Structured Probabilistic Pruning for Deep Convolutional Neural Network Acceleration

Huan Wang, Qiming Zhang, Yuchai Wang*, Roland Hu*
Zhejiang Provincial Key Laboratory of Information Processing, Communication and Networking
College of Information Science and Electronic Engineering, Zhejiang University
No. 38 ZheDa Road
Hangzhou 310027, P. R. China

Abstract

Although deep Convolutional Neural Network (CNN) has shown better performance in various machine learning tasks, its application is accompanied by a significant increase in storage and computation. Among CNN simplification techniques, parameter pruning is a promising approach which aims at reducing the number of weights of various layers without intensively reducing the original accuracy. In this paper, we propose a novel progressive parameter pruning method, named Structured Probabilistic Pruning (SPP), which efficiently prunes weights of convolutional layers in a probabilistic manner. Unlike existing deterministic pruning approaches, in which the pruned weights of a well-trained model are permanently eliminated, SPP utilizes the relative importance of weights during training iterations, which makes the pruning procedure more accurate by leveraging the accumulated weight importance. Specifically, we introduce an effective weight competition mechanism to emphasize the important weights and gradually undermine the unimportant ones. Experiments indicate that our proposed method has obtained superior performance on ConvNet and AlexNet compared with existing pruning methods. Our pruned AlexNet achieves 4.0 ∼ 8.9x (averagely 5.8x) layer-wise speedup in convolutional layers with only 1.3% top-5 error increase on the ImageNet-2012 validation dataset. We also prove the effectiveness of our method on transfer learning scenarios using AlexNet.

Introduction

In the area of computer vision, Convolutional Neural Network (CNN) has obtained better performance in classification, detection and segmentation tasks than traditional methods. However, CNN leads to massive computation and storage consumptions, thus hindering its deployment on mobile and embedded devices. Previous research indicated that CNN acceleration falls into four categories: designing compact network architectures, parameter quantization, matrix decomposition and parameter pruning. Our work belongs to the last category.

Normally, pruning may occur non-structured, which deactivates connections between neurons or channels randomly.

*Corresponding Author.

Figure 1: In CNN calculation, a filter shown in blue will be expanded as one row in the weight matrix. Accordingly, pruning a column of weight matrix corresponds to pruning all weights at the same position in different filters, and pruning a row means pruning the corresponding feature map of the next layer generated by the corresponding filter of the previous layer, as shown in green.

Though non-structured pruning is able to achieve high theoretical speedup ratio, the pruned convolutional layers have an irregular shape which is hard to implement on devices because the hardware structure needs to be modified according to the irregularity of pruned parameters. In contrast, structured pruning directly reduces columns or rows of filters (shown in Fig. 1), which can shrink a network into a thinner one and the implementation of the pruned network become easy and efficient (Anwar and Sung 2016; Sze et al. 2017).

Most existing pruning algorithms tend to eliminate parameters once for all by some established criteria, denoted as one-shot pruning. Recently, P. Tyree, and Karras (2017) proposed a progressive pruning scheme, which prunes parameters gradually by using a new Taylor expansion-based criteria. They showed that progressive pruning performs better in accuracy compared with one-shot pruning. However, existing progressive approaches have two problems – Firstly, progressive pruning cost much time for training because the network needs to be retrained during each pruning iteration; Secondly, it is possible that parameters which are less im-
important at the beginning may become important later, but existing pruning methods immediately remove unimportant weights and never recover again.

To solve the above problems, we propose the Structured Probabilistic Pruning (SPP) for CNN acceleration. Our proposed approach incorporates probability into the pruning process. Specifically, we assign a pruning probability to each weight according to some importance criteria. In each iteration, the Monte Carlo sampling is applied to prune weights based on their pruning probabilities. Then the pruned network is retrained by back propagation and the pruning probabilities are recalculated using the retrained weights. The average pruning probability is slightly increased after each iteration so that the training process stops when some fraction of pruning probabilities reaches 1, which means that the corresponding connections are finally eliminated. The proposed algorithm is more dynamic than existing methods because the pruned weights can be revitalized during later iterations, which gives our method the potential to utilize the weight importance information in longer training epochs.

