

Compressing Deep Neural Network for Facial Landmarks Detection

Dan Zeng^(✉), Fan Zhao, and Yixin Bao

Key Laboratory of Specialty Fiber Optics and Optical Access Networks,
Shanghai University, Shanghai 200072, China
dzeng@shu.edu.cn, shu.zfan@i.shu.edu.cn, elaine.bao@foxmail.com

Abstract. State-of-the-art deep neural networks (DNNs) have greatly improved the performance of facial landmarks detection. However, DNN models usually have a large number of parameters, which leads to high computational complexity and memory cost. To address this problem, we propose a method to compress large deep neural networks, which includes three steps. (1) Importance-based neuron pruning: compared with traditional connection pruning, we introduce weights correlations to prune unimportant neurons, which can reduce index storage and inference computation costs. (2) Product quantization: further use of product quantization helps to enforce weights sharing, which stores fewer cluster indexes and codebooks than scalar quantization. (3) Network retraining: to reduce training difficulty and performance degradation, we iteratively retrain the network, compressing one layer at a time. Experiments of compressing a VGG-like model for facial landmarks detection demonstrate that the proposed method achieves 26x compression of the model with 1.5% performance degradation.

Keywords: Compression · Pruning · Product quantization · Facial landmarks detection

1 Introduction

Facial landmarks detection is a fundamental work in many face vision tasks, such as face detection [1], emotion recognition [2] and face verification [3, 4]. A good detection method should not only be robust to deformation, expression and illumination, but also computational efficient. Recent years have witnessed the significant improvement in the performance of facial landmarks detection [5–7], mainly due to the development of deep neural networks (DNNs). Typically, larger models are always needed for more accurate facial landmarks detection, which results in greater number of parameters and greater storage demand. In order to reduce the size of detection models and make them well suited for mobile applications, model miniaturization methods are in great need.

In this paper, we address the problem by proposing an effective compression method based on the fusion of pruning and product quantization [8]. We first train a baseline model for 68 facial landmarks, which has a good performance,

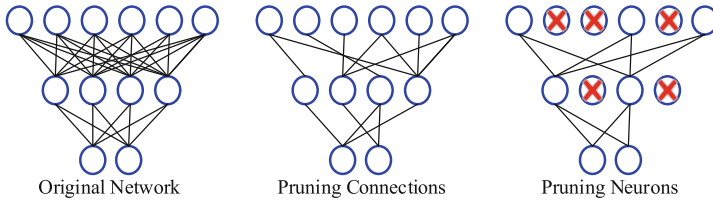


Fig. 1. An illustration of differences between pruning connections and neurons.

but with a large model size. Then we iteratively retrain the network, compressing one layer at a time. When compressing each layer, we prune unimportant neurons based on weights correlations and apply product quantization on the absolute value of remaining weights. Finally, we successfully achieve a 26x compression with only 1.5% performance loss.

Pruning aims at removing redundancy and constructing a sparser network. It shares some similarity with Dropout [11], which is used to avoid overfitting through randomly setting the neural output to zero. Pruning is a common method in dense network compression [12–14], and is usually used to prune connections between neurons. While this kind of pruning can reduce the model size and storage costs, it needs to reconstruct the sparse weights matrix during testing and the computation time and memory may still stay the same. Instead of pruning connections, we introduce the neurons-pruning method, which is equal to build a smaller net with less neurons, as shown in Fig. 1. Product quantization decomposes the original high-dimensional weights into several low-dimensional Cartesian product subspaces, which are then quantized separately. Compared with scalar quantization in [14], product quantization needs fewer cluster indexes and codebooks, which means a high compressing ratio can be obtained without noticeable accuracy loss. On the other hand, public datasets of facial landmarks detection do not provide enough training data. Even by combining several datasets, we have only obtained 4k images of 68 labeled landmarks, which means we must be careful to avoid overfitting. Fortunately, with the use of pruning and quantization, we no longer need to worry about the overfitting problem, and we can use a larger learning rate even without a dropout layer.

