Cogn Comput (2018) 10:359–367
DOI 10.1007/s12559-017-9506-0

Compressing and Accelerating Neural Network for Facial Point Localization

Dan Zeng[1] · Fan Zhao[1] · Wei Shen[1] · Shiming Ge[2]

Received: 23 March 2017 / Accepted: 5 September 2017 / Published online: 17 September 2017

Abstract State-of-the-art deep neural networks (DNNs) have greatly improved the accuracy of facial landmark localization. However, DNN models usually have a huge number of parameters which cause high memory cost and computational complexity. To address this issue, a novel method is proposed to compress and accelerate large DNN models while maintaining the performance. It includes three steps: (1) importance-based pruning: compared with traditional connection pruning, weight correlations are introduced to find and prune unimportant neurons or connections. (2) Product quantization: product quantization helps to enforce weights shared. With the same size codebook, product quantization can achieve higher compression rate than scalar quantization. (3) Network retraining: to reduce compression difficulty and performance degradation, the network is retrained iteratively after compressing one layer at a time. Besides, all pooling layers are removed and the strides of their neighbor convolutional layers are increased to accelerate the network simultaneously. The experimental results of compressing a VGG-like model demonstrate the effectiveness of our proposed method, which achieves 26× compression and 4× acceleration while the root mean squared error (RMSE) increases by just 3.6%.

Keywords Network compression · Network acceleration · Pruning · Product quantization · Facial point localization

✉ Dan Zeng
dzeng@shu.edu.cn

Fan Zhao
shu_zfan@i.shu.edu.cn

Wei Shen
shenwei1231@gmail.com

Shiming Ge
geshiming@iie.ac.cn

1 Key Laboratory of Specialty Fiber Optics and Optical Access Networks, Shanghai University, Shanghai 200072, China

2 Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100195, China

Introduction

Facial landmark localization is a fundamental problem of many face-related vision tasks, such as head pose estimation [1, 2], facial emotion recognition [3], and face verification [4, 5]. Recent deep neural networks (DNNs) have significantly improved the performance of facial landmark localization under extreme conditions [6–8]. However, an excellent localization method should not only be robust to facial deformation, expression, and illumination but also be efficient for storage and computation. DNNs generally have a huge number of parameters which lead to extensive storage and time cost. These are especially undesirable when detecting landmarks on mobile devices. To address this issue, model miniaturization and acceleration techniques are greatly needed.

In this paper, an effective framework is proposed to solve the above problems. It integrates importance-based pruning and product quantization [9] to compress the model. All pooling layers are merged with their neighbor convolutional layers to accelerate the model simultaneously. More specifically, a VGG-like [11] baseline model is firstly trained to locate 68 facial landmarks. This dense network has excellent performance but a large model size. Then this network is retrained iteratively after compressing one layer at a time. When compressing each layer, unimportant neurons or connections are pruned based on weight correlations. Product quantization is then applied on the absolute values of remaining weights. To accelerate the model further, all pooling layers are abandoned and the strides of their neighbor convolutional layers are increased before retraining. Finally, we successfully achieve 26× compression and 4× acceleration while the root mean squared error (RMSE) increases by just 3.6%.

Pruning aims at removing redundancy and constructing a sparser network. It shares some similarity with Dropout [13], which is used to avoid overfitting through randomly setting the neural outputs to zeros. Pruning is a common method in dense network compression [14–16]. It is usually used to prune connections between neurons. Although this kind of pruning can reduce the model size, it needs to reconstruct the sparse weight matrix during testing. That means the computation complexity and memory cost will stay the same. Instead of pruning connections, we introduce a neuron-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 DeepCompression [16], product quantization needs fewer cluster indexes and codes with same quantization error. It means a higher compressing ratio can be obtained without noticeable accuracy loss. After pruning and product quantization, we remove all pooling layers and increase the strides of their neighbor convolutional layers. It can save much time of im2col-based [10] convolutional computation.

On the other hand, there is not enough public datasets for facial landmark detection as training data. There are only 4000 images of 68 labeled landmarks collected from several public datasets [27–30], which means it must be careful to avoid overfitting. Fortunately, pruning and quantization both can be regarded as some regularizers. It is no longer needed to over-worry about the overfitting problem, and a larger learning rate can be used without a dropout layer.