Our approach reports 4× theoretical speedup without performance degeneration by using ConvNet on CIFAR10 (Krizhevsky and Hinton 2009). Our pruned AlexNet achieves 4.0 × 8.9× (averagely 5.8x) layer-wise speedup in convolutional layers with only 1.3% top-5 error increase on the ImageNet-2012 validation dataset (Krizhevsky, Sutskever, and Hinton 2012). We also conduct transfer learning experiments using AlexNet on Flower-102 dataset (Fl), showing that superior performance of SPP in transfer learning tasks compared with existing pruning methods.

**Related Work**

Intensive research has been carried out in CNN acceleration, which is normally categorized into four groups, i.e., designing compact network architectures, parameter quantization, matrix decomposition and parameter pruning.

Compact architecture designing methods use small and compact architectures to replace big and redundant ones. For example, SqueezeNet was proposed to replace 3 × 3 convolution kernels with 1 × 1 counterparts, which decreased the number of parameters 50 times less than the original AlexNet (Iandola, Moskewicz, and Ashraf 2016). GoogLeNet and ResNet also implemented convolution kernel replacement for network acceleration (Szegedy et al. 2015) [He et al. 2016].

Parameter quantization reduces CNN storage by vector quantization in the parameter space. [Han, Mao, and Daily 2015] and [Wu et al. 2016] used vector quantization over parameters to reduce redundancy. [Chen et al. 2015] proposed a hash function to group weights of each CNN layer into hash buckets for parameter sharing. As the extreme form of quantization, binarized networks were proposed to learn binary value of weights or activation functions in CNN training and testing (Courbariaux and Bengio 2016) [Rastegari et al. 2016] [Lin et al. 2016]. Quantization reduces floating computational complexity, but the actual speedup may be very related to hardware implementations.

Matrix decomposition modifies weights into smaller components to reduce computation. [Denton et al. 2014] showed that the weight matrix of a fully-connected layer can be compressed by applying truncated SVD. Tensor train decomposition was proposed and obtained better compression capacity than SVD (Novikov et al. 2014). Several methods based on low-rank decomposition of convolutional kernel tensor were also proposed to accelerate the convolutional layer (Denton et al. 2014) [Jaderberg, Vedaldi, and Zisserman 2014] [Lebedev et al. 2016].

Parameter pruning was pioneered in the early development of neural networks. Optimal Brain Damage leveraged a second-order Taylor expansion to select parameters for deletion, using pruning as regularization to improve training and generalization (LeCun et al. 1989). Deep compression removed close-to-zero connections and quantized the remained weights for further compression (Han, Mao, and Daily 2015). Structured pruning was proposed which tended to prune structured architectures (e.g., rows and columns of weight matrix, or entire feature maps) so that it can accelerate CNN computation without modifying hardware architectures (Anwar and Sung 2016) [Sze et al. 2017]. There are several progressive pruning schemes such as P, Tyree, and Karras (2017), which prunes parameters gradually by using a Taylor expansion-based criteria. However, existing progressive approaches are more time-consuming, and the performance compared with non-progressive ones is rarely exploited.

The main idea of this paper – assigning pruning probabilities to weights and prune by Monte Carlo sampling, is reminiscent of several stochastic methods in machine learning. For example, particle filters used the mutation-selection sampling approach, with a set of particles to represent the posterior distribution of stochastic process given some noisy or partial observations (Liu and Chen 1998). It was successfully used in object tracking, in which the posterior distribution of the current iteration is sampled and updated by a set of states (particles) in previous iterations (Okuma et al. 2004). Another example is simulated annealing, which is a probabilistic technique for approximating the global optimum in a large search space (Granville, Krivanek, and Rason 1994). For each iteration, whether accepting a new solution is sampled from an annealing probability which is calculated by the states of previous iterations. This is very similar to the pruning process of SPP.