The rest of the paper is organized as follows. Section 2 briefly reviews the related work. Section 3 describes the three main parts of our method: neuron pruning, production quantization and network retaining. In Sect. 4, we first introduce our baseline model and then compress the network and evaluate its performance. Finally, the conclusion is given in Sect. 5.

2 Related Work

The most straightforward way to improve the performance of deep neural networks is by making it deeper and wider. However, this strategy makes the networks more prone to be overfitting. Larger amount of training data may solve

this problem, but manually labeling facial landmarks is labor-intensive. Besides, a larger network means more parameters, which can slow down the detection speed and is especially undesirable when we wish to run on mobile devices. Therefore, more and more works begin to explore network compression.

Some works achieve this goal by carefully designing small network architectures. GoogLeNet-V3 [16] not only uses 1×1 convolution kernels to reduce dimension, but also replaces the $n \times n$ convolution with a $1 \times n$ convolution followed by a $n \times 1$ convolution. As a result, the amount of parameters reduces by $\frac{n}{2}$. Unlike this, Courbariaux et al. [17] train a binarized neural network with binary weights and activations, which drastically reduces memory consumption. Denil et al. [18] represent the weight matrix as a product of two low rank factors, during training, they fix one factor and only update the other factor. Similarly, Sainath et al. [20] use a low-rank matrix factorization to reduce parameters in the final weight layer. However, training a factorized representation network directly usually performs poorly. Recently, Scardapane et al. [19] design a new loss function to perform features selecting, network training and weights compression simultaneously, but their work is not suitable for convolutional network.

In addition to training small models directly, compressing a larger model into a smaller one is another popular choice. Denton et al. [21] first consider singular value decomposition (SVD) to compress parameters. Gong et al. [22] systematically explore quantization methods for compressing the dense connected layers of DNNs, including binarization, scalar quantization, product quantization and residual quantization. Their experiments show that product quantization is obviously superior to other methods. However, these methods have no network retraining schemes, which can inevitably cause performance degradation. In our method, we draw the strength of product quantization and use it in the retraining procedure so that we can get a high compression ratio with negligible performance loss.

Han et al. [14] compress the network by combining pruning, trained quantization and huffman coding, which is a popular work recently. Inspired by it, we not only prune neural connections but also prune neurons, which means we are able to spend less to store the sparse structure and inference on a smaller net. We also replace its scalar quantization with absolute product quantization and introduce an iterative way to retrain the network layer by layer, the details are shown in Sect. 3. In addition, Sun et al. [13] find that weight magnitude is not a good indicator to the importance of neural connections, so they prune the network based on correlation rather than the weight magnitude, we improve this method and bring it into our work.

3 Our Method

We first train a dense network as our baseline and then compress it with the following steps.

3.1 Neurons Pruning

Having a pre-trained dense network, we use a pruning ratio R ($R > 1$) to control the number of neurons that will be pruned, e.g., if there are P neurons in the input of one layer, then P/R neurons are preserved after pruning. The only problem here is to decide which neurons will be kept. A fairly straightforward approach is to iteratively drop a neuron with minimum prediction error.

$$\Delta(y) = \|\hat{W}x - Wx\|^2. \quad (1)$$

Where x is the input neurons, Δy is the error of output neurons, W and \hat{W} are the original weights matrix and the pruned-weights matrix respectively, \hat{W} is computed by setting the matrix column where the pruned neuron is located to 0. However, this greedy algorithm is inefficient, especially when there are too many neurons. Inspired by [13], we measure the importance of neuron based on the sum of connection correlation, which has two aspects:

- (1) For fully-connected and locally-connected layers that have no weight-sharing, the correlation coefficient between neuron x_i and y_j is computed as follows:

$$r_{ij} = \frac{E[(x_i - u_{x_i})(y_j - u_{y_j})]}{\sigma_{x_i} \sigma_{y_j}}. \quad (2)$$

where μ_{x_i} , μ_{y_j} , σ_{x_i} and σ_{y_j} denote the mean and standard deviation of x_i and y_j , respectively. Then the importance of neuron x_i is:

$$I_i = \sum_{j=1} |r_{ij}|. \quad (3)$$

Then we keep the most important P/R neurons.

