https://doi.org/10.2352/ISSN.2470-1173.2017.6.MOBMU-306

Prune the Convolutional Neural Networks with Sparse Shrink

Xin Li, Changsong Liu
State Key Laboratory of Intelligent Technology and Systems,
Tsinghua National Laboratory for Information Science and Technology,
Department of Electronic Engineering, Tsinghua University, Beijing 100084, China
Email: {***lixin08, lcs***}***@ocrserv.ee.tsinghua.edu.cn***

Abstract

Nowadays, it is still difficult to adapt Convolutional Neural Network (CNN) based models for deployment on embedded devices. The heavy computation and large memory footprint of CNN models become the main burden in real application. In this paper, we propose a "Sparse Shrink" algorithm to prune an existing CNN model. By analyzing the importance of each channel via sparse reconstruction, the algorithm is able to prune redundant feature maps accordingly. The resulting pruned model thus directly saves computational resource. We have evaluated our algorithm on CIFAR-100. As shown in our experiments, we can reduce 56.77% parameters and 73.84% multiplication in total with only minor decrease in accuracy. These results have demonstrated the effectiveness of our "Sparse Shrink" algorithm.

Introduction

In recent years, great progress has been achieved in computer vision which is arguably attributed to greater computation resources and the application of deep learning algorithms [18, 16, 11, 15]. The convolutional neural networks (CNN) is a popular example of deep learning algorithms. It adopts a deep architecture that consist of many stacked convolutional and fully-connected layers, which is specifically designed for solving computer vision related problems. Although CNN has bring breakthrough into computer vision, we are still not possible to decide the optimum network architecture, *e.g.* number of channels in convolutional layer, for a specific task. Nowadays, people tend to design large networks with large number of channels to build a high-capacity model. However, this brings a large demand on computation and memory capacity, which are especially limited on embedded devices. The heavy computation and large memory footprint of CNN models become the major burden in real application.

On the other hand, it is observed that there is redundancy in large networks [4, 20]. Convolutional layers occupy the main calculation in CNN, and the responses of their resulting feature maps are sometimes largely correlated to each other. Therefore, it is intuitive to prune a large pre-trained model by removing redundant connections. This will results in a lightweight network with comparable level of performance and less demand on both memory and computational complexity.

Motivated by this, we propose a novel "Sparse Shrink" algorithm to prune a CNN model: we evaluate the importance of each channel of feature maps, and prune less important channels to get a slimmer network. The pruned model is of a similar performance with original model, yet thinner structure and lower computational complexity.

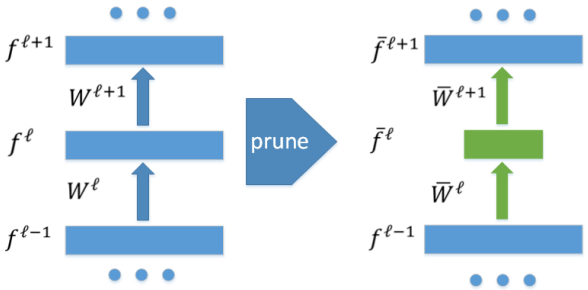

Figure 1. *By evaluating the importance of each channel, "Sparse Shrink" prunes less important channels and builds a slimmer model. Weights in the upper and lower layer are modified accordingly.*

Related Work

Extensive work have been done to accelerate the testing of CNN models or lower its memory cost. Some [6, 22] of them speed up the testing by explore the sparsity in CNN models with low rank decomposition. Vasilache [19] speed up the convolution operation by a Fast Fourier Transform implementation. However, these algorithms focus on either accelerating test speed or lower memory footprint of CNN models without changing their model structures.

Network pruning has been studied by several researchers [9, 5, 17, 14] . Lecun *et al.* [9] and Hassibi *et al.* [5] show that a portion of weights can be set to zero by analyzing their value and Hessian matrix. Han *et al.* [4, 3] gradually prune the small-weights in a network, and further reduce storage requirement by compressing weights in fully connected layer with matrix factorization and vector quantization. Rastegari *et al.* [14] binarize both the weights and layer inputs, such that the resulting network mainly uses XNOR operations. Stepniewski *et al.* [17] prunes network with genetic algorithm and simulated annealing. However, these algorithms only makes use of intra-kernel sparsity, without doing channel wise pruning. This limits GPUs to explit computational savings. Different from existing algorithms, our "Sparse Shrink" algorithm directly prune network structure in convolutional layer by channel wise pruning. The most related work on channel wise pruning would be "Structured pruning" [1]. It naively remove the incoming and outgoing weights of a pruned channel. In contrast, we modify convolutional kernel in the upper layer by reconstructing original feature maps in order to reduce decrease in accuracy.