**The Proposed Method**

Suppose that the training set D consists of N input vectors and their corresponding target outputs:

\[ D = \{(x_1, y_1), (x_2, y_1), \ldots, (x_N, y_N)\}. \]

The parameters of a CNN with K convolutional layers is represented by

\[ \Omega = \{ (\omega^1, b^1), (\omega^2, b^2), \ldots, (\omega^K, b^K) \}, \]

which are learnt to minimize the discrepancy, i.e., the loss function \( L(D|\Omega) \), between the network outputs and the target outputs. For classification tasks, the target output \( y_i \) is a
one-hot vector in which the \( l \)th element is 1 and others are 0. In this case, the loss function is the negative log-likelihood of softmax output, which is defined as

\[
L(D|\Omega) = -\sum_{i=1}^{N} \log P_{i}
\]

where \( P_{i} \) represents the \( i \)th element of the softmax output of the last layer.

The aim of parameter pruning is to find a simpler network \( \Omega \) with fewer parameters based on the original network \( \Omega \), in which the loss function is minimized under the constraint that the number of parameters are decreased. The minimization problem is defined as

\[
\min_{\Omega'} \left| L(D|\Omega') - L(D|\Omega) \right| \quad \text{s.t. } ||\Omega'||_0 < ||\Omega||_0
\]

Normally for CNN, an input tensor \( z^l \in \mathbb{R}^{W^l \times H^l \times C^l} \) of layer \( l \in \{1, 2, ..., K\} \) is firstly convoluted with the weight tensor \( \omega^l \in \mathbb{R}^{C^{l+1} \times C^l \times W^l \times H^l} \), then a non-linear activation function \( f(\cdot) \), normally Rectified Linear Units (ReLu), will be applied to it. Our proposed SPP introduces a mask \( g \in \{0, 1\} \) for every weight, which determines whether this weight is used for training in the current iteration. Thus, the output of layer \( l \) is described as

\[
f((g^l \odot \omega^l) \ast z^l + b^l),
\]

where \( \odot \) denotes element-wise multiplication and \( \ast \) denotes the convolution operation.

For structured pruning, we assign the same \( g \) to weights in the same column, so weights in each column are pruned or retained simultaneously in each iteration. We choose columns to prune because they are the smallest structured granularity in CNN, which gives us more freedom to select pruning components.

In SPP, \( g \) is obtained by weight importance competition procedure. We sort each column by its \( L_1 \) norm, the bigger the \( L_1 \) value, the more important that column is. It was shown by experiments that \( L_1 \) and \( L_2 \) norms have similar performance as pruning criteria (Han, Pool, and Tran 2015). There are also other importance criteria such as Taylor expansions to guide pruning (P. Tyree, and Karras 2017). In this paper, we choose the \( L_1 \) norm for simplicity. Our method can be easily generalized to other criteria.

In each iteration, columns are competing to survive pruning. For a specific column during the \( i \)th iteration, we associate a pruning probability \( p_i \) based on its rank \( r_i \) of the \( L_1 \) norm. The rank \( r_i \) is from small to large, and normalized within the range of \([0, 1]\). The mask \( g_i \) is obtained by Monte Carlo sampling of \( p_i \) from the following distribution:

\[
P(g_i = 0) = p_i.
\]

Thus, \( g_i \) is more likely to be 0 if \( p_i \) is bigger, and vice versa. Then the CNN is pruned by 5 and back propagation is utilized to train the pruned model for several epochs.