- (2) For convolutional layers with weight-sharing, pruning neurons is difficult, especially for pooling operations. So we prune connections first, and then we randomly add or remove weights in order to make the number of weights in each row equal, as shown in Fig. 2. This step dramatically reduces the index storage and makes it easy to apply product quantization on the rest of the weights.

3.2 Product Quantization

We cannot only use neurons pruning to get a high compression ratio, which will dramatically impact the performance, therefore we utilize product quantization to compress further. Product quantization is a popular vector quantization method. With decomposing original high-dimensional space into several low-dimensional subspaces and taking quantization separately, product quantization is able to make a good description of space distribution with less centroid codes.

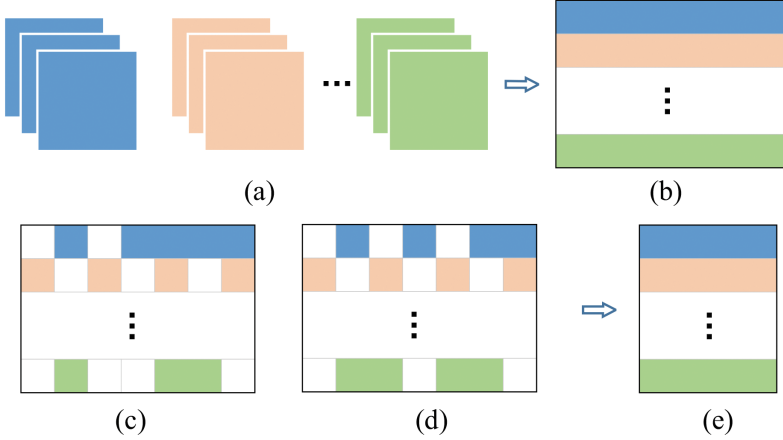


Fig. 2. (a) Convolutional kernels. (b)Weights matrix from reshaping and aligning all kernels. (c) Pruning connections using the method in [13], white squares mean the pruned connections (or weights). (d) Each row has same number of weights left after smoothing. (e)The remaining weights establish a new weights matrix, then product quantization can be applied on it directly.

Given the pruned weights matrix $\hat{W} \in R^{m \times n}$, we first record the positive or negative sign of each weight and substitute \hat{W} for its absolute value. Then we split \hat{W} column-wise into S submatrices

$$\hat{W} = [\hat{W}^1, \hat{W}^2, \dots, \hat{W}^S]. \quad (4)$$

For each submatrix $\hat{W}^i \in R^{m \times (n/s)}$, ($i = 1, \dots, S$), we run the K-Means clustering algorithm on it and get a sub-codebook C_i . The whole code space is therefore defined as the Cartesian product

$$C = C_1 \times C_2 \times \dots \times C_S. \quad (5)$$

For each row \hat{W}_r in \hat{W} , we can reconstruct it as

$$\hat{W}_r = [\hat{W}_r^1, \hat{W}_r^2, \dots, \hat{W}_r^S], \quad \text{where } \hat{W}_r^i \in C_i, \quad r = 1, 2, \dots, m. \quad (6)$$

Supposing that all submatrices have the same cluster number k , we can use $k \times S$ subcodes to generate a codebook with a large size of k^S . That is the reason why product quantization consumes less memory than scalar quantization. Suppose scalar quantization has the same kS clustering centers and the data format is float32, product quantization will yield a $(32kS + mn \log_2(kS))/(32kn + ms \log_2(k))$ compression ratio compared with the scalar quantization case.