The rest of the paper is organized as follows. The "Related Work" section briefly reviews the related work. The "Our Method" section describes the three parts of our method: neuron or connection pruning, production quantization, and network retaining. In the "Experiments" section, the baseline model is introduced and compressed to evaluate the effectiveness of our method. Finally, the conclusion is given in the "Conclusion" section.

Related Work

To reduce the storage and time cost of DNNs, an increasing number of works begin to explore network miniaturization and acceleration techniques.

Some works achieve this goal by carefully designing small network architectures. GoogLeNet-V3 [17] not only used 1 × 1 convolution kernels to reduce the feature dimension but also replaced the $n \times n$ convolution with a $1 \times n$ convolution followed by a $n \times 1$ convolution. Unlike this, Courbariaux et al. [18] trained a binarized neural network with binary weights and activations, which drastically reduced memory consumption. Denil et al. [19] represented the weight matrix as a product of two low-rank factors. During training, they fixed one factor and only updated the other factor. Similarly, Sainath et al. [21] utilized the low-rank matrix factorization to reduce the parameters of fully connected layers. However, training a network with factorized representation directly usually performs poorly. Recently, Scardapane et al. [20] designed a new loss function to perform feature selection, network training, and weight compression simultaneously, but their work is not suitable for convolutional network.

In addition to training small models directly, compressing a large model into a small one is another popular choice. Denton et al. [22] first considered singular value decomposition (SVD) to compress parameters. Gong et al. [23] systematically explored quantization methods for compressing the dense connected layers, including binarization, scalar quantization, product quantization, and residual quantization. Their experiments showed that product quantization is obviously superior to other methods. However, these methods have no network retraining schemes and inevitably cause the performance degradation. In our method, the production quantization is applied with the retraining procedure. So a high compression ratio can be achieved with negligible performance loss.

Han et al. [16] compressed the network by combining pruning connections, scalar quantization, and Huffman coding, which is a popular work recently. In contrast, we prune not only neural connections but also neurons. So it will cost

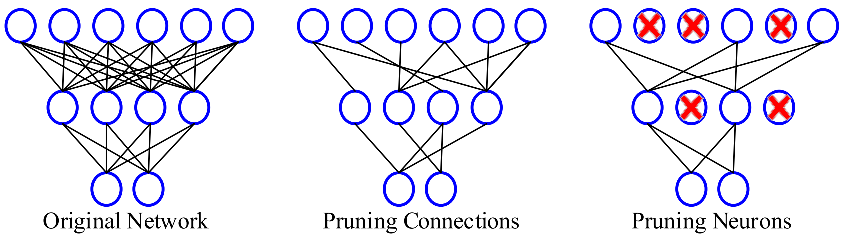


Fig. 1 An illustration of differences between pruning connections and neurons. Pruning a neuron will remove all related connections

less to store the sparse structure and the inference speed will be significantly improved simultaneously. We also replace its scalar quantization with product quantization on absolute values and introduce an iterative way to retrain the network layer by layer. In addition, Sun et al. [15] found that weight magnitude is not a good indicator to the importance of neural connections. So they pruned the network connections based on weight correlations. We improve this method and bring it into our work.

Moreover, some researchers have turned their attentions to the hardware for model compression and acceleration. Han et al. [24] designed a specific hardware accelerator called EIE. It could run a compressed network with sparse and weight sharing neurons directly. IBM team [25] applied structured kernels into convolutional layers on their TrueNorth hardware architecture. It achieved a good trade-off between energy efficiency and classification accuracy. But yet, these techniques have almost not reached a practical level.

Our Method

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

Pruning Neurons and Connections

With the pre-trained dense network, a pruning ratio R ($R > 1$) is used to control the number of neurons or connections that will be pruned. Concretely, only $1/R$ of neurons or connections will be preserved. The key problem here is to decide which neurons or connections will be kept. Deep-Compression [16] removed all connections with weights below a specified threshold. However, weight magnitude cannot indicate the importance of neural connections well. A fairly straightforward approach is to iteratively drop a neuron with minimum prediction error:

$$\Delta y = ||\hat{W}x - Wx||^2 \,. \tag{1}$$

where x is the input, Δy is the error of output, and W and $\hat{W}$ are the original weight matrix and the pruned weight matrix, respectively. To build the $\hat{W}$, all matrix columns corresponding to the pruned neurons will be set to zeros. However, this greedy algorithm is inefficient, especially when there are too many neurons. Inspired by Sun et al. [15], we measure the importance of neuron based on the sum of connection correlations. It contains two parts:

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

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

where μ_{x_i} and σ_{x_i} separately denote the mean and standard deviation of all weights related to input neuron x_i, and μ_{y_j} and σ_{y_j} separately denote the mean and standard deviation of all weights related to output neuron y_j. Then the importance of output neuron y_j is

$$I_j = \sum_{i=1} |r_{ij}| \,. \tag{3}$$

We keep the most important $1/R$ of output neurons and remove the rest.