The last step of the \( i \)th iteration is to update the pruning probability. There are two factors to decide whether we need to increase \( p_i \) or not. The first one is the column’s rank. It is more likely to increase \( p_i \) if its \( L_1 \) norm is big, i.e., its rank \( r_i \) small. The second is associated with the iteration number. At the beginning of the training process, we tend to increase \( p_i \) more often. The frequency drops when the training process carries on. Specifically, we use the increment \( \Delta_i \) to represent the possible increase of \( p_i \), which is given as follows:

\[
\Delta_i = \begin{cases} 
A e^{-\alpha r_i}, & \text{if } r_i < R \\
0, & \text{Otherwise}
\end{cases}
\]

The relationship between rank and increment is depicted in Fig. 2. Generally, the bigger the rank, the smaller its increment. Note that we always set the increment to zero when the rank is greater than a threshold \( R \), which is the pruning ratio. Our experiments indicate that it is very important to keep columns with huge \( L_1 \) norms unpruned, thus we need to set their increments to zeros to protect these columns.

Finally, the pruning probability \( p_i \) is updated by simulated annealing. A random variable \( f_i \in \{0, 1\} \) is Monte Carlo sampled from the following distribution:

\[
P(f_i = 1) = \frac{1}{(a + bi)^c}.
\]

where \( a, b \) and \( c \) are positive numbers to be set by experiments. From 6, we can see that the possibility that \( f_i = 1 \) decreases when training iteration \( i \) increases. We finally update \( p_i \) by \( f_i \), which is given as follows:

\[
p_{i+1} = \begin{cases} 
\min(p_i + \Delta_i, 1), & \text{if } f_i = 1 \\
p_i, & \text{if } f_i = 0
\end{cases}
\]

After updating \( p_i \), the whole training procedure continues in another iteration until the pruning ratio \( R \) is obtained, i.e., the ratio that the pruning probability equals to 1 is \( R \). Finally, the pruned network is fine-tuned for several epochs to obtain the final model. The whole procedure of SPP is shown in Algorithm 1.
Algorithm 1 The SPP Algorithm

1: Input the training set \( D \), the original CNN model \( \Omega \) and the pruning ratio \( R \).
2: Train the CNN by back propagation for several epochs to obtain initial parameter settings.
3: For each column, set its pruning probability \( p_0 = 0 \).
4: Set the iteration number \( i = 0 \).
5: repeat
6: For each column, obtain \( g_i \) by Monte Carlo sampling on \( p_i \), as shown in (4).
7: Prune the CNN by (3).
8: Train the pruned CNN by back propagation for several epochs.
9: For each column, obtain its normalized rank \( r_i \).
10: Obtain \( \Delta_i \) by (5), and \( f_i \) by (6).
11: Update \( p_i \) to obtain \( p_{i+1} \) by (7).
12: \( i = i + 1 \).
13: until The ratio that \( p_i = 1 \) equals to \( R \).
14: Prune the weights that their corresponding \( p_i \) equals to 1.
15: Fine-tune the pruned CNN for several epochs.
16: Output the pruned CNN model \( \Omega' \).

Implementation Details

Before implementing SPP, we need to train the CNN for several epochs, as shown in Algorithm 1. The reason is that the parameters at the beginning are randomly initialized, so the ranks of columns are also randomized. We need several rounds of training to reduce the randomness of ranks, which we call it the free-training stage. In our experiments, the ratio between free-training stage and pruning stage is about \( 1/4 \sim 1/3 \). After pruning, we also need several training epochs to fine-tune the model for accuracy compensation. The time of fine-tuning is usually negligible.

The increment to update the pruning probability is set to be a small value to make pruning smooth. In practice, \( A \) is set to 0.05, and \( \alpha \) in (3) is set to \( \frac{\log(10)}{R} \), which makes \( \Delta_i \) at the pruning ratio \( R \) to be 0.14. The parameters \( a \), \( b \) and \( c \) in (6) are set by experiments. One recommended setting which performs well on our experiments is \( a = 40 \), \( b = 0.0008 \) and \( c = 1.2 \).

We also implement several CNN training techniques for our experiments, such as Stochastic Gradient Descent (SGD), regularization, velocity and momentum (Goodfellow, Bengio, and Courville 2016). The details of these techniques are neglected because they are well-established and not associated with the main idea of this paper.