3.3 Network Retraining

We compress the network layer by layer from backward to forward. For every iteration, we use the previously trained model to calculate correlated coefficients,

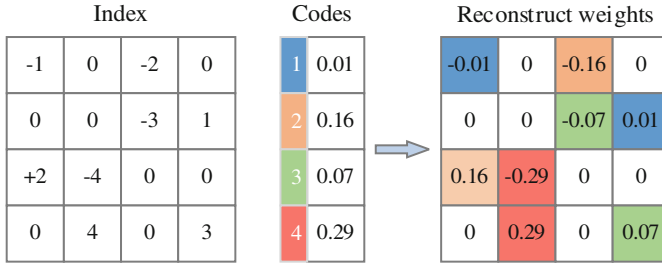


Fig. 3. Reconstructing weights with index and codes. Only if the index is nonzero, the corresponding weight will be recovered by looking up codes tables and adding a corresponding positive or negative sign.

and then prune and quantify one additional layer. To retrain the network, we use deep learning tools Caffe [23] and simply modify its convolutional and fully-connected layers by adding another two blobs to store index and codes. Each time before forward-propagation, we use index and codes to reconstruct the weights. Particularly, if the index is zero, it means the corresponding connection has been pruned, otherwise if the index is nonzero, we recover the corresponding weight by looking up tables and adding a positive or negative sign, as shown in Fig. 3. During back-propagation, we only update the centroids codes using the method describes in [14]. Layers after pruning and quantization become very sparse, therefore we no longer need the dropout layer before or after them, which can speed up training without suffering from overfitting.

4 Experiments

In this section, we first introduce our baseline VGG-like model, and compare its performance with SDM [10]. Then we compress the model, and compare the performance before and after compression. Finally, we discuss the impact of some parameters on the trade-off between accuracy and compression ratio.

4.1 Baseline Model

Our facial landmarks dataset contains 4025 images with 68 landmarks collected from LFPW, AFW, HELEN and 300W [24–27], and we randomly select 425 images as our test set. Then we augment the training set using mirror transform and geometric transform such as shifting, rotating. The baseline model is fine-tuned from VGG [9] net with the number of output of last fully-connected layer changing to 136 and the original softmax-loss layer replacing with Euclidean-loss layer. The architecture of our baseline model is shown as Table 1.

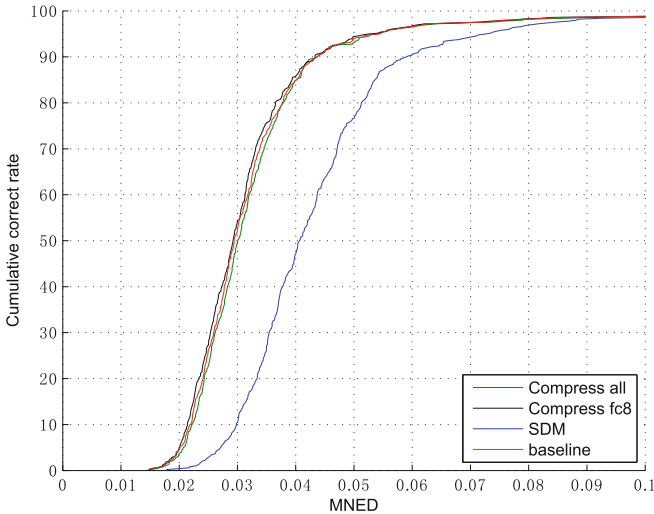
In order to measure the performance, we introduce two metrics:

- (1) Mean normalized euclidean distance (MNED):

$$MNED = \frac{1}{M} \sum_{i=1}^M \frac{\sum_{j=1}^{68} (p_j - \hat{p}_j)^2}{d_i}. \quad (7)$$