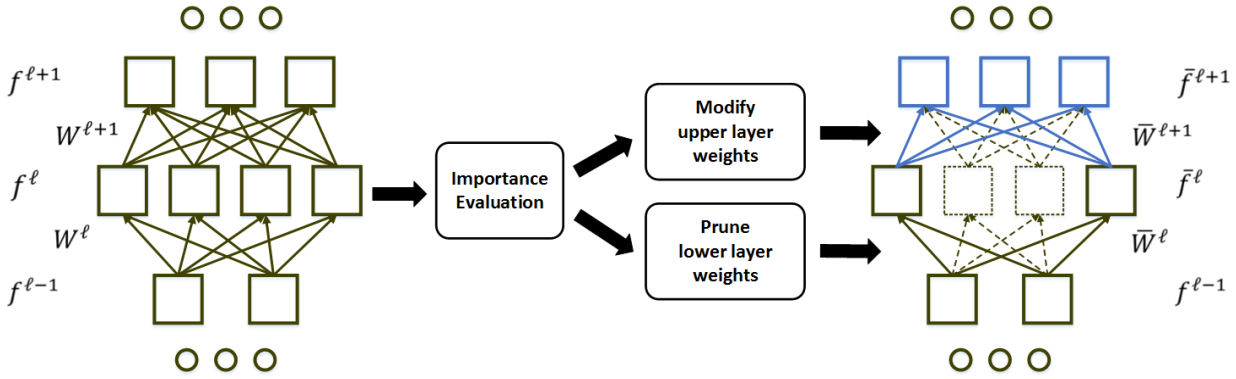

Figure 2. *Illustration of "Sparse Shrink" algorithm. We evaluate the importance factor of each channel of feature maps f^{ℓ}, and prune the least important channels (dashed box). The pruning operation involves removing corresponding channels in W^{ℓ} (dashed line), and modifying convolutional kernel $\overline{W}^{\ell}$ (blue line).*

Sparse Shrink

In this section, we elaborate how our "Sparse Shrink" algorithm prune an existing network by channel-level pruning in convolutional layer. The basic idea of "Sparse Shrink" is intuitive: there exists redundancy in convolutional layers, and we can remove redundant channels to produce a pruned model with minimum loss in accuracy. There are three major steps in our algorithm. Firstly, we evaluate the importance of each channel with "Sparse Reconstruction" algorithm. Secondly, those redundant, *i.e.* less important channels, are removed, and related convolutonal kernels are modified, as shown in Figure 2. This results in a pruned model with a minor decrease in accuracy. Finally, the pruned model is re-trained to achieve its best performance.

Importance Evaluation

Sparse reconstruction [2, 12, 13] is a well-studied problem which focus on finding representative data points, such that each data point in the dataset can be described as a linear combination of a set of representative points. Formally, with a data matrix $D \in \mathbb{R}^{m\times N}$, *i.e.* N data points of a dataset in $\mathbb{R}^m$, the standard ℓ_1 relaxation of the optimization problem can be written as

$$min \|D - DU\|_F^2, s.t. \|U\|_{1,q} \leq \tau, \mathbf{1}^\top \mathbf{U} = \mathbf{1}^\top \tag{1}$$

where $U \in \mathbb{R}^{N\times N}$ is corresponding reconstruction coefficient matrix and $\|U\|_{1,q} \triangleq \sum_{i=1}^{N} \left\|u^i\right\|_q$ is the sum of the ℓ_q norms of the rows of U. We choose $q = 2$ so that the optimization program is convex and $\tau > 0$ is an appropriately chosen parameter. $\mathbf{1}^\top \mathbf{U} = \mathbf{1}^\top$ is a affine constraint to make the representatives be invariant with respect to a global translation of the data.