Experimental Results

We evaluate our approach for ConvNet and AlexNet (Krizhevsky, Sutskever, and Hinton 2012) on CIFAR10 (Krizhevsky and Hinton 2009) and ImageNet (Deng et al. 2009), respectively. We also test our approach for transfer learning tasks by using the Oxford Flower Dataset (Nilsback and Zisserman 2008). We use Caffe (Jia et al. 2014) for network evaluation.

Experiments with ConvNet on CIFAR10 Database

We first demonstrate the effectiveness of our method on the CIFAR10 database, which contains 10 classes with 50,000 images for training and 10,000 for testing. We take 5,000 images from the training set as the validation set. ConvNet was firstly proposed by Krizhevsky, Sutskever, and Hinton (2012) for classification on CIFAR10, which was composed of 3 convolutional layers and 1 fully connected layer. The baseline classification accuracy for ConvNet on CIFAR10 is 81.5%.

The batch size is set to 256 for training. We firstly train the model freely for about 150 epochs before SPP is imposed. The target pruning ratio \( R \) is set to 0.75, which means that we need to prune 75% parameters, or the theoretical speedup is 4 times faster than baseline. When the target pruning ratio is reached after about 500 epochs, we use the validation set to fine-tune the model for another 100 epochs until obtaining the best accuracy.

Methods for comparison includes three structured pruning approaches, namely, Structured Sparsity Learning (Wen, Wu, and Wang 2015), Taylor Pruning (P. Tyree, and Karras 2017) and Filter Pruning (Li et al. 2017). SSL was implemented through open source codes provided by the authors, and we implemented the other two (TP and FP) ourselves.

For our proposed SPP method, we noticed that the settings of initial learning rate is an important factor to influence convergence and accuracy of the pruned model. Thus, we set two learning rates, i.e., \( LR = 0.001 \) and \( LR = 0.002 \) and obtain two variants of SPP.

The performance of these methods is shown in Table 1. The column (row) sparsity is calculated by the ratio between the number of all-zero columns (rows) and the total number of columns (rows) after pruning. In Table 1, we list the column and row sparsity of the three convolutional layers sequentially. Because SPP prunes the model column-wisely, its row sparsity is relatively small.

The average speedup is calculated by the ratio between GFLOps of the pruned model and that of the baseline. It is an indicator of how faster the pruned CNN would work than its original counterpart. The last column of Table 1 lists the training epochs of each method. The baseline model is trained with 700 epochs. Because SSL, TP and FP tend to prune a well-trained model, we list their pruning and fine-tune epochs right after the 700 epochs for training the baseline. However, because SPP merges training and pruning into a coherent process, the training epochs have been split into 150 for pretraining, 500 for SPP, and 100 for fine-tune.

It can be observed that our proposed SPP is better than other three methods. SPP has achieved comparable speedup with other three, but it requires shorter training time and obtains much better classification accuracy.

Experiments with AlexNet on ImageNet

We also verify our method on ImageNet, which is a large dataset of 1,000 classes containing 1.28M images for training, 100,000 for testing and 50,000 for validation. The network to be pruned is AlexNet, which was proposed by Krizhevsky, Sutskever, and Hinton (2012). It was composed
of 5 convolutional layers and 3 fully connected layers. Here we adopt an open source re-implementation of AlexNet on Caffe as the baseline.

The baseline is trained from scratch for 82 epochs and obtains 73.0% top-5 accuracy on the ImageNet-2012 validation dataset, with initial learning rate 0.01 and batch size 256. For SPP, the model is firstly pre-trained for 10 epochs; then pruning is applied for another 30 epochs; and finally we fine-tune the model for another 6 epochs to regain accuracy. SSL, TP and FP are applied on the well-trained baseline model with a smaller learning rate of 0.001.