Table 1. Structure of baseline model and configurations for compressed network

Layer	Weights	Params	Proportion	R	K	D
conv1+Maxpool	$3 \times 7 \times 7 \times 96$	14K	0.0001	-	-	-
conv2+Maxpool	$96 \times 5 \times 5 \times 256$	614K	0.0062	-	-	-
conv3	$256 \times 3 \times 3 \times 512$	1.18M	0.0119	-	128	4
conv4	$512 \times 3 \times 3 \times 512$	2.36M	0.0237	2	128	4
conv5+Maxpool	$512 \times 3 \times 3 \times 512$	2.36M	0.0237	4	128	4
fc6+dropout1	18432×4096	75.50M	0.7598	4	256	8
fc7+dropout2	4096×4096	16.78M	0.1689	2	128	4
fc8+loss	4096×136	557K	0.0056	2	64	4

**Fig. 4.** Performance of the compressed network. The “Compress all” represents the compressed model with the configuration in Table 1. And the “Compress fc8” represents a same model with fc8 layer compressed alone.

where M is the number of images in the test set, d_i is the eye distance, p_j is the ground truth position of the specific facial landmark and \hat{p}_j is the predicted position.

- (2) A ROC-like curve, whose horizontal axis represents MNED and vertical axis denotes the percentage of images with error less than its MNED.

The MNED of our baseline model is 0.0329, while the MNED of SDM [10] is 0.0440, much larger than ours. The ROC-like curve is shown as Fig. 4.

4.2 Compressed Network

We compress the baseline network iteratively with the method described in Sect. 3. We directly remove the dropout layers before retraining the network, since they are unnecessary and will reduce retraining speed. The compressing configurations are shown as Table 1, where R indicates the ratio of pruning, D indicates dimensionality of subspace in product quantization and K indicates the cluster number in each subspace.

We prune neurons for fully connected layer and connections for convolutional layer respectively. For example, the size of left dense weights matrix in fc6 layer becomes 18432×2014 after pruning neurons in fc7 layer, and this matrix further becomes 4 times sparser after pruning connections in fc6 layer. Since the parameters in the fully connected layer are mostly redundant, a larger compression ratio is used. On the other hand, parameters in lower layers contribute a smaller part of the whole network and are harder to be reduced, so we take a smaller (or zero) compression ratio on them. As shown in Fig. 4, by making necessary trade-offs between performance and compression ratio, we achieve 26x compression of the baseline model with only 1.5% performance degradation. It is worth noting that, the black line in Fig. 4 represents a same model with fc8 layer compressed alone, whose performance is surprisingly better than baseline. We think some suitable sparseness will enhance generalization ability of DNNs models.

4.3 Discussion About Different Configurations

By now we know that quantization helps to reduce parameter size, while pruning can not only reduce parameter size, but also improve inference speed. It seems that we should heavily prune to get both high compression ratio and speed improvement. However, for facial landmarks detection, we find that a higher pruning ratio leads to significant performance degradation. Figure 5 shows the comparison of different pruning ratio on fully-connected layers, and the pruning method is described in [13]. The result shows the method [13] is not suitable for our facial landmarks detection, and pruning too many connections will severely degrade the performance.

In this case, we must be careful to prune neurons or connections and use product quantization to compress more parameters. We find that the performance of product quantization is positively correlated with the cluster number K in each subspace, and negatively correlated with the dimensionality D of subspace. To get a higher compression ratio, we obviously need a small K and a large D , but the performance will suffer from this. Fortunately, we find that applying product quantization on the absolute values can help, and the only extra expense is that we need one bit to record a positive or negative sign for each value. Figure 6 shows the comparison of taking product quantization on the absolute values for original values. The experiment of the weights of fc7 layer demonstrates that our method can use fewer quantization centroids with less error and compress the network further.

