# Structured Deep Neural Network Pruning by Varying Regularization Parameters

**Huan Wang, Qiming Zhang, Yuehai Wang*, Roland Hu***

Zhejiang University

{huanw, qmzhang, wyuehai, haoji_hu}@zju.edu.cn

## Abstract

Convolutional Neural Networks (CNN's) are restricted by their massive computation and high storage. Parameter pruning is a promising approach for CNN compression and acceleration, which aims at eliminating redundant model parameters with tolerable performance loss. Despite its effectiveness, existing regularization-based parameter pruning methods usually assign a fixed regularization parameter to all weights, which neglects the fact that different weights may have different importance to CNN. To solve this problem, we propose a theoretically sound regularization-based pruning method to incrementally assign different regularization parameters to different weights based on their importance to the network. On AlexNet and VGG-16, our method can achieve $4\times$ theoretical speedup with similar accuracies compared with the baselines. For ResNet-50, the proposed method also achieves $2\times$ acceleration and only suffers 0.1% top-5 accuracy loss.

## 1 Introduction

In recent years, deep convolutional neural networks (CNN's) have made remarkable success in computer vision tasks such as classification, detection and segmentation by leveraging large-scale networks learning from big amount of data. However, CNN leads to massive computation and storage consumptions, thus hindering its deployment on mobile and embedded devices. To reduce computation cost, many research works are carried out to compress the scales of CNN's, which include designing compact network architectures, parameter quantization, matrix decomposition and parameter pruning.

Parameter pruning is a promising approach for CNN compression and acceleration, which aims at eliminating redundant model parameters with tolerable performance loss. One problem of parameter pruning is that it often produces unstructured and random connections which is hard to implement for speedup on general hardware platforms [Han *et*

---

*Contact author

*al.*, 2016]. Even with sparse matrix kernels, the speedup is very limited [Wen *et al.*, 2015]. To solve this problem, many works focus on structured pruning which can shrink a network into a thinner one so that the implementation of the pruned network is efficient [Anwar and Sung, 2016; Sze *et al.*, 2017].

There are two categories for structured pruning. The first is importance-based methods, which prune weights in groups based on some established importance criteria. Another category is regularization-based methods, which add group regularization terms to the objective function and prune the weights by optimizing the objective function. In this paper, we propose a new regularization-based method to learn group sparsity which also combines the idea of importance-based methods.

Existing group regularization approaches tend to use the *same* regularization parameter for all weight groups in the network [Wen *et al.*, 2015; Lebedev and Lempitsky, 2016]. One hidden assumption of equally assigning regularization parameters is that all weights in different groups are equally important. However, intuitively, weights with greater magnitude tend to be more important during pruning than those with smaller magnitude. Thus, we need a scheme to distinguish the importance of weights and treat them with different regularization strengths. Compared with previous works, the main novelty of our proposed method is to assign different regularization parameters to different weight groups based on their estimated importance.

One advantage of the proposed method is that it is theoretically sound. In this paper, we theoretically prove that the proposed scheme of varying regularization parameters results in zeroing out unimportant weights and finally make the network converge to a pruned one. Considering that many methods in parameter pruning are heuristic, the proposed method stands out with concrete theoretical basis.

Our method accelerates AlexNet for $4\times$ theoretical speedup while improving its performance, and VGG-16 for $4\times$ theoretical speedup with 0.8% accuracy loss. The proposed method can also be applied to compact multi-branch networks such as ResNet-50, on which we achieves $2\times$ speedup and only suffers 0.1% accuracy loss.

## 2 Related Work

Parameter pruning has a long history in the development of neural networks [Reed, 1993], which can be categorized into importance-based and regularization-based methods. Importance-based methods prune weights in groups based on some established importance criteria. For example, Optimal Brain Damage (OBD) [LeCun *et al.*, 1990] and Optimal Brain Surgery (OBS) [Hassibi and Stork, 1993] proposed an importance criteria based on the second-order derivatives of loss function derived from Taylor expansions. Deep compression [Han *et al.*, 2015a; Han *et al.*, 2015b] pruned small-magnitude weights and obtained $9 \sim 13\times$ parameter reduction on AlexNet and VGG-16. Taylor Pruning [Molchanov *et al.*, 2017] also derived a new importance criteria based on Taylor expansion, but they used the first-order derivatives, which is easy to access in back-propagation. Their method was shown effective to prune filters of CNN's on transfer learning tasks. [Li *et al.*, 2017] used $L_1$ norm to guide one-shot filter pruning, which was proved effective on CIFAR-10 and ImageNet with VGG-16 and ResNet, but the reported speedup is very limited. Channel Pruning [He *et al.*, 2017] alternatively uses LASSO-regression-based channel selection and feature map reconstruction to prune filters, and achieves the state-of-the-art result on VGG-16. Among regularization-based methods, [Weigend *et al.*, 1991] added a penalty-term to eliminate redundant weights for better generalization. Group-wise Brain Damage [Lebedev and Lempitsky, 2016] and Structured Sparsity Learning [Wen *et al.*, 2015] embedded Group LASSO [Yuan and Lin, 2006] into CNN regularization and obtained regular-shape sparsity.

