Neuron Pruning for Compressing Deep Networks Using Maxout Architectures

Fernando Moya Rueda(✉), Rene Grzeszick, and Gernot A. Fink

Department of Computer Science, TU Dortmund University, Dortmund, Germany
fernando.moya@tu-dortmund.de

Abstract. This paper presents an efficient and robust approach for reducing the size of deep neural networks by pruning entire neurons. It exploits maxout units for combining neurons into more complex convex functions and it makes use of a local relevance measurement that ranks neurons according to their activation on the training set for pruning them. Additionally, a parameter reduction comparison between neuron and weight pruning is shown. It will be empirically shown that the proposed neuron pruning reduces the number of parameters dramatically. The evaluation is performed on two tasks, the MNIST handwritten digit recognition and the LFW face verification, using a LeNet-5 and a VGG16 network architecture. The network size is reduced by up to 74% and 61%, respectively, without affecting the network's performance. The main advantage of neuron pruning is its direct influence on the size of the network architecture. Furthermore, it will be shown that neuron pruning can be combined with subsequent weight pruning, reducing the size of the LeNet-5 and VGG16 up to 92% and 80% respectively.

1 Introduction

Having today available a big number of large-scale datasets and powerful GPUs, deep neural networks have become the state-of-the-art in many computer vision, and speech recognition tasks [1,6,10]. They achieve high performance in many applications, e.g., scene and object recognition, object detection, scene parsing, face recognition, and medical imaging. However, they utilize high computational resources coming along with high memory cost [13]. For example, AlexNet and DeepFace have around 60M and 120M parameters, respectively [7]. Furthermore, they consume significant energy making their application on embedded devices difficult [7]. Containing a huge amount of parameters, deep neural networks may also be subject to over-parametrization. Thus, there could exist redundancies, and their generalization is not proper [12]. In general, networks with a small number of parameters generalize better extracting the important information of the data, rather than over-parametrized networks. Nevertheless, smaller networks are harder to train, since they are sensible to initialization [13].

Designing a network, i.e., setting the number of layers, neurons per layer, and parameters is typically still a "trial and error" process. Mostly, it depends on

V. Roth and T. Vetter (Eds.): GCPR 2017, LNCS 10496, pp. 177–188, 2017.
DOI: 10.1007/978-3-319-66709-6_15

experience [1]. Moreover, training does not affect the structure of the network [8]. Several attempts have been developed for reducing the effect of the huge number of parameters, e.g., dropout [10], creating an optimal-sized network by adding additional regularizers, or pruning the network parameters [1,12,17]. The latter ones attempt to either remove edges or complete neurons from a network. However, most of these approaches require an expensive comparison of all neurons in the network or additional and expensive post-processing, e.g., the computation of the network's Hessian.

We propose an efficient and robust method for neuron pruning based on a local decision for reducing the number of parameters in a deep neural network. We will empirically show that pruning neurons rather than weights is essential for reducing the size of a neural network at runtime. The proposed approach for pruning neurons is based on the good performance of maxout units [5], which were developed for boosting the impact of dropout in training, and on their capacity to combine neurons for approximating more complex functions. It is assumed that redundancies typically exist in a neural network, so they also exist in a maxout unit, which combines the output of multiple neurons. Thus, pruning can be performed in a very local approach based on a single maxout unit.

The remainder of the paper is structured as follows: Sect. 2 will discuss the related work in the field of parameter pruning for deep networks. In Sects. 3 and 4, the maxout approach will be reviewed and an approach for pruning neurons will be introduced. Experiments on two datasets, the MNIST digits dataset and the labeled faces in the wild dataset, will be shown in Sect. 5. Face recognition has been chosen as an application that is of special interest for embedded devices such as smartphones. The last section presents a short conclusion.

2 Related Work

Though deep neural networks are very powerful, they are known to be over-parametrized, possessing millions of parameters [13]. This over-parametrization may cause performance deficits, e.g., poor generalization, overfitting, slow testing time, and enormous energy and memory consumption [3,7,8,12]. Therefore, reducing the size of the network by removing unimportant parameters, or designing optimal-sized networks becomes imperative. For those purposes, different attempts have been developed. These attempts can be grouped in constructive and destructive methods.