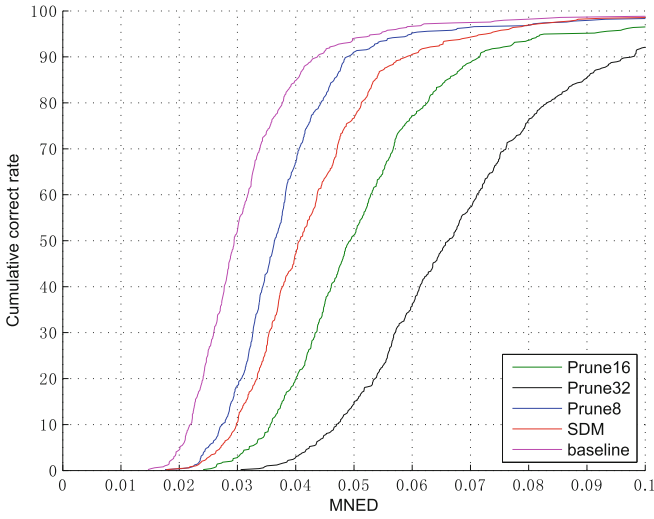


Fig. 5. Comparison of different pruning ratio on fully-connected layers, the pruning method is described in [13]. “PruneX” represents all fully-connected layers are pruned with a same pruning ratio X.

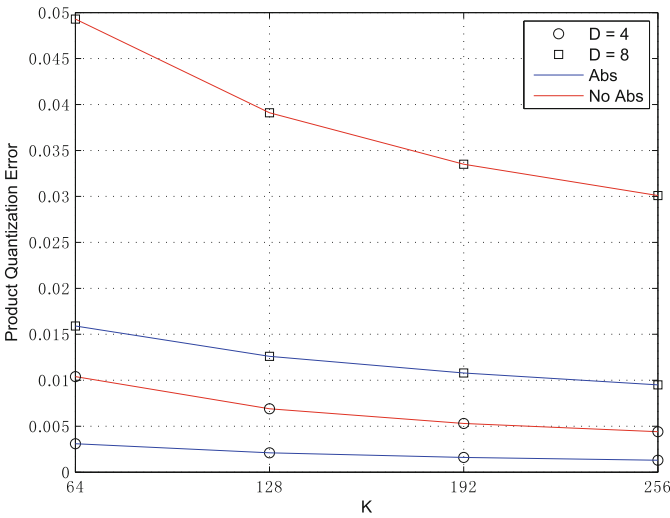


Fig. 6. Comparison of product quantization on absolute values or original values with different dimensionality D of subspace. It shows it is able to use fewer quantization centroids with less error on absolute values.

5 Conclusion

In this paper, we propose an efficient method to compress large DNN models. We fuse pruning and product quantization directly to reduce the time of compressing the whole network and the risk of overfitting. In addition, we prune unimportant neurons rather than connections, which not only reduces model size but also speeds up the computation. We also take product quantization on the absolute values of the remaining weights, so we use fewer quantization centroids and compress the network further.

In the experiment of facial landmarks detection with VGG-like model, we successfully achieve a 26x compression with only 1.5% performance degradation. In the future work, we will pay more attention to compression in convolution layers and acceleration on test phase.