Apart from parameter pruning, there are other methods for CNN model compression and acceleration, including designing compact architecture, parameter quantization and matrix decomposition. Compact architecture designing methods target designing more efficient and compact neural network architectures. For example, SqueezeNet [Iandola *et al.*, 2016] was proposed to stack compact blocks, which decreased the number of parameters by $50\times$ less than the original AlexNet. MobileNet [Howard *et al.*, 2017] and ShuffleNet [Zhang *et al.*, 2017] leveraged new convolutional implementations to design networks for mobile applications.

Parameter quantization methods reduce CNN storage by quantizing the weights and using less representation bits. [Chen *et al.*, 2015] proposed a hash function to group weights of each CNN layer into different hash buckets for parameter sharing. As the extreme form of quantization, binarized networks [Courbariaux and Bengio, 2016; Lin *et al.*, 2016; Rastegari *et al.*, 2016] proposed to learn binary weights or activations. Quantization reduces floating computational complexity, but the actual speedup may be very related to specific hardware implementations.

Matrix decomposition decomposes large matrices into several small matrices to reduce computation. [Denton *et al.*, 2014] showed that the weight matrix of a fully-connected layer can be compressed via truncated SVD. Tensor decomposition was proposed and obtained better compression result than SVD [Novikov *et al.*, 2014]. Several methods based on low-rank decomposition of convolutional kernel tensors were also proposed to accelerate convolutional layers [Lebedev *et al.*, 2016; Zhang *et al.*, 2016].

## 3 The Proposed Method

Suppose weights of convolutional layers in a CNN form a sequence of 4-D tensors $\mathbf{W}^{(l)} \in \mathbb{R}^{N_l \times C_l \times M_l \times K_l}$. Here $N_l$, $C_l$, $M_l$ and $K_l$ are the dimension of the $l$th ($1 \leq l \leq L$) weight tensor along the axis of filter, channel, spatial height and spatial width, respectively. Our proposed objective function for regularization can be formulated as:

$$E(\mathbf{W}) = L(\mathbf{W}) + \frac{\lambda}{2}R(\mathbf{W}) + \frac{\lambda_g}{2}\sum_{l=1}^{L} R_g(\mathbf{W}^{(l)}) \quad (1)$$

Here $\mathbf{W}$ represents the collection of all weights in the CNN; $L(\mathbf{W})$ is the loss function caused by data; $R(\mathbf{W})$ is non-structured regularization on every weight, which is the $L_2$ norm in this paper. $R_g$ is the structured sparsity regularization on each layer. In [Wen *et al.*, 2015; Lebedev and Lempitsky, 2016], the authors used the same $\lambda_g$ for all groups and adopted Group LASSO for $R_g$ because it can effectively zero out weights in groups, i.e., $R_g(\omega) = \sum_{g=1}^{G} \|\omega^{(g)}\|$. In this paper, we use the squared $L_2$ norm for $R_g$, i.e., $R_g(\omega) = \sum_{g=1}^{G} \|\omega^{(g)}\|^2$ and vary the regularization parameters $\lambda_g$ for different groups.

The finally learned 'structure' is decided by the way of splitting groups of $\omega^{(g)}$. Normally, there are filter-wise, channel-wise and shape-wise sparsity with different size of weight groups [Wen *et al.*, 2015]. The sparsity terms $R_g$ are represented as

$$R_{g(filter)} = \frac{\lambda_g}{2}\sum_{l=1}^{L}\left(\sum_{n_l=1}^{N_l} \|\mathbf{W}_{n_l,:,:,:}^{(l)}\|^2\right)$$

$$R_{g(shape)} = \frac{\lambda_g}{2}\sum_{l=1}^{L}\left(\sum_{c_l=1}^{C_l}\sum_{m_l=1}^{M_l}\sum_{k_l=1}^{K_l} \|\mathbf{W}_{:,c_l,m_l,k_l}^{(l)}\|^2\right)$$

$$R_{g(channel)} = \frac{\lambda_g}{2}\sum_{l=1}^{L}\left(\sum_{c_l=1}^{C_l} \|\mathbf{W}_{:,c_l,:,:}^{(l)}\|^2\right)$$

In im2col implementation, weight tensors are expanded into matrices where shapes are represented as columns and filters are represented as rows. Thus, shape and filter-wise sparsity are equivalent to column and row-sparsity, as shown in Fig.1.