(2) For convolutional layers with weight sharing, it is impracticable to prune neurons. So we decide to prune connections instead. The correlation coefficients are firstly computed with the method described by Sun et al. [15] to indicate the importance of the weights. Then the same number of weights with minimum importance are removed from each convolution kernel. This specific pruning operation relates to our strategy for convolutional computation. We use im2col [10] to perform fast convolution which reduces this problem to matrix-matrix multiplication. So an aligned weight matrix after pruning will help the reconstruction of the sparse matrix and make it easy to apply product quantization further, as shown in Fig. 2.

Product Quantization

The network performance is sensitive to the pruning ratio. Although a larger pruning ratio can generate a higher compression ratio, the network performance will be severely curtailed as shown in Fig. 7. Therefore, product quantization [9] is applied to compress the network further. Product quantization is a popular vector quantization method. With decomposing original high-dimensional space into several low-dimensional subspaces and taking quantization separately, the data distribution can be described well with less centroid codes.

Given a pruned weight matrix $\hat{W} \in R^{m\times n}$, the positive or negative sign of each weight is first recorded and the $\hat{W}$ is substituted for its absolute value. Then the $\hat{W}$ is split column-wisely into S submatrices:

$$\hat{W} = [\hat{W}^1, \hat{W}^2, \ldots, \hat{W}^S] \,. \tag{4}$$

For each submatrix $\hat{W}^i \in R^{m\times D}, (D = n/S, i = 1, \ldots S)$, the K-means clustering algorithm is performed on it to generate a sub-codebook C_i with k codes. The whole code space is therefore defined as their Cartesian product:

$$\begin{aligned} C &= C_1 \times C_2 \times \ldots \times C_S \,. \\ C_i &= [C_i^1, C_i^2, \ldots, C_i^k]. \end{aligned} \tag{5}$$

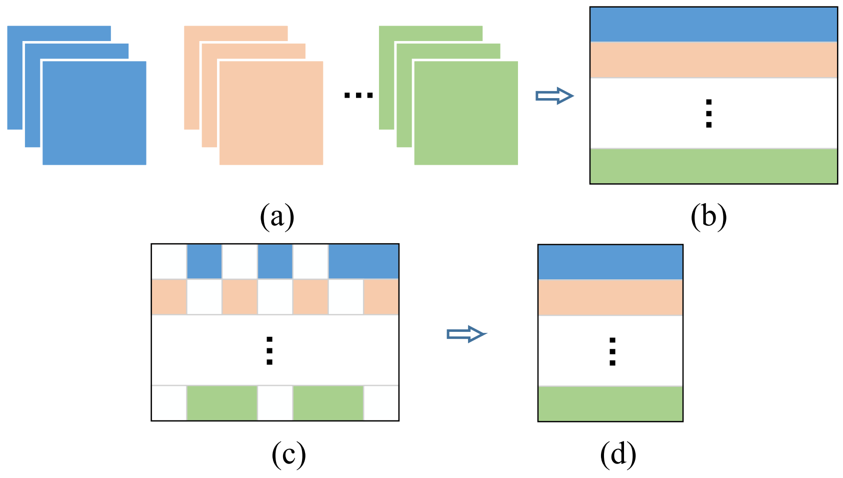


Fig. 2 **a** Convolutional kernels. **b** Weight matrix from reshaping and merging all kernels. **c** Each row has the same number of weights left after pruning, where white squares mean the pruned connections (or weights). **d** A new weight matrix is established from remaining weights, then product quantization can be applied to it directly

For each row $\hat{W}_r$ in $\hat{W}$, it can be reconstructed with the closest code vector:

$$\begin{aligned}\hat{W}_r &= [\hat{W}_r^1, \hat{W}_r^2, \dots, \hat{W}_r^S].\\ \hat{W}_r^i &\leftarrow C_i^j, j = \arg\min_j \| W_r^i - C_i^j \|. \end{aligned} \tag{6}$$

Supposing that all submatrices have the same cluster number k, we can use kS subcodes to generate a large codebook with k^S codes. It is the reason why product quantization consumes less memory than scalar quantization with the same quantization error.