In constructive methods, neurons or layers are added to a trained shallow neural network. For example, in [17] a very deep convolutional neural network (CNN) is trained by continuously adding convolutional layers to an initial CNN of 11 layers for obtaining a better performance. However, the initial shallow networks must be properly trained, as the network can otherwise get stuck into a local optimum. Moreover, as the idea is to improve the network's performance by adding layers and neurons, the network's size increases, and redundancies could be introduced into the network.

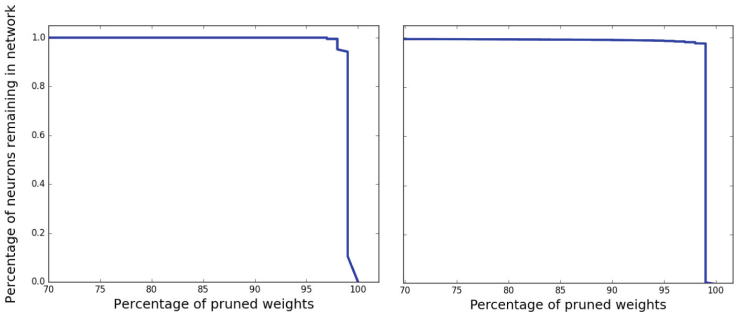
In destructive methods, non-relevant neurons (neuron pruning) and/or parameters (weights pruning) of an initial deep neural network are removed, while

maintaining its behaviour. The authors in [12,14] started the concept of pruning neural networks, both using a sort of relevance measure. In [12], parameters, with the smallest relevance in the network, which is computed by using the Hessian of the loss function, are deleted. In [14], complete neurons are deleted by using a relevance measurement based on the difference of the network's performance with and without the neuron. Nevertheless, especially with today's very deep neural networks with millions of parameters, computing the relevance of each neuron or parameter demands very high computational resources.

Different from the relevance measure methods, the authors in [7,8] prune weights by thresholding them. Afterwards, the network is re-trained for compensating the lost connections. One of the most prominent destructive methods is Deep Compression [7]. The authors reduced the storage required for a deep CNN by a factor of 35 and 49. They used a combination of three steps: weight pruning, weight quantization and Huffman coding. First, weight pruning is applied by thresholding the weights and thus setting them to zero. The remaining weights are then quantized, which reduces the number of bits for representing weights. Finally, a Huffman coding is applied. However, the networks are currently de-compressed for inference.

Recently, the authors in [1] determined the best number of parameters, by using a regularizer while training the network. The regularizer forces all the weights of single neurons to be zero. For testing, these dead neurons are removed from the network. Nevertheless, additional hyper-parameters must be determined for the regularization. The authors in [3] reduced the number of parameters to be learned by factorizing the weight matrices as a low rank product of two matrices: a static, and a dynamic matrix. First, they trained the static matrices as a general dictionary, obtaining a prior knowledge of the smoothness structures that are expected to be seen. Second, they fine-tuned the dynamic matrix, which are the weights to be learned.

Comparing the approaches, the destructive methods are more popular than the constructive ones as the networks are often easier to train. Good compression results are achieved by the deep compression approach in [7]. However, pruning weights has the disadvantage of rarely removing neurons from the network architecture. To make this clear, we show in Fig. 1 the relation between the proportion of remaining neurons in the network versus the proportion of pruned weights using the LeNet-5 [11] and the VGG16 [16] as examples. It clearly shows that neurons only get pruned at a very high compression ratio, which typically influences the networks performance. Thus, thresholding parameters allows for compressing a network but does not influence the size of the architecture. At runtime a sparse matrix library would be required, which efficiently evaluates the compressed network. Otherwise, the network must be de-compressed for inference. The zeroed weights are again stored in the memory, as in [7], which generates a waste in memory consumption as well as computational power.



(a) LeNet-5 **(b)** VGG16

Fig. 1. Percentage neurons remaining in network vs percentage of pruned weights for a LeNet-5 trained for digit recognition and for a VGG16 trained for face recognition (for details see Sect. 5).