The performance of these methods is shown in Table 2. Under the same 0.75 pruning ratio settings, our method achieves layer-wise $4.0 \times 8.6 \times$ speedup with only 1.3% loss of top-5 accuracy. Weighted by GFLOPs of five convolutional layers, the average speedup is $5.8 \times$. It can be seen that our proposed SPP is superior in speed, accuracy and training time compared to other three methods.

It needs to be noted that the accuracy increases from 0.730 to 0.741 after SPP pruning, which indicates that our proposed method can capture important components of the baseline network. The phenomenon that accuracy increases after pruning is also reported by other research works (Han, Mao, and Dally 2015; Wen, Wu, and Wang 2015).

| Method   | Row Sparsity | Column Sparsity | Average Speedup | Accuracy | Training Epochs |
|----------|--------------|-----------------|-----------------|----------|-----------------|
| Baseline | 0-0-0        | 0-0-0           | 1×              | 0.815    | 700             |
| SSL      | 0.50-0.44-0  | 0.07-0.72-0.50  | 3.2×            | 0.765    | 700+1000+100    |
| TP       | 0.75-0.50-0.50 | 0-0.75-0.50    | 5.5×            | 0.751    | 700+300+100     |
| FP       | 0.75-0.50-0.50 | 0-0.75-0.50    | 5.5×            | 0.758    | 700+300+100     |
| SPP ($LR = 0.001$) | 0.125-0-0 | 0.76-0.75-0.75 | 4.1×            | 0.815    | 150+500+100     |
| SPP ($LR = 0.002$) | 0.156-0-0 | 0.80-0.80-0.813 | 5.3×            | 0.812    | 150+500+100     |

Table 1: Accelerating the ConvNet model using SSL, TP, FP and SPP. The best performance of each column is shown in bold.

Transfer Learning

We apply SPP to transfer learning in which a well-trained model is fine-tuned by data from other knowledge domains. We use AlexNet as the pre-trained model, and use the 102-class flower images in the Oxford Flower Dataset to fine-tune the model (Nilsback and Zisserman 2008). The dataset is composed of 8,149 images. We use 1,020 for training, 1,020 for validation and the other 6,149 for testing. Under the goal of obtaining $4 \times$ speedup, we compare the performance of SPP with TP. The latter is reported effective for transfer learning on the Flower dataset (P. Tyree, and Karras 2017).

Actually, there are two approaches for transfer learning based model compression. The first one is to fine-tune the pre-trained model using the data from other knowledge domains, then compress the fine-tuned model to a simplified one; the other approach is to compress the pre-trained model first and then fine-tune the compressed model. We denote the two approaches as Approach 1 (fine-tune then compress) and Approach 2 (compress then fine-tune), respectively.

Table 3 shows the results of TP and SPP under these two approaches. Our proposed SPP performs better than TP in both approaches. Especially, our SPP shows great advantages in accuracy for Approach 2 (compress then fine-tune). One possible reason to account for this is that during the SPP process, we do not lose the classification information contained in the pre-trained model, while TP does not retain this information during pruning.

Conclusions

We proposed the Structured Probabilistic Pruning for CNN acceleration. It is a progressive pruning method which incorporates model training and pruning together. Our approach has shown a better balance among pruning ratio, speedup, accuracy and training time. In the future, we plan to implement our method to much deeper CNNs such as VGG and ResNet.

Acknowledgements

This work is supported by the Natural Key R&D Prora of China (Grant No. 2017YFB1002400), Natural Science Foundation of Zhejiang Province (Grant No. LY16F010004) and Zhejiang Public Welfare Research Program (Grant No. 2016C31062).

References

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

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

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

[Deng et al. 2009] Deng, J.; Dong, W.; Socher, R.; Li, L. J.; Li, K.; and Feifei, L. 2009. Imagenet: A large-scale hierarchical image database. In Proceedings of the International Conference on Computer Vision and Pattern Recognition, CVPR’09, 248–255.