We further find that the distribution of the weight values is symmetrical about the zero. So, we apply product quantization on the absolute values. It reduces the quantization error with the same amount of quantization centroids. And the only extra expense is one bit for each weight to record its positive or negative sign.

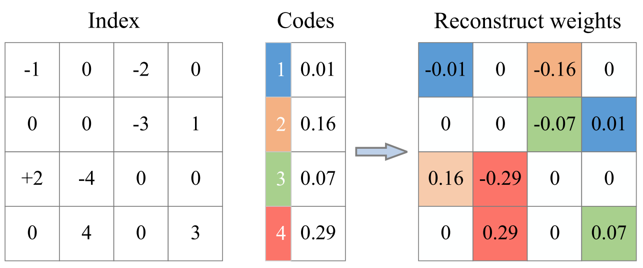
Network Retraining

The network is compressed layer by layer from backward to forward. In each iteration, the correlated coefficients are firstly calculated from the previously trained model. Then one additional layer is pruned and quantified. To retrain the network, we use deep learning framework Caffe [26] and simply modify its convolutional and fully connected layers by adding another two blobs to store indexes and codes. Each time before forward-propagation, the weight matrix is reconstructed with the indexes and codes. Particularly, if the index is zero, it means the corresponding connection has been pruned. Otherwise, this connection will be recovered by looking up tables and adding a positive or negative sign, as shown in Fig. 3. During back-propagation, the centroid codes are updated using the method described in DeepCompression [16]. Layers after pruning and quantization become very sparse. It is no longer needed to use the dropout layers to restrict overfitting.

On the other hand, the parameter size and computation time of the whole convolutional layers are measured in our baseline model. We find that convolutional layers take up more than 44% of operation time while have less than 7% of parameters. It is because im2col [10] spends much time rearranging all convolution patches into a large dense matrix. To optimize this process, we abandon all max-pooling layers and increase the strides of their neighbor convolutional layers. For example, after the first max-pooling layer is removed, the stride of the first convolutional layer will increase from 2 to 6. Increasing the stride reduces both the frequencies of catching pathes and the size of dense matrix. Therefore, the convolution cost can be reduced by about two-thirds.

Experiments

In this section, the baseline model is introduced and compared with SDM [12] and TCDCN [31]. Then the model is compressed and the performance before and after compression are compared. Finally, the impact of some parameters on the trade-off between accuracy and compression ratio is discussed.



Index

-1	0	-2	0
0	0	-3	1
+2	-4	0	0
0	4	0	3

Codes

1	0.01
2	0.16
3	0.07
4	0.29

Reconstruct weights

-0.01	0	-0.16	0
0	0	-0.07	0.01
0.16	-0.29	0	0
0	0.29	0	0.07

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

Table 1 The structure of our baseline model

Layer	Input size	Kernel size	Stride
Conv1	224 × 224 × 3	96 × 7 × 7 × 3	2
MaxPool	109 × 109 × 96	3 × 3	3
Conv2	37 × 37 × 96	256 × 5 × 5 × 96	1
MaxPool	33 × 33 × 256	2 × 2	2
Conv3	17 × 17 × 256	512 × 3 × 3 × 256	1
Conv4	17 × 17 × 512	512 × 3 × 3 × 512	1
Conv5	17 × 17 × 512	512 × 3 × 3 × 512	1
MaxPool	17 × 17 × 512	3 × 3	3
FC6	6 × 6 × 512	18432 × 4096	1
FC7	1 × 4096	4096 × 4096	1
FC8	1 × 4096	4096 × 136	1

Table 2 Configurations for compressed network

Layer	Params	Proportion	R	K	D
Conv1	14 K	0.0001	–	–	–
Conv2	614 K	0.0062	–	–	–
Conv3	1.18 M	0.0119	–	128	4
Conv4	2.36 M	0.0237	2	128	4
Conv5	2.36 M	0.0237	4	128	4
FC6	75.50 M	0.7598	4	256	8
FC7	16.78 M	0.1689	2	128	4
FC8	557 K	0.0056	2	64	4