Now we elaborate how to make use of sparse reconstruction to evaluate the importance of each channel in a convolutional layer. Throughout this paper, we use the following notations for the simplicity of explanation. Let f^{ℓ} denote the output feature maps for the ℓ-th layer and f_i^{ℓ} denote the value of the i-th channel. The feature maps has a dimension of $C_\ell \times H \times W$, where C_ℓ is the number of channels in layer ℓ, and $H \times W$ is the corresponding spatial size. To evaluate the importance of each channel in feature maps f^{ℓ}, we randomly select N input image, and get a data matrix $D^{N\times C_\ell \times H\times W}$. In contrast to standard sparse reconstruction algorithm as Equation (1), which focus on finding representative data points among N total data points, our algorithm aims at finding representative channels among the C_ℓ channels. Therefore we reshape the data matrix into $D^{(N\times H\times W)\times C_\ell}$, and regard each channel c_i as a "data point" in $\mathbb{R}^{N\times H\times W}$. With this representation, we are able to find the most representative channels by reconstructing data matrix D.

More specifically, we use the entire data matrix as dictionary and try to reconstruct the data matrix with reconstruction coefficients $U \in \mathbb{R}^{C_\ell \times C_\ell}$.

$$\begin{bmatrix} d_1 & d_2 & \dots & d_{C_\ell} \end{bmatrix} \approx \begin{bmatrix} d_1 & d_2 & \dots & d_{C_\ell} \end{bmatrix} \begin{bmatrix} u^1 \\ u^2 \\ \dots \\ u^{C_\ell} \end{bmatrix}$$

Then we solve the optimization problem in Equation (1) to get the reconstruction coefficients U.

The regularization term $\|U\|_{1,2} \triangleq \sum_{i=1}^{C_\ell} \left\|u^i\right\|_2$ in Equation (1) provides information about relative importance between channels. A more representative channel takes larger part in reconstruction, and thus the corresponding reconstruction coefficients have more non-zeros elements with larger values. Hence, the resulting coefficients can be intuitively utilized to rank importance of each channel, and to evaluate feature maps redundancy. More precisely, we rank a channel i by its importance factor $\left\|u^i\right\|_2$, where $u^i \in \mathbb{R}^{1\times C_\ell}$ indicates the i-th row of reconstruction matrix U. The lower importance factor is, the more redundant the channel become. Therefore, we prune these bottom-ranking channels to get a slimmer network.

Network Pruning

Once we rank the importance factors, we can prune the network in layer ℓ, by removing the least important K channels. This involves two specific modifications in network weights, removing channels in layer ℓ and reconstructing feature maps in layer $\ell+1$.

As illustrated in Figure 2, the feature maps f^{ℓ} are obtained by convolving $f^{\ell-1}$ with kernel $W^{\ell} \in \mathbb{R}^{C_\ell \times C_{\ell-1}\times k\times k}$, where k is the spatial size of convolutional kernel. To remove a channel c_i in f^{ℓ}, we only need to remove corresponding "Slice" in W^{ℓ}, *i.e.* $W_{c_i}^{\ell} \in \mathbb{R}^{C_{\ell-1}\times k\times k}$. Having pruned K least important feature maps, the new pruned convolutional kernel $\overline{W}^{\ell} \in \mathbb{R}^{(C_\ell - K)\times C_{\ell-1}\times k\times k}$ has a channel number $C_\ell - K$. And the new feature maps $\overline{f^{\ell}} \in \mathbb{R}^{(C_\ell - K)\times C_{\ell-1}\times H\times W}$ is obtained by convolving $\overline{W}^{\ell}$ with f^{ℓ}.