### 3.1 Theoretical Analysis

For the proposed method, the regularization parameter $\lambda_g$ is differently assigned for different weight groups. We find that by slightly augmenting $\lambda_g$ of a weight group, and then train the network through back-propagation until the objective function, i.e. Eqn.(1), reaches the local minimum, the $L_2$ norm of that group will also decrease. This phenomenon is formally summarized by the following theorem.

**Theorem 1** *Considering the objective function*

$$E(\lambda, \omega) = L(\omega) + \frac{\lambda}{2}\omega^2, \quad (2)$$

(a) filter sparsity/row sparsity      (b) shape sparsity/column sparsity      (c) channel sparsity
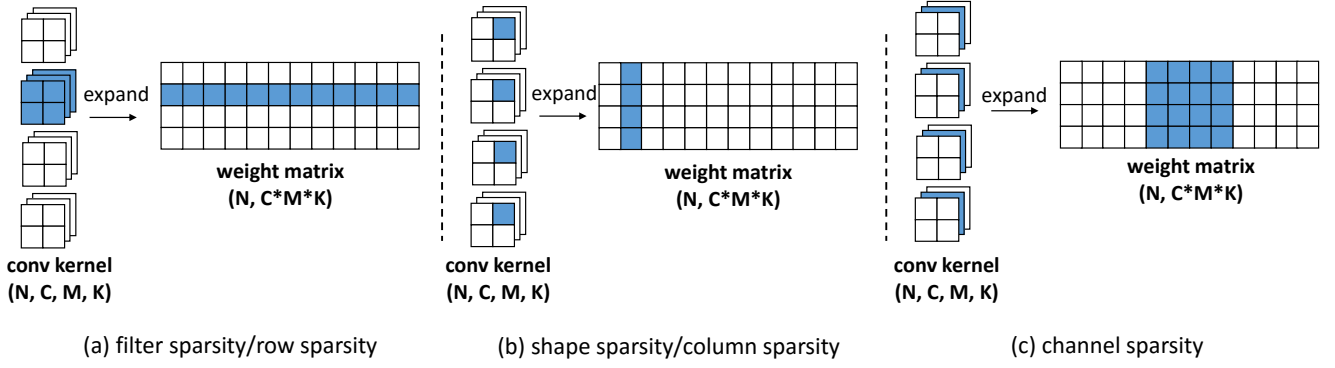
Figure 1: The im2col implementation of CNN is to expand tensors into matrices, so that convolutional operations are transformed to matrix multiplication. The weights at the blue squares are to be pruned. (a) Pruning a row of weight matrix is equivalent to pruning a filter in convolutional kernel. (b) Pruning a column of weight matrix is equivalent to pruning all the weights at the same position in different filters. (c) Pruning a channel is equivalent to pruning several adjacent columns in the expanded weight matrix.

*if there exists a tuple $(\lambda_0, \omega_0)$ which satisfies the following three properties:*

*1. $\lambda_0 > 0$;*

*2. $L(\omega)$ has the second derivative at $\omega_0$;*

*3. $\omega_0$ is the local minimum of function $Y_{\lambda_0}(\omega) = E(\lambda_0, \omega)$,*

*then there exists an $\epsilon > 0$ that for any $\lambda_1 \in (\lambda_0, \lambda_0 + \epsilon)$, we can find an $\omega_1$ which satisfies:*

*1. $\omega_1$ is the local minimum of function $Y_{\lambda_1}(\omega) = E(\lambda_1, \omega)$;*

*2. $|\omega_1| < |\omega_0|$.*

This theorem is proved in Appendix A, which indicates that we can slightly increase the regularization parameter to compress the magnitude of weights to zero. By Eqn.(12), the magnitude of weights will be more compressed if the regularization parameter increases more.

## 3.2 Method Description

Theorem 1 guarantees that we can modify the $L_1$ norm of weight groups by increasing or decreasing their corresponding regularization parameters. Thus, we can assign different regularization parameters to weight groups based on their importance to the network. In this paper, we use the $L_1$ norms as the importance criterion for weight groups. Note that our method can be easily applied to other criteria such as $L_2$ norm and Taylor expansions.