R ratio of pruning, *D* dimensionality of subspace in product quantization, *K* cluster number in each subspace

Baseline Model

There are 4025 images with 68 landmarks collected from LFPW [27], AFW [28], HELEN [29], and 300W [30]. Four hundred twenty-five images are randomly selected for testing and the rest are for training. Then the training images are augmented through mirror transformation and geometric transformation such as shifting and rotating. The baseline model is simplified from VGG11-Net [11] by removing three convolutional layers. The number of outputs in the last fully connected layer is changed to 136 for facial landmark regression. And the original softmax loss layer is also replaced with a modified Euclidean loss layer as the following formula:

$$Loss = \sum_{i=1}^{136} h_i(y_i - \hat{y}_i)^2. \tag{7}$$

$$h_i = \begin{cases} 0 & |y_i - \hat{y}_i| < \alpha \\ 1 & elsewise \end{cases} . \tag{8}$$

where y_i is the ground truth and $\hat{y}_i$ is the network output. There are 136 outputs in total for 68 facial point coordinates. It is noted that all coordinate values have been normalized to $[-1, +1]$. The α is an adjustable parameter. Prediction error that is less than this tolerance will be ignored. It is mainly

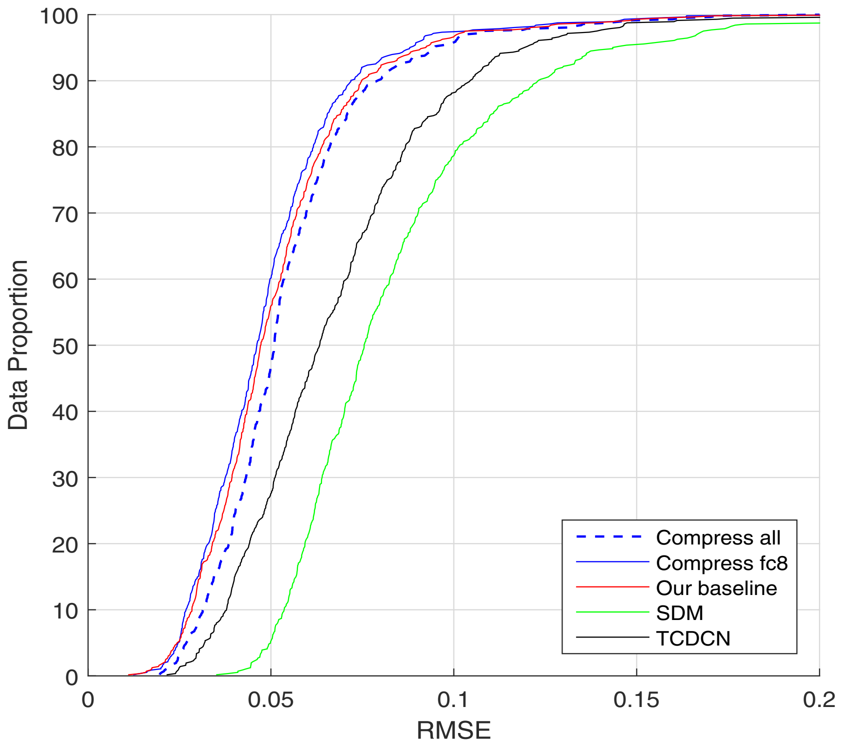


Fig. 4 Performance of the compressed network. The "Compress all" represents the compressed model with the configuration in Table 2 and the "Compress fc8" represents the same model with fc8 layer compressed alone

Table 3 Comparisons between the baseline model and the compressed model

Model	File size	Speed	Mean RMSE
Baseline	379 MB	3.2 FPS	0.0526
Compressed	14.6 MB	12.7 FPS	0.0545

because labeling facial points is affected by subjective factors and the ground truth may be not entirely accurate. Besides, this loss helps to mine hard samples online and strength the network power.

The architecture of our baseline model is shown in Table 1. There are five convolutional layers and three fully connected layers. The entire network model exceeds 379 MB and has about 99.4 M float parameters.