Pruning layer ℓ will obviously affect layer $\ell+1$. Instead of naively removing corresponding channels in $W^{\ell+1}$, we manage to

get a new convolutional kernel by reconstructing the original feature maps f^ℓ, in order to minimize the decrease in accuracy after pruning. Given a data matrix $\overline{D} \in \mathbb{R}^{(C_\ell - K)\times(N\times H\times W)}$ of pruned feature maps $\overline{f^\ell}$, we try to reconstruct original f^ℓ data matrix by minimizing reconstruction error,

$$\min Err = \min_V \left\| D - \overline{D}V \right\| \quad (2)$$

Where $V \in \mathbb{R}^{(C_\ell - K)\times C_\ell}$ is the reconstruction coefficients. We can obtain a closed-form solution for Equation (2),

$$V = \left(\overline{D}^\top \overline{D}\right)^{-1} \overline{D}D \quad (3)$$

Let $\widehat{V} \in \mathbb{R}^{C_\ell \times (C_\ell - K)\times 1\times 1}$ denote the 1×1 convolutional kernel derived from V, where $\widehat{V}_{i,j,1,1} \triangleq V_{j,i}$. The reconstructed feature maps $\widehat{f^\ell}$ is obtained with,

$$\widehat{f^\ell} = \widehat{V} * \overline{f^\ell}$$

And the feature maps $\overline{f}^{\ell+1}$ in the pruned network can thus be written as,

$$\begin{aligned} \overline{f}^{\ell+1} &= ReLU\left(W^{\ell+1} * \widehat{f^\ell}\right) \\ &= ReLU\left(W^{\ell+1} * \left(\widehat{V} * \overline{f^\ell}\right)\right) \\ &= ReLU\left(\left(W^{\ell+1} * \widehat{V}\right) * \overline{f^\ell}\right) \end{aligned}$$

And the new convolution kernel $\overline{W}^{\ell+1} \in \mathbb{R}^{C_{\ell+1}\times(C_\ell - K)}$ is,

$$\begin{aligned} \overline{W}^{\ell+1} &= W^{\ell+1} * \widehat{V} \\ &= W^{\ell+1} V^\top \end{aligned} \quad (4)$$

Now we get a pruned network with $C_\ell - K$ channels in layer ℓ, and pruned convolution kernels $\overline{W}^\ell$, $\overline{W}^{\ell+1}$. The newly pruned model may perform better after further training for more iterations.

Experiment

We evaluated the performance of "Sparse Shrink" algorithm on the benchmark dataset CIFAR-100 [8]. CIFAR-100 is a widely used benchmark dataset for image classification with 60,000 color images of 100 categories in total. This size of images is 32×32. Images are split into 50,000 training set and 10,000 test set. Following NIN [10] we use global contrast normalization and ZCA whitening as pre-processing. We use NIN [10] model as a baseline model, which has been proven to be a successful CNN structure on CIFAR-100. There are three convolutional layers in the NIN model, *i.e.* *Conv*1,*Conv*2,*Conv*3, with 192 channels in each of them. In this paper we focus on pruning these three convolutional layers to obtain slimmer networks. We employ Caffe [7] implementation as our experiment platform. Throughout the experiments, we fix the initial learning rate to 0.01 and the weight decay coefficient to 0.001. The code and models is released at: https://github.com/lixincn2015.

We conduct three sets of experiments to evaluate our algorithm. In the first experiment, we apply "Sparse Shrink" algorithm to each of the three convolutional layers separately. And the sorted importance factors of each layer are shown in Figure 3. As shown in Figure 3, there are some channels with obviously larger importance in all three convolutional layers, while others have relatively smaller ones. Pruning those channels with smaller importance factors is supposed to result in less decrease in performance.

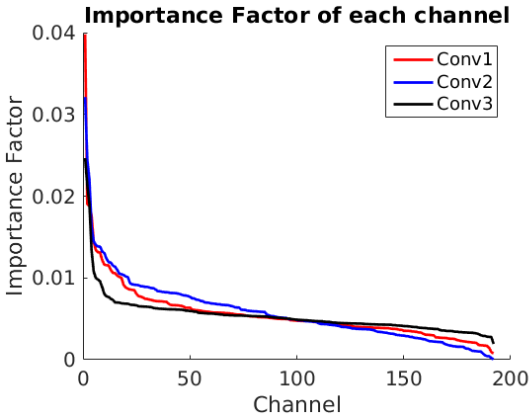

Figure 3. *Importance factors of each channel in the three convolutional layer.*

Table 1. Comparison of accuracies (%) of the pruned models on CIFAR-100 test set.