Normalization of importance criteria is necessary because the values of $L_1$ norms have huge variations across different networks, layers and weight groups. Contrary to previous works, normalization is based on the ranks of weight groups in the same layer. The advantages of rank-based normalization lies in two parts – (1) Compared to other normalization methods like max/min normalization, the range of ranks is fixed from 0 to $N_g-1$, where $N_g$ is the total number of weight groups in the layer; (2) For the pruning task, we need to set a pruning ratio $R$ to each layer, say, $R = 0.6$ means that we need to prune $60\%$ of weight groups which are ranked the lowest when pruning is finished. Normalization by ranks makes the pruning process controllable since it is directly towards the goal of pruning.

Specifically, we sort weight groups by their $L_1$ norms in ascending order. Meanwhile, to mitigate the oscillation of ranks in one training iteration, we average the rank of each group through training iterations. For a weight group, its average rank through $N$ iterations is defined as

$$\bar{r}_{avg} = \frac{1}{N} \sum_{n=1}^{N} r_n. \tag{3}$$

Here $r_n$ is the rank of the $n$th iteration. The final average rank $\bar{r}$ is obtained by sorting $\bar{r}_{avg}$ of different weight groups in ascending order, making its range from 0 to $N_g - 1$.

Our aim is to assign an increment $\Delta\lambda_g$ to each weight group, so that its regularization parameter $\lambda_g$ is gradually updated through the pruning process.

$$\lambda_g^{(new)} = \lambda_g + \Delta\lambda_g \tag{4}$$

Following the above idea, $\Delta\lambda_g$ of each group is assigned by its average rank $\bar{r}$ with a piecewise linear function, as shown in Eqn.(5).

$$\Delta\lambda_g(\bar{r}) = \begin{cases} -\dfrac{A}{RN_g}\bar{r} + A & \text{if } \bar{r} \le RN_g \\[2ex] -\dfrac{A}{N_g(1-R)-1}(\bar{r} - RN_g) & \text{if } \bar{r} > RN_g \end{cases} \tag{5}$$

Fig.2 depicts $\Delta\lambda_g(\bar{r})$. It is seen from Fig.2 that for weight groups whose $L_1$ norms are small, i.e., the average ranks less than $RN_g$, we need to increase their regularization parameters to further decrease their $L_1$ norms; and those with greater $L_1$ norms and rank above $RN_g$, we need to decrease regularization parameters to further increase their $L_1$ norms. After obtaining $\Delta\lambda_g$ and $\lambda_g^{(new)}$ by Eqn.(5) and (4), we threshold it by zero to prevent negative values of regularization:

$$\lambda_g^{(new)} = \max(\lambda_g + \Delta\lambda_g, 0). \tag{6}$$

After updating $\lambda_g$, the weights of CNN are trained through back-propagation deduced from Eqn.(1).

$$\omega^{(new)} = \omega - \alpha \left[ \frac{\partial L}{\partial \omega} + (\lambda + \lambda_g)\omega \right]. \tag{7}$$
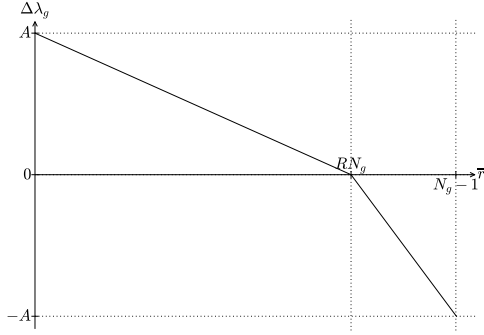
Figure 2: The functional of $\Delta\lambda_g(\bar{r})$, as defined in Eqn.(5).

After training the weights for several iterations, we recalculate $\lambda_g$ and the training process continues until convergence. Since we decrease the $L_1$ norms of weight groups whose ranks are less than $RN_g$ and increase the $L_1$ norms of weight groups whose ranks are greater than $RN_g$, there should be $RN_g$ pruned weight groups at the convergence point. In Eqn.(5), $A$ is a hyper-parameter to control the speed of convergence. Greater value of $A$ results in faster convergence. Finally, we summarize the proposed algorithm in Algorithm 1.