In order to measure and compare the performance, two metrics are introduced:

(1) Root mean squared error (RMSE):

$$RMSE = \frac{\sum_{i=1}^{68}\sqrt{(p_i - \hat{p}_i)^2}}{68d}\,. \tag{9}$$

where d is the distance between the eyes, p_i is the ground truth position of the specific facial landmark, and $\hat{p}_i$ is the predicted position.

(2) Cumulative error distribution (CED) curve: The CED curve plots the error tolerance on the X-axis versus the percentage of samples within the tolerance on the Y-axis.

The mean RMSE of our baseline model is 0.0526. The mean RMSE of SDM [12] and TCDCN [31] are 0.0837 and 0.0680 separately, which are larger than ours. The CED curve is shown in Fig. 4, where a bigger area under curve (AUC) corresponds with a higher regression accuracy. It is clear that the performance of our baseline model is significantly better than SDM and TCDCN.

Compressed Network

The baseline network is compressed iteratively with the method described in the "Related Work" section. The dropout layers are removed before retraining the network, since they are unnecessary and will reduce retraining speed. The compressing configurations are shown in Table 2.

Some connections and neurons are pruned in convolutional layers and fully connected layers separately. For example, the size of dense weight matrix in fc6 layer becomes 18432 × 2014 after pruning neurons in fc7 layer. This matrix further becomes four 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. Parameters in convolutional layers contribute a smaller part of the whole network and are harder to be reduced, so a smaller (or zero) compression ratio is taken on them. By making necessary trade-offs between performance and compression ratio, we achieve 26× compression of the baseline model while the mean RMSE increases by just 3.6%. More specifically, the compressed model has about 1.6 M float parameters and 3 M 8-bit quantization indices. With additional memory for the storage of sparse indexes, the model finally is compressed from 379 to 14.6 MB. The comparisons between the baseline model and the compressed model are shown in Table 3. It is worth noting that the black line in Fig. 4 represents the same model with fc8 layer compressed alone, whose performance is surprisingly better than baseline. It is mainly because some appropriate sparseness can improve the generalization ability of DNN models.

In retraining, all three max-pooling layers are removed and the strides of their neighbor convolutional layers are

