG-16 acceleration

Figure 2: Comparison of different filter selection schemes for accelerating AlexNet and VGG-16. "Scratch" means that the network is trained from the scratch, and "Ori" denotes the original CNNs. GDP-D refer to global pruning filter without dynamic updating.

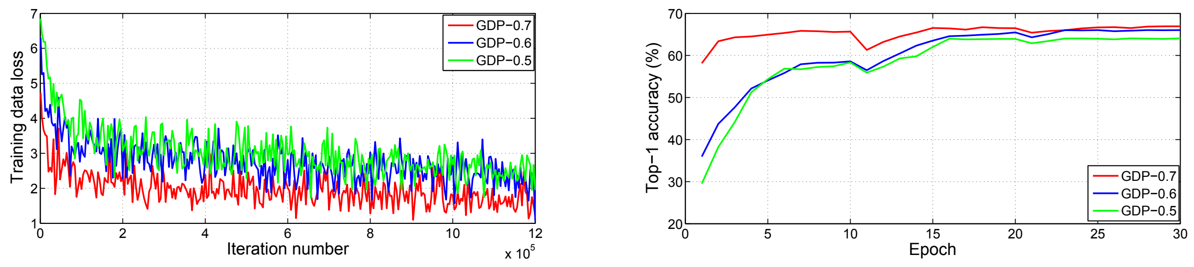
Method	Hy-P	FLOPs	Speedup (ms)		Top-1	Top-5
			CPU	GPU	Acc.	Acc.
VGG-16	-	15.5B	10,824	322	70.32%	89.42%
FMP	-	4.2B	5,237	167	65.20%	84.86%
CP	-	4.9B	5,618	159	67.34%	87.92%
GDP	0.7	7.5B	7,122	205	69.88%	89.16%
	0.6	6.4B	6,680	188	68.80%	88.77%
	0.5	3.8B	4,979	139	67.51%	87.95%

Table 3: The results of accelerating VGG-16.

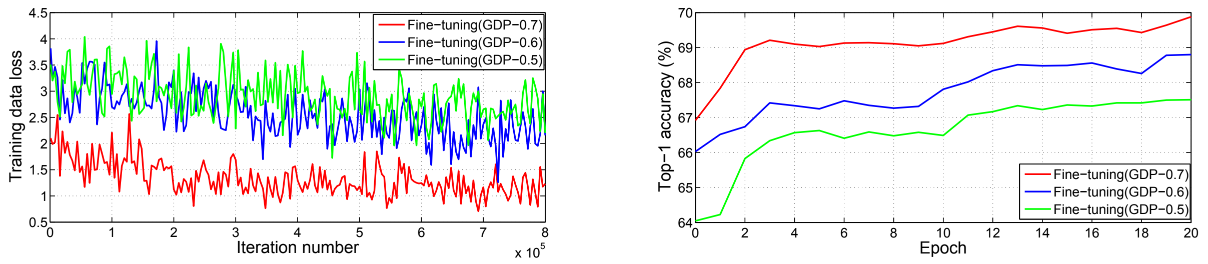
4.2 Quantitative Evaluation

We compare the proposed GDP method with the state-of-the-art filter pruning methods, including structured sparsity learning (SSL) [Wen *et al.*, 2016], ℓ_1-norm pruning (L1) [Li *et al.*, 2016], channel-based pruning (CP) [Luo *et al.*, 2017], feature map based pruning (FMP) [Molchanov *et al.*, 2017], and average percentage of zeros (APoZ) [Hu *et al.*, 2016].