3 Fundamentals

The proposed approach builds on a maxout architecture [5] for pruning the neural networks in a destructive manner. A maxout layer can be considered as a cross-channel pooling operation, performing a max operation between k adjacent neurons. These layers were designed to boost the model's averaging ability of dropout [10], thought for preventing overfitting, and to improve the optimization. Given an input layer $X = [x_0, x_1, x_2, \ldots, x_N]$ with N neurons, a maxout layer computes:

$$h(X) = \max[x_{jk+0}, x_{jk+1}, x_{jk+2}, \ldots, x_{jk+(k-1)}] \quad \forall j \in [0, N/k - 1], \qquad (1)$$

where k is the number of neurons that are combined into a single maxout unit.

As the authors in [5] show, a maxout unit is a universal approximator. It combines k single neurons implementing a piecewise linear function that can approximate arbitrary convex functions. So, in theory, the maximum of several neurons is able to approximate a more complex neuron. Moreover, the maxout unit becomes a sort of an activation function, replacing other activation functions, but with a factor of k smaller number of parameters. For example, two linear functions can implement a ReLU function, or five different linear functions can implement an approximation of a quadratic one, as shown in [5].

4 Compressing Networks with Maxout Architectures

The idea of the proposed approach is to use the maxout units and their model selection abilities for pruning entire neurons from an architecture without expensive processing. Thus, reducing the size and the memory consumption of a deep network. In some cases, the performance of the network may even increase as redundancies get reduced or eliminated.

Following the assumption that redundancies exist in a deep neural network, it is assumed that if a network contains a maxout layer, redundancies will, also, exist in the maxout units. This is a valid assumption, since dropout and other regularization approaches cause the learn process to create different paths through the deep network, which yield similar outputs [5]. Using this premise, a reduction of the number of neurons in a maxout layer can be done without an expensive relevance measurement.

4.1 Neuron Pruning

For reducing the size of a CNN using maxout units, an iterative process is followed. First, a CNN with a maxout layer is trained. This maxout layer performs a max function among k adjacent neurons, reducing the amount of weights connecting with the next layer by a factor of k. So, placing this maxout layer after the one with the highest number of weights would be advisable. Second, by counting the number of times neurons become the maximal value in each maxout unit when computing a forward pass over the training dataset, the least active neurons of each maxout unit are removed from the network. Their effects are negligible with respect to other neurons. Third, the remaining neurons of the CNN are re-trained. After re-training, the process is repeated; in this case, the maxout layer performs a max function among $k-1$ neurons, and so on. Figure 2 shows an example for $k = 4$.

In comparison with [14], pruning neurons takes place locally, since relevance values are not computed depending on the network's output for each single neuron. The pruning in maxout architectures is therefore more feasible for very large networks with millions of parameters.

4.2 Weight Pruning

Having reduced the number of parameters in the network by pruning neurons from the maxout units of the network, further compression operations can be performed. Following the approach in [7,8], connections (weights) can be pruned in an additional processing step. Based on thresholding, edges with lower value than a threshold are set to zero. Thus, learning which connections are important and deleting the unimportant ones. By this weight pruning, the network

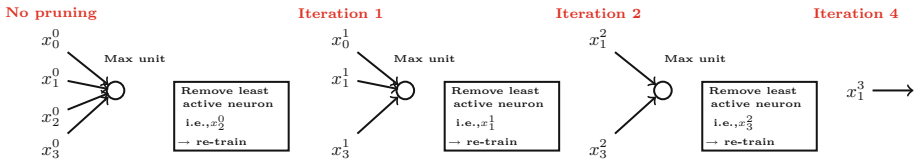


Fig. 2. Neuron pruning process for $k = 4$ inputs per maxout unit. x_a^b represents a neuron with 'a' the neuron index and 'b' the iteration.

becomes a sparse network. For pruning weights, a three-step procedure is followed, as proposed in [7,8]. Given the network that has been compressed by the proposed neuron pruning, the important connections are learned based on a global threshold. The threshold can be set such that as many connections as possible are removed without deteriorating the performance on a validation set. Second, weights below this threshold are deleted; that is, weights are set to zero. Third, the network is re-trained, learning the final weights.