---

**Algorithm 1** The Proposed Algorithm

---

1: Input the training set $D$, the original pre-trained model $\Omega$, the non-structural regularization parameter $\lambda$, pruning threshold $\epsilon$ and target pruning ratio $R$.
2: Initialize $\lambda_g = 0$ for each weight group.
3: Initialize the iteration number $i = 0$.
4: **repeat**
5:     **for** each weight group in each layer **do**
6:         Obtain $\bar{r}$ of by Eqn.(3).
7:         Update $\lambda_g$ by Eqn.(5) and (6).
8:     **end for**
9:     Update weights in each weight groups through back-propagation by Eqn.(7).
10:     **for** each weight group in each layer **do**
11:         **if** the $L_1$ norm of this group is less than $\epsilon$ **then**
12:             prune this weight group permanently.
13:         **end if**
14:     **end for**
15:     $i = i + 1$.
16: **until** The ratio of pruned weight groups reaches $R$.
17: Retrain the pruned CNN for several epochs.
18: Output the pruned CNN model $\Omega'$.

---

### 3.3 Discussions

The difference between Algorithm 1 and Theorem 1 is that Theorem 1 requires weights reach local minimum of the objective function (Eqn.1) before another updates of $\lambda_g$, but in Algorithm 1, we only train the network through back-propagation for several iterations. Such a compromise is mainly attributed to time-complexity. Actually, it is time-consuming if we obtain local minimum by gradually lower-

ing the learning rate $\alpha$. In addition, because objective functions of CNN's are very complex, it is also difficult to decide whether local minimum is obtained or not. Experiments indicate that $3 \sim 5$ iterations are enough for convergence of the method.

An amazing factor of the proposed method is that it can automatically adjust search steps without any knowledge about the property of the objective function itself. By Eqn.(12), the increment of weight $d\omega$ is reversely proportional to the second derivative of the objective function $L(\omega)$ which is produced by CNN. Such a property makes the modification of $\omega$ slower when the objective function $L(\omega)$ reaches steeper areas even without knowing the exact form of $L(\omega)$, which is a nice property to make refined search of local minimum possible. We believe that the good performance of the proposed method is partly because of the automatic adjustment of search steps.

## 4 Experiments

We first compare fixed and varying regularization with ConvNet on CIFAR-10 dataset [Krizhevsky and Hinton, 2009]. Then we evaluate the proposed method with large networks on large scale datasets. All of our experiments are conducted with Caffe. We set the weight decay factor $\lambda$ the same with baselines, and set the hyper-parameter $A$ as half of $\lambda$. Since we focus on acceleration in this paper, we only compress weights in convolutional layers and keep fully connected layers unchanged. Methods for comparison include Taylor Pruning (TP) [Molchanov *et al.*, 2017], Filter Pruning (FP) [Li *et al.*, 2017], Structured Sparsity Learning (SSL) [Wen *et al.*, 2015], Channel Pruning (CP) [He *et al.*, 2017] and Structured Probabilistic Pruning (SPP) [Wang *et al.*, 2017]. For all experiments, speedup is calculated by GFLOPS reduction.

### 4.1 Analysis with ConvNet on CIFAR-10

We firstly compare our proposed method with the Group LASSO [Yuan and Lin, 2006] where the regularization parameter is fixed for all weights. Group LASSO is widely applied to generate group sparsity [Lebedev and Lempitsky, 2016; Wen *et al.*, 2015]. The test network is ConvNet, which is a small CNN with three convolutional layers and one fully-connected layers. CIFAR-10 is a 10-class dataset of $60k$ tiny images, among which $45k$ images are used for training, $5k$ images for validation and the other $10k$ images for testing.

We first trained a baseline model with test accuracy $81.5\%$. Then the proposed method is applied to learn structured sparsity, where both row sparsity and column sparsity are explored. For comparison, we employ the Group LASSO with fixed regularization parameters ($0.04$ for row sparsity and $0.01$ for column sparsity).

Experimental results are shown in Tab.1. We can see that varying regularization consistently achieves higher speedups and accuracies than fixed regularization. Fig.3 illustrates the process that the $L_1$ norms of columns changes with training iterations in the conv1 layer, from which we can see that while fixed regularization gradually suppresses unimportant weights, many important weights are also unnecessarily suppressed. In varying regularization, the magnitude

| Method | Row pruning | | Column pruning | |
|---|---|---|---|---|
| | speedup | accuracy | speedup | accuracy |
| SSL | 3.6× | 77.3 | 3.1× | 78.6 |
| Ours | 4.1× | **79.2** | 4.6× | **81.2** |