AlexNet and VGG-16 on ImageNet. Both AlexNet and VGG-16 contain several convolutional layers and 3 fully-connected layers. We first compare our proposed GDP to five layer-wise filter pruning methods on GPU speedup and Top-1 accuracy. As shown in Figure 2, the results reveal three key observations: (1) Without dynamic updating (GDP-D), layer-wise pruning (*e.g.*, L1 and APoZ) performs better than global pruning. To explain, GDP-D prunes many filters with potential inter-relation at once, which leads to a significant accuracy loss even with global fine-tuning. Instead, local fine-tuning is repeatedly utilized in L1, APoZ, CP and FMP to reduce the accuracy loss, after each layer is pruned. However, repeating local fine-tuning is pretty time-consuming and seriously affects the pruning efficiency. For example, pruning each layer of VGG-16 in L1, APoZ and CP requires local fine-tuning with average 10 epochs, which requires 130 epochs in total to finish pruning the entire network, which is 6.5 times and 2.5 times more epochs than GDP-D and GDP, respectively. (2) We randomly initialize the model with the same number of filters per layer to GDP and train it from scratch, which achieves not as good accuracy as L1, APoZ and GDP. This result explicitly verifies that the initialization of deep model is pretty critical for non-convex optimization. (3) Dynamic updating is very effective to improve the network's discriminability in GDP. Compared to GDP-D, GDP employs the dynamic updating to iteratively tune the filters in a retrievable way, which can restore the salient filters with misjudgement and correspondingly improve the classification



(a) The training loss and testing Top-1 accuracy using GDP to prune VGG-16 at different β



(b) The training loss and testing Top-1 accuracy for fine-tuning the pruned network by GDP at different β

Figure 3: Comparison of different β for pruning VGG-16 via GDP scheme.

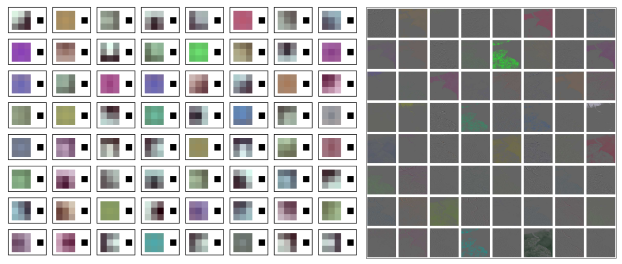
Method	Hy-P	FLOPs	Speedup (ms)		Top-1 Acc.	Top-5 Acc.
			CPU	GPU		
ResNet-50	-	3.86B	9,822	345	75.13%	92.30%
CP	-	2.44B	5,999	278	72.04%	90.67%
	-	1.71B	5,253	246	71.01%	90.02%
GDP	0.7	2.24B	6,616	279	72.61%	91.05%
	0.6	1.88B	5,821	261	71.89%	90.71%
	0.5	1.57B	5,012	242	70.93%	90.14%

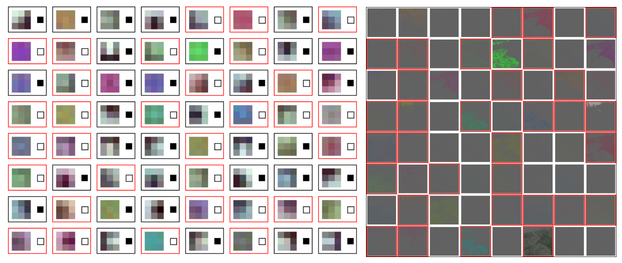
Table 4: The results of accelerating ResNet-50.

accuracy of the pruned network. For example, GDP performs 3.9% higher in Top-1 accuracy than GDP-D when pruning VGG-16 with about 2.32× speedup. Moreover, GDP tends to prune more filters in the layers with high computation complexity, which leads to the reduction of FLOPs and the increase of CNN speedup. For example, as shown in Table 1, GDP prunes more filters on the middle layers of VGG-16 with high computation complexity (*e.g.*, Conv4_1, Conv4_2 and Conv4_3), while GDP-D tends to prune more filters on the last layers (*e.g.*, Conv5_2 and Conv5_3). By equipping with the dynamic updating, GDP achieves the best performance to prune both AlexNet and VGG-16 comparing to all filter pruning schemes. For AlexNet, GDP achieves 1.1% higher in Top-1 accuracy than FMP at the about 1.63× GPU speedup. For VGG-16, the speedup rate is increased by a factor of 2.32× with 67.51% Top-1 accuracy in GDP, compared to 2.03× with 67.37% Top-1 accuracy in CP. Specifically, the detailed changed process of training loss and testing Top-1 accuracy using GDP scheme to prune VGG-16 is presented in Figure 3. The figure shows that GDP converges to a high accuracy after 30 epochs, and we can further improve the classification of the pruned network by a simple fine-tuning.