5 Evaluation

An evaluation of both neuron and weight pruning is carried out for two different tasks: handwritten digit recognition, MNIST dataset [11], and face verification, LFW dataset [9]. The later is of special interest for embedded domains, e.g., in mobile phones. In general, the performance of the networks is evaluated with a varying percentage of pruned weights: after applying maxout, when pruning several neurons from the maxout units, and finally after applying additional weight pruning. While in the first task a very small LeNet-5 architecture is compressed, in the second task a large VGG16 architecture is compressed.

For the experiments, we chose $k = 4$ for the size of the maxout units as it allows for a fairly good compression and does not reduce the descriptiveness of the network compared to a network without maxout units. The neurons are then iteratively pruned from the maxout units and the network is re-trained after each pruning step.

5.1 Handwritten Digit Recognition

For the digit-recognition task, two networks, using the LeNet-5 architecture [11] with two convolutional layers, a fully connected layer and a softmax layer as a classificator, were trained. One network contains a maxout layer after the fully connected layer (LeNet-MFC), while the other has a maxout layer after the last convolutional layer (LeNet-MC). An iterative training following the steps in Sect. 4.1 is executed using the MNIST dataset [11]. This dataset consists of 60000 handwritten-digit images (of size $[28 \times 28]$) for training and 10000 images for testing. We used stochastic gradient descent (SGD) with a momentum of 0.9, weight decay of 5×10^{-4} with inverse decay, a base learning rate of 0.01 that is iteratively reduced and a batch size of 64 for training. The networks were trained for 10000 iterations.

Table 1 shows the classification accuracy for both networks with different fully connected layer sizes, with and without maxout (after the fully connected layer or the last convolutional layer). Pruning of one up to three neurons is evaluated. It shows also the percentage of pruned weights which do not remain in the network's architecture, denoted by $p.w.\%$. In general, for both networks when using maxout and pruning neurons, the accuracy is maintained. The slight deviations of the accuracies of both networks with respect to the original networks

Table 1. Accuracies in [%] and pruned weight's proportions (p.w.%) in [%] for LeNet-5 with Maxout layer ($k = 4$) after last fully connected layer (LeNet-MFC) and last convolutional layer (LeNet-MC).

Network		No maxout	No prun		1 neuron prun.		2 neuron prun.		3 neuron prun.	
Type	FC size	Acc%	Acc%	p.w.%	Acc%	p.w.%	Acc%	p.w.%	Acc%	p.w.%
LeNet-MFC	128	98.1	99.1	0.74	99.1	22.6	99.1	40.4	99.1	60.2
	256	98.3	99.2	0.82	99.2	22.8	99.2	44.8	99.0	66.8
	512	99.1	99.2	0.87	99.2	24.1	99.2	47.4	**99.1**	**70.7**
LeNet-MC	128	98.1	99.2	59.4	99.2	64.2	99.2	69.1	99.0	73.9
	256	98.3	99.0	65.9	99.2	68.6	99.3	71.3	99.0	73.9
	512	99.1	99.2	69.7	99.3	71.1	**99.3**	**72.6**	**99.3**	**74.0**

are not significant based on a randomization test [15]. Moreover, the number of weights are considerably reduced with up to 70% for LeNet-MFC and 74% for LeNet-MC. However, this reduction changes with respect to the position of the maxout layer. In LeNet-MFC, each neuron pruning step reduces the number of weights by 19.8%, because the neurons are pruned from the fully connected layer, which has the largest number of weights in the network. Besides, the maxout layer does not provide a considerable reduction, since it reduces the size of the softmax layer that has less number of weights compared with the other layers. In contrast, the weight reduction in LeNet-MC due to neuron pruning is just 1.4% per step, and it comes mostly from the maxout layer. In this case, the maxout layer reduces the fully connected layer instead, and the neurons are pruned from the last convolutional layer. However, in the last convolutional layer the number of weights is negligible.