Table 1: Comparison of varying and fixed regularization with ConvNet on CIFAR-10. The baseline test accuracy is $81.5\%$.
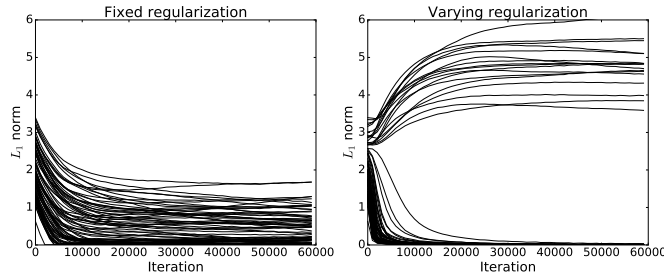


Figure 3: Comparing the training process of varying regularization with that of fixed regularization. Each line illustrates the $L_1$ norm of a column in the conv1 layer (75 columns in total).

of some weights increases dramatically when training starts while some decreases meanwhile. Thus, the pruning process converges much faster. In addition, with varying regularization, the final weights are driven into two distinct groups – the group formed by important weights with very high $L_1$ norms and the group for unimportant weights with close-to-zero $L_1$ norms.

We also find that under similar speedup ratios, column pruning is better than row pruning in accuracy. It is because that a row typically consists of much more weights than a column and pruning rows may cause more severe side-effects for accuracy. In the following experiments, we only choose column as our sparsity group to obtain better accuracies.

We further compare our proposed method with more recent pruning methods, and the results are shown in Tab.2. Under different speedup ratios, our method consistently outperforms other pruning methods.

| Method | Increased err. (%) | | |
|---|---|---|---|
| | 2× | 4× | 6× |
| TP (our impl.) | 1.0 | 3.4 | 5.4 |
| FP (our impl.) | 1.6 | 3.6 | 5.0 |
| SSL | 3.0 | 5.8 | 7.4 |
| SPP | -0.2 | 0.3 | 1.2 |
| Ours | **-0.5** | **0.1** | **1.0** |

Table 2: The increased error of different pruning methods when accelerating ConvNet on CIFAR-10. The baseline test accuracy is $81.5\%$. Minus means the test accuracy is improved.

## 4.2 AlexNet on ImageNet

We apply the proposed method to AlexNet [Krizhevsky *et al.*, 2012], which is composed of 5 convolutional layers and 3 fully-connected layers. We download an open caffemodel

from Caffe model zoo as our pre-trained model. The baseline single view top-5 accuracy on ImageNet 2012 validation dataset is $80.0\%$. All images are rescaled to size $256 \times 256$, then a $227 \times 227$ patch is randomly cropped from each scaled image and randomly mirrored for data augmentation. For testing, the $227 \times 227$ patches are cropped from the center of the scaled images.

Intuitively, different layers have different sensitivity to pruning, but there are few theories to quantify the redundancy of different layers in deep neural networks. Most pruning methods empirically set pruning ratio for different layers [Li *et al.*, 2017; Molchanov *et al.*, 2017; He *et al.*, 2017; Wang *et al.*, 2017]. Following these works, we empirically set the proportion of remaining columns of the five convolutional layers as $1 : 1 : 1 : 1.5 : 1.5$. We train the baseline model with batch size $64$ for about 7 epochs until reaching the target pruning ratio. We use a small learning rate $\alpha = 0.0005$ and fix it during training. Then the pruned model is retrained with batch size $256$ to regain accuracy.

Experimental results are shown in Tab.3. The proposed

| Method | Increased err. (%) | | |
|---|---|---|---|
| | 2× | 4× | 5× |
| TP | 3.9 | 9.2 | 13.9 |
| FP (SPP's impl.) | 0.6 | 4.1 | 4.7 |
| SSL | 1.3 | 4.3 | 5.3 |
| SPP | **-0.7** | 0.3 | 0.9 |
| Proposed | **-0.7** | **-0.2** | **0.8** |

Table 3: Accelerating AlexNet on ImageNet. The baseline top-5 accuracy of the original network is $80.0\%$.

method and SPP are consistently better than the other three methods, and the proposed method is slightly better than SPP on average. With $4\times$ speedup, our method can even improve the accuracy while SPP degrades the accuracy in this situation.

## 4.3 VGG-16 on ImageNet

We further demonstrate our method on VGG-16 [Simonyan and Zisserman, 2014], which has 13 convolutional layers and 3 fully-connected layers. We download the open caffemodel as our pre-trained model, whose single-view top-5 accuracy on ImageNet 2012 validation dataset is $89.6\%$. Similar to AlexNet, the resized images are randomly cropped to input size $224 \times 224$ and mirrored for data augmentation in training.