Subsequently, GDP is also compared to several state-of-the-art filter pruning methods about FLOPs reduction, CPU and GPU speedup, which is shown in Table 2 and Table 3. For better comparison, SSL [Wen *et al.*, 2016] employs the filter-wise sparsity regularization and achieves a limited computation reduction, *i.e.*, about 170M FLOPs reduction with 1.2× CPU speedup, but obtains 1.32% loss in Top-1 accuracy. As for FMP [Molchanov *et al.*, 2017], their motivation is similar to our TE mask, but with totally different filter selection and training designs. Our GDP prunes the filters at once in a retrievable way, while FMP prunes one filter permanently at a time. As shown in Table 2, GDP achieves higher Top-1/5 accuracy with more FLOPs reduction than FMP. As for CP [Luo *et al.*, 2017], it conducts a greedy local channel selection to prune the channel with the smallest channel approximated error. As shown in Table 3, CP yields a final pruned network with 3.16× FLOPs reduction, 1.9× CPU speedup and a 1.5% loss in Top-5 accuracy. Compared to CP, GDP is faster to prune the redundant filters in a global manner without intermediate feature responses, and achieve better performance with about 4× FLOPs reduction, 2.17× CPU speedup and a 1.47% Top-5 accuracy loss.

ResNet-50 on ImageNet. ResNet-50 is a more compact structure with less redundancy than AlexNet and VGG-16. Since significantly smaller FLOPs are located in the last layer and the projection shortcut layer, we only prune the first two layers of each residual block and leave the last layer and the projection shortcut layer unchanged, as to match the dimension of output. In fact, FLOPs in the last convolutional layer can be significantly reduced, since large number of channels as the input have been reduced, which is caused by pruning the number of filter in the second convolutional layer. As shown in Table 4, GDP achieves better performance in terms of pruning ResNet-50. With the increase number of filter pruning, (*i.e.*, the value of β is from 0.7 to 0.5), FLOPs can be significantly reduced via GDP, but with slight increase in Top-5 accuracy loss. Compared to CP, GDP achieves the higher Top-5 accuracy (90.14% in GDP vs. 90.02% in CP) with


(a) Epoch 0


(b) Epoch 1

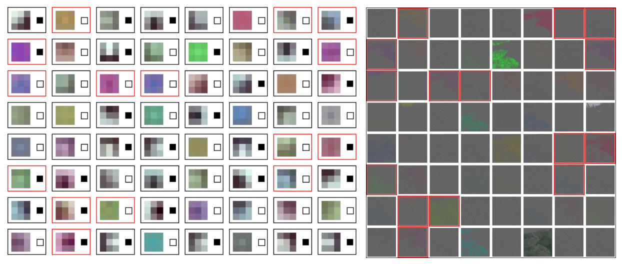

(c) Epoch 30

Figure 4: Dynamically update filters, masks and output feature maps on the first layer of VGG-16. Left: filters and masks, Right: output feature maps. In the left column, each rectangle contains filter and mask, in which the black one indicates the mask is unchanged, and the red one presents the filter and mask are updated. In addition, the mask is a smaller rectangle, in which ■ indicates the corresponding filter is salient, and □ is unsalient. In the right column, the feature maps changed correspondingly are shown in the red boxes.

a higher CPU and GPU speedup (5,012ms CPU and 242ms GPU online inference in GDP vs. 5,253ms CPU and 246ms GPU online inference in CP). To explain, dynamic updating in GDP significantly improves the discriminability and generalization of the pruned network.

4.3 Visualization of Dynamic Updating

Quantitatively, we have testified the effectiveness of dynamic updating in our global pruning scheme. To show the process of dynamic updating, we visualize filters, masks and output feature maps of the first convolutional layer for VGG-16 by using the proposed GDP method and setting β to be 0.5, as presented in Figure 4. Before pruning the filters (*i.e.*, the masks all equal to 1), low-level features (*e.g.*, edge, color and corner detectors of various directions) can be found among the listed filters and output feature maps, as shown in Figure 4(a). After the first-round mask updating, the network is pruned temporarily by selecting the salient filters based on TE, and then is updated to adapt to the pruned network, as shown in Figure 4(b). After 30 epochs, the network is convergent to adaptively obtain the salient filters and their masks by dynamic updating. In Figure 4(c), several number of filters, masks and output feature maps are different with the ones after the first updating, which indicates that the saliency of filters were mistakenly judged in the beginning, but were successfully updated during the dynamic updating.