Following the neuron pruning, additional weight pruning, as discussed in Sect. 4.2, can be applied. As mentioned in [8], neurons could also be pruned from the network if all their input weights are zero; that is, the neuron can be considered as *dead*. So, the number of neurons, and thus the number of weights, could be considerably reduced if a proper threshold is used. Nevertheless, analyzing the proportion of dead neurons versus the proportion of pruned weights, neurons do not become *dead* before pruning more than 98% of the weights, see Fig. 1(b). Consequently, weight pruning rarely prunes neurons. Thus, zeroed weights remain in the network as part of the neurons and the network's architecture does not change so that a sparse representation would be required at runtime [7]. However, assuming the usage of such a representation and for reducing storage size of the network, additional weight pruning is applied to both networks. As a basis, we use the networks after pruning three out of four neurons in the maxout units. The results in Table 2 show that the accuracy will not drop if less than 70% of the weights are thresholded. So, a total compression rate of **91**% for LeNet-MFC and **92**% for LeNet-MC, of pruned and zeroed weights, can be reached.

Table 2. Accuracies in [%] for both networks, after pruning three neurons out of four, under different proportions of pruned weights.

Network		Proportions in [%] of pruned weights								
Type	FC size	0	10	30	50	60	70	80	90	98
LeNet-MFC	128	99.0	99.0	99.0	99.0	99.0	**99.0**	98.9	98.3	82.1
	256	99.3	99.3	99.3	99.3	**99.3**	99.2	99.2	98.9	91.0
	512	99.3	99.3	99.3	99.3	99.3	**99.3**	99.2	99.1	92.2
LeNet-MC	128	99.2	99.2	99.2	99.2	99.1	99.1	98.9	97.8	26.5
	256	99.3	99.3	99.3	99.2	99.2	99.1	98.9	96.3	55.3
	512	99.2	99.2	99.2	99.2	99.2	**99.2**	99.0	97.4	60.8

5.2 Face Verification

The neuron pruning was also carried out for a larger network for the purpose of face verification. The task is to verify whether two face-images portray the same person or not. For that purpose, the VGG16 network [16] was utilized, using The Visual Geometry Group Face Dataset (VGG face-dataset) as a training-dataset. This dataset is a large collection of face-images containing 2.6 million face-images from 2622 identities. It does not contain overlapping identities with standard benchmark datasets (LFW, YFT), so it is suitable for training. The VGG16 network, configuration D in [16], is a deep CNN with 16 layers: 13 convolutional layer, two fully connected layers, and a softmax layer. Analogous to the previous LeNet configurations, two configuration of VGG16 are used, in which a maxout network with $k = 4$ is added after the first fully connected layer ($fc6$), called VGG16-MFC, and after the last convolutional layer ($conv\_5$), called VGG16-MC, see Fig. 3. The positions of the maxout layers are set after the layers with the most quantity of weights. Since, the connections between $conv\_5$ and $fc6$ have 70.1% of the total amount of weights in the network and the connections between $fc6$ and $fc7$ have an additional 11.6% of the network's weights. The last three fully connected layers were fined-tuned for both networks. In the case of VGG16-MC, the $conv\_5$ was also fine-tuned. We used SGD with a momentum of 0.9, weight decay of 5×10^{-4}, three learning rates $[10^{-2}, 10^{-3}, 10^{-4}]$, as [16], and a batch size of 128.

The network was tested following the procedure in [16], but using the restricted configuration of The Labeled Faces in the Wild (LFW) [9]. The LFW dataset is a standard benchmark dataset for face verification. It contains 13233 face-images from 5749 identities extracted from the Internet. Faces in images were detected using the Viola-Jones face detector [9]. Faces are roughly centered, contain lesser noise but larger bounding-box than the ones in the VGG dataset. Besides, the Bray-Curtis distance (BC; [2]) was used instead of the Euclidean distance. Since, the BC distance works better for high-dimensional vectors in comparison with the Euclidean and the L1 distances [18]. The BC distance was measured between the descriptors of two face-images from a set

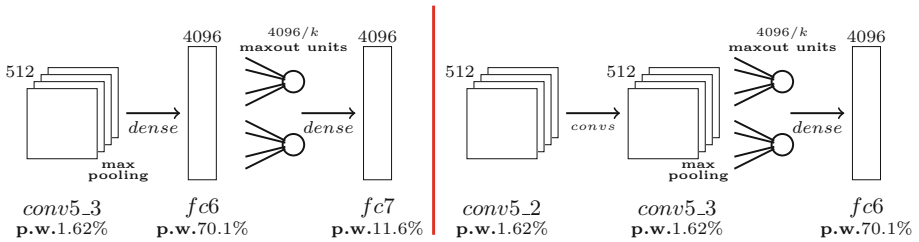


Fig. 3. Comparison of two architectural approaches for placing the maxout units: (left) VGG16-MFC, (right) VGG16-MC. The weight proportions p.w.% per layer are also shown.