References

1. Chen, D., Ren, S., Wei, Y., Cao, X., Sun, J.: Joint cascade face detection and alignment. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8694, pp. 109–122. Springer, Heidelberg (2014). doi:[10.1007/978-3-319-10599-4_8](https://doi.org/10.1007/978-3-319-10599-4_8)
2. Dhall, A., Goecke, R., Joshi, J., Sikka, K., Gedeon, T.: Emotion recognition in the wild challenge 2014: baseline, data and protocol. In: Proceedings of the 16th International Conference on Multimodal Interaction, pp. 461–466 (2014)
3. Taigman, Y., Yang, M., Ranzato, M.A., Wolf, L.: Web-scale training for face identification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2746–2754 (2015)
4. Chen, J.C., Patel, V.M., Chellappa, R.: Unconstrained face verification using deep CNN features. In: 2016 IEEE Winter Conference on Applications of Computer Vision, pp. 1–9 (2016)
5. Sun, Y., Wang, X., Tang, X.: Deep convolutional network cascade for facial point detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3476–3483 (2013)
6. Zhang, Z., Luo, P., Loy, C.C., Tang, X.: Facial landmark detection by deep multi-task learning. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8694, pp. 94–108. Springer, Heidelberg (2014). doi:[10.1007/978-3-319-10599-4_7](https://doi.org/10.1007/978-3-319-10599-4_7)
7. Toshev, A., Szegedy, C.: DeepPose: human pose estimation via deep neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1653–1660 (2014)
8. Jegou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.* **33**, 117–128 (2011)
9. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint [arXiv:1409.1556](https://arxiv.org/abs/1409.1556) (2014)
10. Xiong, X., De la Torre, F.: Supervised descent method and its applications to face alignment. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 532–539 (2013)
11. Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.R.: Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint [arXiv:1207.0580](https://arxiv.org/abs/1207.0580) (2012)

12. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: *Advances in Neural Information Processing Systems*, pp. 1135–1143 (2015)
13. Sun, Y., Wang, X., Tang, X.: Sparsifying Reural Network Connections for Face Recognition. arXiv preprint [arXiv:1512.01891](https://arxiv.org/abs/1512.01891) (2015)
14. 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)
15. 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)
16. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. arXiv preprint [arXiv:1512.00567](https://arxiv.org/abs/1512.00567) (2015)
17. Courbariaux, M., Bengio, Y.: BinaryNet: training deep neural networks with weights and activations constrained to +1 or -1. arXiv preprint [arXiv:1602.02830](https://arxiv.org/abs/1602.02830) (2016)
18. Denil, M., Shakibi, B., Dinh, L., de Freitas, N.: Predicting parameters in deep learning. In: *Advances in Neural Information Processing Systems*, pp. 2148–2156 (2013)
19. Scardapane, S., Comminiello, D., Hussain, A., Uncini, A.: Group sparse regularization for deep neural networks. arXiv preprint [arXiv:1607.00485](https://arxiv.org/abs/1607.00485) (2016)
20. Sainath, T.N., Kingsbury, B., Sinthwani, V., Arisoy, E., Ramabhadran, B.: Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6655–6659 (2013)
21. Denton, E.L., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R.: Exploiting linear structure within convolutional networks for efficient evaluation. In: *Advances in Neural Information Processing Systems*, pp. 1269–1277 (2014)
22. Gong, Y., Liu, L., Yang, M., Bourdev, L.: Compressing deep convolutional networks using vector quantization. arXiv preprint [arXiv:1412.6115](https://arxiv.org/abs/1412.6115) (2014)
23. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Darrell, T.: Caffe: convolutional architecture for fast feature embedding. In: *Proceedings of the 22nd ACM International Conference on Multimedia*, pp. 675–678 (2014)
24. Belhumeur, P.N., Jacobs, D.W., Kriegman, D.J., Kumar, N.: Localizing parts of faces using a consensus of exemplars. *IEEE Trans. Pattern Anal. Mach. Intell.* **35**, 2930–2940 (2013)
25. Zhu, X., Ramanan, D.: Face detection, pose estimation, and landmark localization in the wild. In: *Computer Vision and Pattern Recognition (CVPR)*, pp. 2879–2886 (2012)
26. Liang, L., Xiao, R., Wen, F., Sun, J.: Face alignment via component-based discriminative search. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) *ECCV 2008*. LNCS, vol. 5303, pp. 72–85. Springer, Heidelberg (2008). doi:[10.1007/978-3-540-88688-4_6](https://doi.org/10.1007/978-3-540-88688-4_6)
27. Sagonas, C., Tzimiropoulos, G., Zafeiriou, S., Pantic, M.: A semi-automatic methodology for facial landmark annotation. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 896–903 (2013)