5 Conclusion

This work presents a global dynamic pruning (GDP) scheme to prune redundant filters for CNN acceleration. We employ a global discriminative function based on prior knowledge of each filter to globally prune the unsalient filters across all layers. To decrease accuracy loss caused by incorrect globally pruning, we dynamically update the filter saliency all over the pruned sparse network. Specially, we further handle the corresponding non-convex optimization problem of the proposed GDP, which is effectively solved via stochastic gradient descent with greedy alternative updating. In experiments, the proposed GDP achieves superior performance to accelerate various cutting-edge CNNs on ILSVRC-12, comparing to the state-of-the-art filter pruning methods.

Acknowledgments

This work is supported by the National Key R&D Program (No.2017YFC0113000, and No.2016YFB1001503), the Natural Science Foundation of China (No.U1705262, No.61772443, No.61402388 and No.61572410), the Post Doctoral Innovative Talent Support Program under Grant BX201600094, the China Post-Doctoral Science Foundation under Grant 2017M612134, Scientific Research Project of National Language Committee of China (Grant No. YB135-49), and Natural Science Foundation of Fujian Province, China (No. 2017J01125 and No. 2018J01106).

References

[Abadi *et al.*, 2016] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[Anwar *et al.*, 2015] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *arXiv preprint arXiv:1512.08571*, 2015.

[Courbariaux and Bengio, 2016] M. Courbariaux and Y. Bengio. Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[Courbariaux *et al.*, 2015] M. Courbariaux, Y. Bengio, and J. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*, 2015.

[Denton *et al.*, 2014] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, pages 1269–1277, 2014.

[Girshick *et al.*, 2014] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, pages 580–587, 2014.

[Gong *et al.*, 2014] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[Guo *et al.*, 2016] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *NIPS*, pages 1379–1387, 2016.

[Han *et al.*, 2015a] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR, abs/1510.00149*, 2, 2015.

[Han *et al.*, 2015b] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NIPS*, pages 1135–1143, 2015.

[Han *et al.*, 2016] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *ISCA*, 2016.

[Hassibi and Stork, 1993] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *NIPS*, 1993.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[Hu *et al.*, 2016] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[Jaderberg *et al.*, 2014] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[Jia *et al.*, 2014] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM MM*, pages 675–678. ACM, 2014.

[Kim *et al.*, 2015] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[Lebedev and Lempitsky, 2016] Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *CVPR*, pages 2554–2564, 2016.

[Lebedev *et al.*, 2014] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

[LeCun *et al.*, 1989] Yann LeCun, John S Denker, Sara A Solla, Richard E Howard, and Lawrence D Jackel. Optimal brain damage. In *NIPS*, 1989.

[Li *et al.*, 2016] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[Lin *et al.*, 2016] Shaohui Lin, Rongrong Ji, Xiaowei Guo, and Xuelong Li. Towards convolutional neural networks compression via global error reconstruction. In *IJCAI*, pages 1573–1759, 2016.

[Lin *et al.*, 2017] Shaohui Lin, Rongrong Ji, Chao Chen, and Feiyue Huang. Espace: Accelerating convolutional neural networks via eliminating spatial & channel redundancy. In *AAAI*, pages 1424–1430, 2017.

[Liu *et al.*, 2015] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *CVPR*, pages 806–814, 2015.

[Long *et al.*, 2015] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, pages 3431–3440, 2015.

[Luo *et al.*, 2017] Jianhao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, 2017.

[Molchanov *et al.*, 2017] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *ICLR*, 2017.

[Park *et al.*, 2017] Jongsoo Park, Sheng Li, Wei Wen, Ping Tak Peter Tang, Hai Li, Yiran Chen, and Pradeep Dubey. Faster cnns with direct sparse convolutions and guided pruning. In *IJCAI*, 2017.

[Rastegari *et al.*, 2016] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *arXiv preprint arXiv:1603.05279*, 2016.

[Russakovsky *et al.*, 2015] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[Simonyan and Zisserman, 2014] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[Wen *et al.*, 2016] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *NIPS*, 2016.

[Wu *et al.*, 2016] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *CVPR*, 2016.