Table 3. EER in [%] and pruned weight's proportions for the VGG16 with maxout layer ($k = 4$) after the first fully connected layer (VGG16-MFC) and after the last convolutional layer (VGG16-MC).

Network	No maxout	No prun		1 neuron prun.		2 neuron prun.		3 neuron prun.	
Type	EER%	EER%	p.w.%	EER%	p.w.%	EER%	p.w.%	EER%	p.w.%
VGG16-MFC	3.7	3.33	8.68	**2.83**	26.39	**2.76**	44.11	**2.66**	**61.82**
VGG16-MC	3.7	3.7	53.15	3.36	53.56	3.36	53.96	**3.23**	**54.37**

of 3000 matched and 3000 non-matched pairs of images. Different from [10,16], the feature vectors from the crops of the image's corners were not utilized for computing the final descriptor, but only the crops from the image's centers. If the BC distance is smaller than a threshold, then the two images portray the same identity. The Equal Error Rate (EER) [4,16] was used as the metric, which is defined as the value where the False Acceptance Rate (FAR), and the False Rejection Rate (FRR) are equal.

Table 3 shows the EER for networks without a maxout layer and with a maxout layer with $k = 4$, as well as the results for pruning from one up to three neurons from each maxout unit. Similar to the previous results, neurons are pruned, and consequently weights are reduced, from the networks without affecting their performance negatively. In fact, the EER decreases by 1% and 0.47% for the VGG16-MFC and the VGG16-MC respectively. It is assumed that this improvement is produced by the elimination of redundancies in the maxout units. Based on a randomization test, the improvement in the VGG16-MFC is highly significant with $p = 0.0026$ [15].

The weight reduction changes depending on the position of the maxout layer and on the layer where neurons are pruned in the network. In VGG16-MFC, neurons are pruned from the largest layer in the network $fc6$ reducing the number of weights by 17.7% per each neuron pruning step. Moreover, the maxout layer reduces directly the size of the second largest layer $fc7$. In VGG16-MC on the contrary, neuron pruning does not affect considerably the size of the network, since it reduces a small layer $conv5$ compared to $fc6$. The weight reduction comes precisely from the maxout layer, which reduces the size of $fc6$. There is

Table 4. EER in [%] for both VGG16 networks, after pruning three out of four neurons, under different proportions of pruned weights.

Network	Proportions in [%] of pruned weights								
Type	0	10	30	50	60	70	80	90	98
VGG16-MFC	2.66	2.90	3.4	**3.34**	3.5	4.0	4.53	50.00	50.00
VGG16-MC	3.26	3.77	**3.93**	4.30	5.40	8.3	35.47	47.63	50.00

a difference of 7.45% between the weight reduction for both networks, since the size of the layer $fc7$ is never changed in VGG16-MC.

Additional to neuron pruning, weights from both networks were thresholded after pruning three out of four neurons in the maxout units with the results shown in Table 4. The network's performance will be, deeply, affected if more than 50% for the VGG16-MFC and 30% for the VGG16-MC of the weights are pruned. Nevertheless, a total compression rate of 80.1% for VGG16-MFC and 68% for VGG16-MC without performance deterioration can be reached.

6 Conclusion

We have presented an efficient approach for reducing the size of deep neural networks. This approach prunes entire neurons and thus reduces the number of weights in neural networks. It uses maxout units for combining k single neurons into complex ones. A maxout layer reduces the number of weights between two adjacent layers by k. By using these maxout units, the network's performance is not negatively affected, since they boost the dropout benefits reducing redundancies in the network. Within these maxout units, neurons are pruned based on a local and non-expensive relevance measure. This relevance measure depends on the number of times neurons are maximal for each of the k adjacent input-neurons per maxout unit. It differs from previous relevance measures, because it does not depend on the overall network's performance with and without individual neurons [14]. In general, this approach does not require expensive post-processing, only a single re-training after pruning. The performance of this reduction approach depends strongly on the position of the maxout layer in the network. As inputs from maxout units are the neurons to be pruned, it is advisable to place the maxout units after the largest layer in the network, because neurons in this layer have large numbers of weights compared with neurons in other layers. So, pruning these neurons out of the network is favorable.