[Denton et al. 2014] Denton, E.; Zaremba, W.; Bruna, J.; LeCun, Y.; and Fergus, R. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In Advances in Neural Information Processing Systems, NIPS.
### Table 2: Accelerating the AlexNet model using SSL, TP, FP and SPP. The best performance of each column is shown in **bold**.

| Method | Validation Accuracy | Test Accuracy |
|--------|---------------------|---------------|
| Baseline | 0.773 | 0.735 |
| TP (Approach 1) | 0.710 | 0.672 |
| SPP (Approach 1) | 0.740 | 0.709 |
| TP (Approach 2) | 0.498 | 0.443 |
| SPP (Approach 2) | 0.817 | 0.779 |

Table 3: Comparing the transfer learning task for SPP and TP methods.

| Method | Row Sparsity | Column Sparsity | Average Speedup | Accuracy | Training Epochs |
|--------|--------------|-----------------|-----------------|----------|-----------------|
| Baseline | 0-0-0-0 | 0-0-0-0 | 1× | 0.730 | 82 |
| SSL | 0.37-0.51-0.63-0.57-0.50 | 0-0.17-0.70-0.55-0.37 | 3.0× | 0.634 | 82+44+6 |
| TP | **0.75-0.75-0.75-0.75-0.75** | 0-0-0-0-0 | 4.0× | 0.601 | 82+20+6 |
| FP | **0.75-0.75-0.75-0.75-0.75** | 0-0-0-0-0 | 4.0× | 0.600 | 82+20+6 |
| SPP | 0-0.55-0.38-0.18-0 | **0.76-0.75-0.75-0.75-0.75** | **5.8×** | **0.741** | **10+30+6** |

### References

- [Jia et al. 2014](http://www.robots.ox.ac.uk/~vgg/)
- [Krizhevsky and Hinton 2009](http://www.robots.ox.ac.uk/~vgg/)
- [Lebedev et al. 2016](http://www.robots.ox.ac.uk/~vgg/)
- [Lin et al. 2016](http://www.robots.ox.ac.uk/~vgg/)
- [LeCun et al. 1989](http://www.robots.ox.ac.uk/~vgg/)
- [Okuma et al. 2004](http://www.robots.ox.ac.uk/~vgg/)
- [Jaderberg, Vedaldi, and Zisserman 2014](http://www.robots.ox.ac.uk/~vgg/)
- [Lebedev et al. 2016](http://www.robots.ox.ac.uk/~vgg/)
- [Lin et al. 2016](http://www.robots.ox.ac.uk/~vgg/)
- [Krizhevsky, Sutskever, and Hinton 2012](http://www.robots.ox.ac.uk/~vgg/)
- [He et al. 2016](http://www.robots.ox.ac.uk/~vgg/)
- [Granville, Krivanek, and Rasson 1994](http://www.robots.ox.ac.uk/~vgg/)
- [Jaderberg, Vedaldi, and Zisserman 2014](http://www.robots.ox.ac.uk/~vgg/)
- [Lin et al. 2016](http://www.robots.ox.ac.uk/~vgg/)

[Granville, Krivanek, and Rasson 1994](http://www.robots.ox.ac.uk/~vgg/): Granville, V.; Krivanek, M.; and Rasson, J. P. 1994. Simulated annealing: A proof of convergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16(6):652–656.

[Han, Mao, and Dally 2015](http://www.robots.ox.ac.uk/~vgg/): 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.

[Han, Pool, and Tran 2015](http://www.robots.ox.ac.uk/~vgg/): Han, S.; Pool, J.; and Tran, J. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, NIPS, 1135–1143.

[He et al. 2016](http://www.robots.ox.ac.uk/~vgg/): He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition, CVPR*, 770 – 778.

[Iandola, Moskewicz, and Ashraf 2016](http://www.robots.ox.ac.uk/~vgg/): Iandola, F.; Moskewicz, M.; and Ashraf, K. 2016. SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and<0.5MB model size. *arXiv preprint* arXiv:1602.07360.