Pruned Channels	0	64	96	128	160	176
Conv1	68.08	67.80	67.86	67.86	67.36	67.38
Conv2	68.08	67.51	67.36	66.98	65.95	64.67
Conv3	68.08	67.68	67.00	66.07	65.09	61.17

By pruning different number of channels according to importance factors, we get corresponding pruned models and then evaluate these models on CIFAR-100 test set. Detailed result is shown in Table 1 , where *Conv*1,*Conv*2,*Conv*3 are three convolutional layers from the bottom up. The baseline NIN model, *i.e.* not pruning any channels on any layer, has an accuracy of 68.08%. As shown in Table 1 , with a decrease of $\sim 1\%$ in accuracy, we can prune as many as 176, 128, and 96 channels on three convolutional layers respectively (highlighted in blue). It is worth mentioning that pruning 176 channels on *Conv*1 layer brings only minor decrease of 0.7% in accuracy. We attribute this to the effectiveness of our "Sparse Shrink" algorithm, which can dramatically reduce redundancy in feature maps while preserving important information.

Pruning any one of three convolutional layer results in decreased performance, wheres the decrease show different features. Pruning lower layers brings less accuracy decrease. More specifically, with the same level of decrease in accuracy (highlighted in blue), we can prune much more channels in *Conv*1 than *Conv*3 (176 vs 96). It indicates that there is more redundancy in the lower layers of NIN model than in the upper layers, and *Conv*1 needs much less feature maps than *Conv*3. This finding is consistent with previous studies [21, 16]. It's well observed that there is a hierarchical nature of the features in deep networks. Feature maps of lower layers mostly responds to low-level visual features, *e.g.* edges or corners, which can be shared between high-level

Table 2. Comparison of number of parameters and multiplication between pruned model and baseline model.

Layer	Input Size	Number of Parameters			Number of Multiplications		
		Baseline Model	pruned model	Reduction (%)	Baseline Model	pruned model	Reduction (%)
Conv1	32×32	$192 \times 3 \times 5 \times 5$	$16 \times 3 \times 5 \times 5$	91.67	1.47×10^7	1.23×10^6	91.67
Cccp1	32×32	$160 \times 192 \times 1 \times 1$	$160 \times 16 \times 1 \times 1$	91.67	3.15×10^7	2.62×10^6	91.67
Cccp2	32×32	$96 \times 160 \times 1 \times 1$	$96 \times 160 \times 1 \times 1$	0	1.57×10^7	1.57×10^7	0
Conv2	16×16	$192 \times 96 \times 5 \times 5$	$64 \times 96 \times 5 \times 5$	66.67	1.18×10^8	3.93×10^7	66.67
Cccp3	16×16	$192 \times 192 \times 1 \times 1$	$192 \times 64 \times 1 \times 1$	66.67	9.44×10^6	3.15×10^6	66.67
Cccp4	16×16	$192 \times 192 \times 1 \times 1$	$192 \times 192 \times 1 \times 1$	0	9.44×10^6	9.44×10^6	0
Conv3	8×8	$192 \times 192 \times 3 \times 3$	$96 \times 192 \times 3 \times 3$	50.00	2.12×10^7	1.06×10^7	50.00
Cccp5	8×8	$192 \times 192 \times 1 \times 1$	$192 \times 96 \times 1 \times 1$	50.00	2.36×10^6	1.18×10^6	50.00
Cccp6	8×8	$100 \times 192 \times 1 \times 1$	$100 \times 192 \times 1 \times 1$	0	1.23×10^6	1.23×10^6	0
Overall	-	9.83×10^5	4.25×10^5	56.77	3.23×10^8	8.45×10^7	73.84

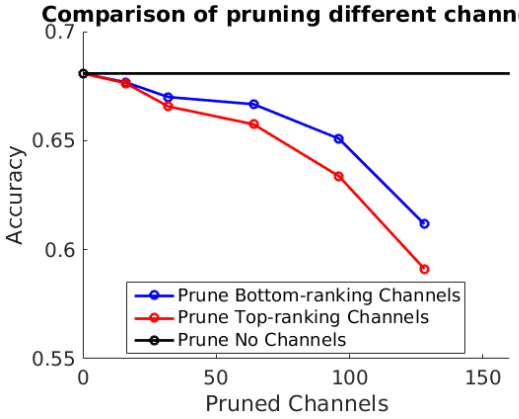

Figure 4. *Comparison of pruning top-ranking and bottom-ranking channels in Conv3.*

patterns. Upper layers then assemble the low-level features to exponentially more complex visual patterns. Hence we need a lot more channels in upper layers than in lower layers.