By comparing the number of pruned neurons and network's performances between the aforementioned approach and weight pruning, the last one does not delete entire neurons, but rather sets weights to zero. Therefore, the architecture's size is only implicitly reduced, and the memory footprint remains equal without a sparse representation. The proposed approach allows to reduce a network's size by 61–74% on an architectural level and without affecting the network's performance. Assuming a sparse representation, a combination of the

proposed neuron pruning with additional weight pruning allows for reducing the size of a network by up to 92%.

Acknowledgment. This work has been supported by the German Research Foundation (DFG) within project Fi799/9-1 ('Partially Supervised Learning of Models for Visual Scene Recognition').

References

1. Alvarez, J.M., Salzmann, M.: Learning the number of neurons in deep networks. In: Advances in Neural Information Processing Systems, pp. 2262–2270 (2016)
2. Bray, J.R., Curtis, J.T.: An ordination of the upland forest communities of southern wisconsin. In: Ecological monographs, vol. 27, pp. 325–349. Wiley Online Library (1957)
3. Denil, M., Shakibi, B., Dinh, L., Ranzato, M., de Freitas, N.: Predicting parameters in deep learning. CoRR abs/1306.0543 (2013). http://arxiv.org/abs/1306.0543
4. Giot, R., El-Abed, M., Rosenberger, C.: Fast computation of the performance evaluation of biometric systems: application to multibiometrics. In: Future Generation Computer Systems, vol. 29, pp. 788–799. Elsevier (2013)
5. Goodfellow, I.J., Warde-Farley, D., Mirza, M., Courville, A.C., Bengio, Y.: Maxout Networks. In: ICML (3), vol. 28, pp. 1319–1327 (2013)
6. Grzeszick, R., Sudholt, S., Fink, G.A.: Optimistic and pessimistic neural networks for scene and object recognition. CoRR abs/1609.07982 (2016). http://arxiv.org/abs/1609.07982
7. Han, S., Mao, H., Dally, W.J.: Deep compression: compressing deep neural network with pruning, trained quantization and huffman coding. CoRR abs/1510.00149 (2015). http://arxiv.org/abs/1510.00149
8. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural networks. CoRR abs/1506.02626 (2015). http://arxiv.org/abs/1506.02626
9. Huang, G.B., Ramesh, M., Berg, T., Learned-Miller, E.: Labeled faces in the wild: a database for studying face recognition in unconstrained environments. Technical Report, pp. 07–49. University of Massachusetts, Amherst, October 2007
10. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems, pp. 1097–1105 (2012)
11. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
12. LeCun, Y., Denker, J.S., Solla, S.A., Howard, R.E., Jackel, L.D.: Optimal brain damage. In: Advances in Neural Information Processing Systems, vol. 2, pp. 598–605. Morgan-Kaufmann (1989). http://papers.nips.cc/paper/250-optimal-brain-damage.pdf
13. Liu, B., Wang, M., Foroosh, H., Tappen, M., Pensky, M.: Sparse convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 806–814 (2015)
14. Mozer, M.C., Smolensky, P.: Skeletonization: a technique for trimming the fat from a network via relevance assessment. In: Advances in Neural Information Processing Systems, pp. 107–115 (1989)

15. Ojala, M., Garriga, G.C.: Permutation tests for studying classifier performance. J. Mach. Learn. Res. **11**, 1833–1863 (2010)
16. Parkhi, O.M., Vedaldi, A., Zisserman, A.: Deep face recognition. In: British Machine Vision Conference (2015)
17. Simonyan, K., Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR abs/1409.1 (2014)
18. Sudholt, S., Fink, G.A.: A modified isomap approach to manifold learning in word spotting. In: Gall, J., Gehler, P., Leibe, B. (eds.) GCPR 2015. LNCS, vol. 9358, pp. 529–539. Springer, Cham (2015). doi:10.1007/978-3-319-24947-6_44