Previous works [He *et al.*, 2017; Wang *et al.*, 2017] found that lower layers were more redundant in VGG-16. Therefore, the proportion of remaining ratios of low layers (conv1_x to conv3_x), middle layers (conv4_x) and high layers (conv5_x) are set to $1 : 1.5 : 2$, the same as [Wang *et al.*, 2017]. The first and last convolutional layer (conv1_1 and conv5_3) are not pruned because both of them require very small amount of GFLOPs calculation. We first train the baseline model with batch size $64$ and with a fixed learning rate $0.0005$. Pruning is finished after around 6 epochs. Then the network is fine-tuned with batch size $256$ to regain accuracy.

Experimental results are shown in Tab.4. Our method is slightly better than CP and SPP, and beats TP and FP by a significant margin.

| Method | Increased err. (%) | | |
|---|---|---|---|
| | 2× | 4× | 5× |
| TP | − | 4.8 | − |
| FP (CP's impl.) | 0.8 | 8.6 | 14.6 |
| CP | **0** | 1.0 | 1.7 |
| SPP | **0** | **0.8** | 2.0 |
| Ours | **0** | **0.8** | **1.5** |

Table 4: Acceleration of VGG-16 on ImageNet. The values are increased single-view top-5 error on ImageNet. The baseline top-5 accuracy of the original network is 89.6%.

## 4.4 ResNet-50 on ImageNet

Unlike AlexNet and VGG-16, ResNet-50 is a multi-branch deep neural network, which has 53 convolutional layers and no fully-connected layers. Open pre-trained caffemodel is adopted as our baseline, whose single view top-5 accuracy on ImageNet 2012 validation dataset is 91.2%. The images are augmented the same way as in the experiment of VGG-16 (Sec.4.3).

For simplicity, we adopt the same pruning ratio 0.4 for all 53 convolutional layers. The training settings are similar to that of VGG-16. The pruning process stops after less than 2 epochs before retraining.

From Tab.(5), our method achieves much better results than two recent methods CP and SPP. This is probably because the proposed method imposes gradually regularization, making the network adapt little-by-little in the parameter space, while both the matrix decomposition in CP and direct pruning in SPP may bring much modification that a network as compact as ResNet-50 cannot endure.

| Method | Increased err. (%) |
|---|---|
| CP (enhanced) | 1.4 |
| SPP | 0.8 |
| Proposed | **0.1** |

Table 5: Acceleration of ResNet-50 on ImageNet. The baseline top-5 accuracy of the original network is 91.2%.

## 5 Conclusion

We propose structured sparsity pruning with varying regularization parameters for CNN acceleration, which assigns different regularization parameters to weight groups according to their importance to the network. Theoretical analysis guarantees the convergence of our method. The effectiveness of the proposed method is proved by comparison with state-of-the-art methods on popular CNN architectures.

## Appendix A – Proof of Theorem 1

**Proof:** For a given $\lambda > 0$, the $\omega$ which is the local minimum of the function $Y_\lambda(\omega) = E(\lambda, \omega)$ should satisfy $\frac{dY_\lambda(\omega)}{d\omega} = 0$, which gives:

$$\lambda = -\frac{L'(\omega)}{\omega}. \tag{8}$$

In this situation, we can calculate the derivative of $\lambda$ by using Eqn.(8):

$$\frac{d\lambda}{d\omega} = \frac{L'(\omega) - \omega L''(\omega)}{\omega^2}. \tag{9}$$

Since $\omega_0$ is the local minimum of the function $Y_{\lambda_0}(\omega) = E(\lambda_0, \omega)$, it should satisfy that

$$\frac{dY_{\lambda_0}(\omega)}{d\omega}\bigg|_{\omega_0} = 0 \text{ and } \frac{d^2 Y_{\lambda_0}(\omega)}{d\omega^2}\bigg|_{\omega_0} > 0,$$

which yields

$$L'(\omega_0) + \lambda_0 \omega_0 = 0 \tag{10}$$
$$L''(\omega_0) + \lambda_0 > 0 \tag{11}$$

If we take Eqn.(10) into (9), we can obtain

$$\frac{d\lambda}{d\omega}\bigg|_{(\lambda_0, \omega_0)} = -\frac{L''(\omega_0) + \lambda_0}{\omega_0} \tag{12}$$

By taking Eqn.(11) into (12), we can conclude that

$$\begin{cases} \frac{d\lambda}{d\omega}\bigg|_{(\lambda_0, \omega_0)} < 0, & \text{if } \omega_0 > 0 \\ \frac{d\lambda}{d\omega}\bigg|_{(\lambda_0, \omega_0)} > 0, & \text{if } \omega_0 < 0 \end{cases} \tag{13}$$

In other words, when $\omega_0$ is greater than zero, a small increment of $\lambda_0$ will decrease the value of $\omega_0$; and when $\omega_0$ is less than zero, a small increment of $\lambda_0$ will increase the value of $\omega_0$. In both cases, when $\lambda_0$ increases, $|\omega_0|$ will decrease at the new local minimum of $E(\lambda, \omega)$.