[Jaderberg, Vedaldi, and Zisserman 2014](http://www.robots.ox.ac.uk/~vgg/): Jaderberg, M.; Vedaldi, A.; and Zisserman, A. 2014. Speeding up convolutional neural networks with low rank expansions. *Computer Science* 4(4):1–13.

[Jia et al. 2014](http://www.robots.ox.ac.uk/~vgg/): Jia, Y.; Shelhamer, E.; Donahue, J.; Karayev, S.; Long, J.; Girshick, R.; Guadarrama, S.; and Darrel, T. 2014. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint* arXiv:1408.5093.

[Krizhevsky and Hinton 2009](http://www.robots.ox.ac.uk/~vgg/): Krizhevsky, A., and Hinton, G. E. 2009. Learning multiple layers of features from tiny images.

[Krizhevsky, Sutskever, and Hinton 2012](http://www.robots.ox.ac.uk/~vgg/): Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Information Processing Systems, NIPS*, 1097–1105.

[Lebedev et al. 2016](http://www.robots.ox.ac.uk/~vgg/): Lebedev, V.; Ganin, Y.; Rakhuba, M.; Oseledets, I.; and Lempitsky, V. 2016. Speeding-up convolutional neural networks using fine-tuned CP-decomposition. *arXiv preprint* arXiv:1510.03009.

[LeCun et al. 1989](http://www.robots.ox.ac.uk/~vgg/): LeCun, Y.; Denker, J.; Solla, S.; Howard, R.; and Jackel, L. 1989. Optimal brain damage. In *Advances in Neural Information Processing Systems, NIPS*.

[Li et al. 2017](http://www.robots.ox.ac.uk/~vgg/): Li, H.; Kadow, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2017. Pruning filters for efficient convnets. In *International Conference on Learning Representations, ICLR*.

[Lin et al. 2016](http://www.robots.ox.ac.uk/~vgg/): Lin, Z.;bourbia, M.; Memisevic, R.; and Bengio, Y. 2016. Neural networks with few multiplications. *arXiv preprint* arXiv:1510.03009.

[Liu and Chen 1998](http://www.robots.ox.ac.uk/~vgg/): Liu, J. S., and Chen, R. 1998. Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association* 93(443):1032–1044.

[Nilsback and Zisserman 2008](http://www.robots.ox.ac.uk/~vgg/): Nilsback, M. E., and Zisserman, A. 2008. Automated flower classification over a large number of classes. In *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing, ICGVIP* ’08.

[Novikov et al. 2014](http://www.robots.ox.ac.uk/~vgg/): Novikov, A.; Podoprikhin, D.; Osokin, A.; and Vetrov, D. 2014. Tensorizing neural networks. *arXiv preprint* arXiv:1509.06569.

[Okuma et al. 2004](http://www.robots.ox.ac.uk/~vgg/): Okuma, K.; Taleghani, A.; Freitas, N.; Little, J.; and Lowe, D. G. 2004. A boosted particle filter: Multitarget detection and tracking. In *Proceedings of the European Conference on Computer Vision, ECCV*, 28–39.

[P, Tyree, and Karras 2017](http://www.robots.ox.ac.uk/~vgg/): P, P. M.; Tyree, S.; and Karras, T. 2017. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations, ICLR*.

[Rastegari et al. 2016](http://www.robots.ox.ac.uk/~vgg/): 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, ECCV*, 525–542.

[Sze et al. 2017](http://www.robots.ox.ac.uk/~vgg/): Sze, V.; Chen, Y. H.; Yang, T. J.; and Emer, J. 2017. Efficient processing of deep neural networks: A tutorial and survey. *arXiv preprint* arXiv:1703.09039.
[Szegedy et al. 2015] Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition, CVPR*, 1–9.

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

[Wu et al. 2016] Wu, J.; Leng, C.; Wang, Y.; Hu, Q.; and Cheng, J. 2016. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 4820 – 4828.