In the second experiments, we compare the accuracy of pruning different channels in *Conv*3 layer. More specifically, we prune top-ranking and bottom-ranking channels according to importance factors, and evaluate the pruned models on test set. As shown in Figure 4, pruning both top-ranking and bottom-ranking channels results in decrease in accuracy. However, pruning bottom-ranking channels brings less decrease. As the number of pruned channels increases, the gap becomes larger. And pruning 128 bottom-ranking channels has an advantage of 2% over pruning top-ranking channels (61.17% vs 59.12%) . This validates that our "Sparse Shrink" algorithm is able to successfully evaluate the importance of each channel, and hence keep the most important feature maps during pruning.

Finally, in the third experiment, we further prune all the three convolutional layers in the network from the bottom up, and remove 176, 128, and 96 channels in *Conv*1, *Conv*2, *Conv*3 respectively. The final pruned model has an accuracy of 65.53% on test set. Table 2 provides a detailed comparison between baseline model and the pruned model in terms of number of parameters and number of multiplication. For a convolutional kernel $W^\ell \in \mathbb{R}^{C_\ell \times C_{\ell-1} \times k \times k}$ in layer ℓ, the corresponding number of parameter is $C_\ell \times C_{\ell-1} \times k \times k$. And the number of multiplication in layer ℓ is $C_\ell \times C_{\ell-1} \times k \times k \times H \times W$, where H and W are the input size of layer ℓ. Compared to the baseline model, this pruned model reduces 56.77% parameters and 73.84% multiplication, at a minor decrease of 2.55% in accuracy. This validates that our "Sparse Shrink" algorithm is able to save computational resource of a well-trained model without serious performance degradation.

Conclusion

In this paper, we propose a "Sparse Shrink" algorithm for convolutional neural network pruning. The Sparse Shrink algorithm evaluates the importance of each channel by sparse reconstruction. Channels with smaller importance factors is considered to be more redundant, and is pruned to get a slimmer network. New convolutional kernels can be derived from reconstructing original feature maps. Experiments on CIFAR-100 dataset show that the "Sparse Shrink" algorithm is able to significantly save computational resource with only minor decrease in performance.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant No. 61471214 and the National Basic Research Program of China (973 program) under Grant No. 2013CB329403 .

Reference

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

[2] Ehsan Elhamifar, Guillermo Sapiro, and Rene Vidal. See all by looking at a few: Sparse modeling for finding representative objects. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1600–1607. IEEE, 2012.

[3] Song Han, Huizi Mao, and William J Dally. A deep neural network compression pipeline: Pruning, quantization, huffman encoding. *arXiv preprint arXiv:1510.00149*, 2015.

[4] Song Han, Jeff Pool, John Tran, and William Dally. Learn-

ing both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

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

[6] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference. BMVA Press*, 2014.

[7] 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 *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[8] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[9] Yann LeCun, John S Denker, Sara A Solla, Richard E Howard, and Lawrence D Jackel. Optimal brain damage. In *NIPs*, volume 2, pages 598–605, 1989.

[10] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[11] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[12] Julien Mairal, Francis Bach, Jean Ponce, Guillermo Sapiro, and Andrew Zisserman. Discriminative learned dictionaries for local image analysis. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[13] Ignacio Ramirez, Pablo Sprechmann, and Guillermo Sapiro. Classification and clustering via dictionary learning with structured incoherence and shared features. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3501–3508. IEEE, 2010.

[14] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *arXiv preprint arXiv:1603.05279*, 2016.

[15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

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

[17] Slawomir W Stepniewski and Andy J Keane. Pruning backpropagation neural networks using modern stochastic optimisation techniques. *Neural Computing & Applications*, 5(2):76–98, 1997.

[18] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[19] Nicolas Vasilache, Jeff Johnson, Michael Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. Fast convolutional nets with fbfft: A gpu performance evaluation. *arXiv preprint arXiv:1412.7580*, 2014.

[20] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[21] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer, 2014.

[22] Xiangyu Zhang, Jianhua Zou, Xiang Ming, Kaiming He, and Jian Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1984–1992, 2015.