Thus, we finished the proof of Theorem 1.

## References

[Anwar and Sung, 2016] S. Anwar and W. Sung. Compact deep convolutional neural networks with coarse pruning. *arXiv preprint*, arXiv:1610.09639, 2016.

[Chen *et al.*, 2015] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *Proceedings of the International Conference on Machine Learning, ICML*, pages 1–10, Lille, France, 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.

[Denton *et al.*, 2014] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems, NIPS*, Montréal, Canada, 2014.

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

[Han *et al.*, 2015b] S. Han, J. Pool, and J. Tran. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems, NIPS*, pages 1135–1143, Montréal, Canada, 2015.

[Han *et al.*, 2016] S. Han, X. Liu, and H. Mao. EIE: efficient inference engine on compressed deep neural network. *ACM Sigarch Computer Architecture News*, 44(3):243–254, 2016.

[Hassibi and Stork, 1993] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Information Processing Systems, NIPS*, pages 164–171, Denver, CO, 1993.

[He *et al.*, 2017] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. 2017.

[Howard *et al.*, 2017] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. 2017.

[Iandola *et al.*, 2016] F. Iandola, M. Moskewicz, and K. Ashraf. SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv preprint*, arXiv:1602.07360, 2016.

[Krizhevsky and Hinton, 2009] A. Krizhevsky and G. E. Hinton. Learning multiple layers of features from tiny images. 2009.

[Krizhevsky *et al.*, 2012] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Information Processing Systems, NIPS*, pages 1097–1105, Lake Tahoe, CA, 2012.

[Lebedev and Lempitsky, 2016] V. Lebedev and V. Lempitsky. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition, CVPR*, pages 2554–2564, Las Vegas, NV, 2016.

[Lebedev *et al.*, 2016] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky. Speeding-up convolutional neural networks using fine-tuned CP-decomposition. *arXiv preprint*, arXiv:1510.03009, 2016.

[LeCun *et al.*, 1990] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in Information Processing Systems, NIPS*, pages 598–605, Denver, CO, 1990.

[Li *et al.*, 2017] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. In *International Conference on Learning Representations, ICLR*, 2017.

[Lin *et al.*, 2016] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio. Neural networks with few multiplications. *arXiv preprint*, arXiv:1510.03009, 2016.

[Molchanov *et al.*, 2017] P. Molchanov, S. Tyree, and T. Karras. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations, ICLR*, 2017.

[Novikov *et al.*, 2014] A. Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov. Tensorizing neural networks. *arXiv preprint*, arXiv:1509.06569, 2014.

[Rastegari *et al.*, 2016] M. Rastegari, V. Ordonez, J. Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision, ECCV*, pages 525–542, Amsterdam, Netherland, 2016.

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

[Simonyan and Zisserman, 2014] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *Computer Science*, 2014.

[Sze *et al.*, 2017] V. Sze, Y. H. Chen, T. J. Yang, and J. Emer. Efficient processing of deep neural networks: A tutorial and survey. *arXiv preprint*, arXiv:1703.09039, 2017.

[Wang *et al.*, 2017] Huan Wang, Qiming Zhang, Yuehai Wang, and Roland Hu. Structured probabilistic pruning for deep convolutional neural network acceleration. *CoRR*, abs/1709.06994, 2017.

[Weigend *et al.*, 1991] Andreas S. Weigend, David E. Rumelhart, and Barnardo A. Huberman. Generalization by weight elimination with application to forecasting. In *Advances in Neural Information Processing Systems*, pages 875–882, 1991.

[Wen *et al.*, 2015] W. Wen, C. Wu, and Y. Wang. Learning structured sparsity in deep neural networks. In *Advances in Information Processing Systems, NIPS*, pages 2074–2082, Barcelona, Spain, 2015.

[Yuan and Lin, 2006] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society*, 68(1):49–67, 2006.

[Zhang *et al.*, 2016] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10):1943–1955, 2016.

[Zhang *et al.*, 2017] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. 2017.