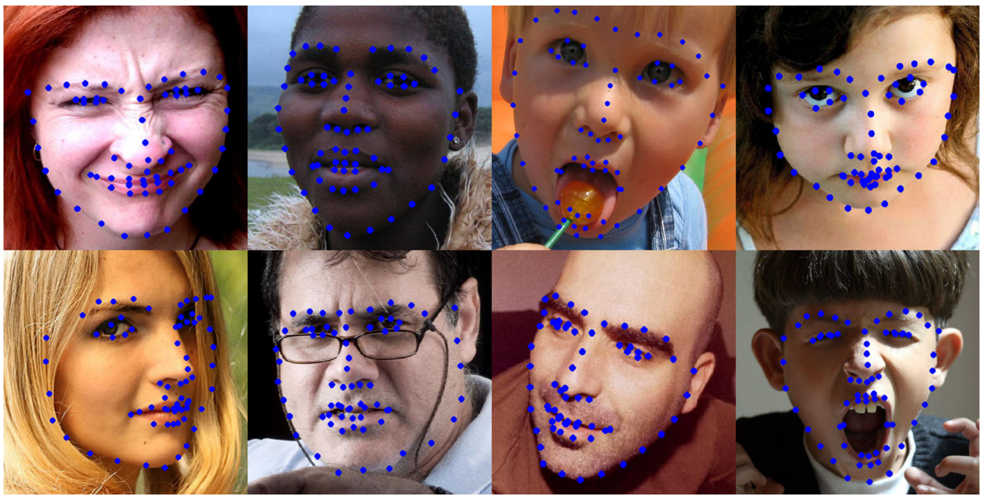


Fig. 5 Some example detection results from our compressed network

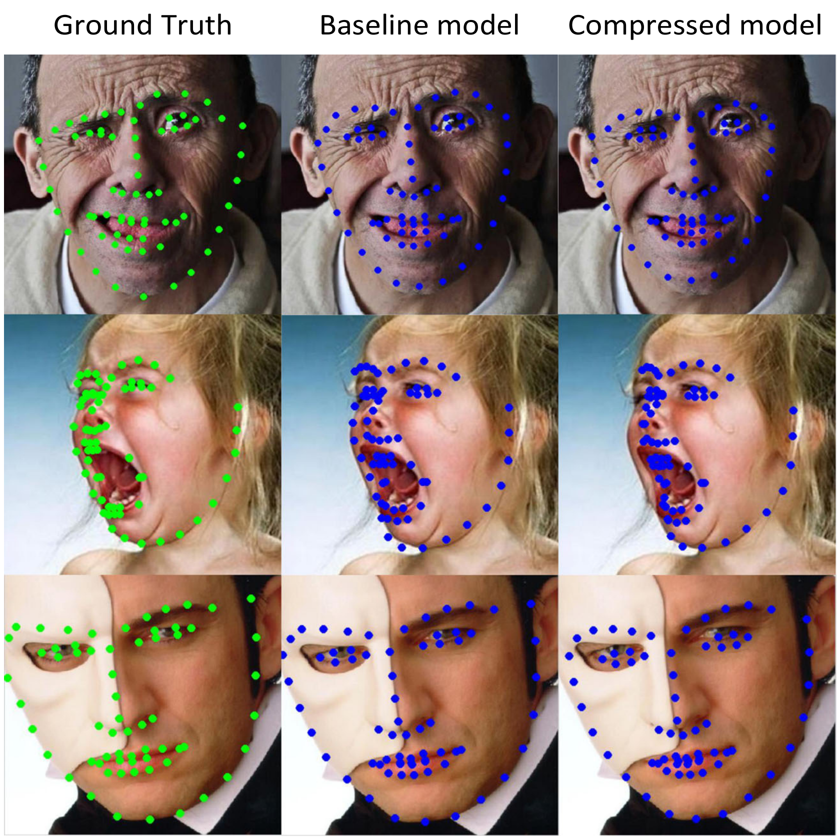


Fig. 6 Some samples with obvious degradation in performance

increased accordingly. Therefore, the speed of the compressed network is further improved.

Some detection examples in Fig. 5 verify the robustness of the compressed model to deformation, expression, and illumination. Moreover, the RMSE of the compressed model is just 3.6% higher than the baseline model. For most testing images, it is hard to see the difference between the baseline model and the compressed model with the naked

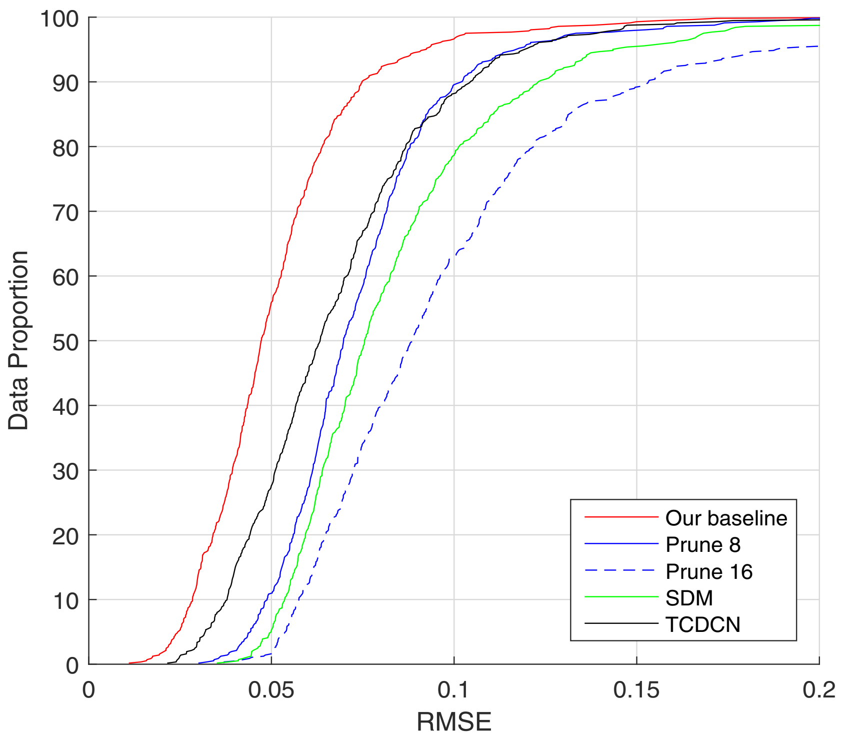


Fig. 7 Comparison of different pruning ratios on fully connected layers. The pruning method is described in [15]. “Prune X” represents all fully connected layers are pruned with the same pruning ratio X

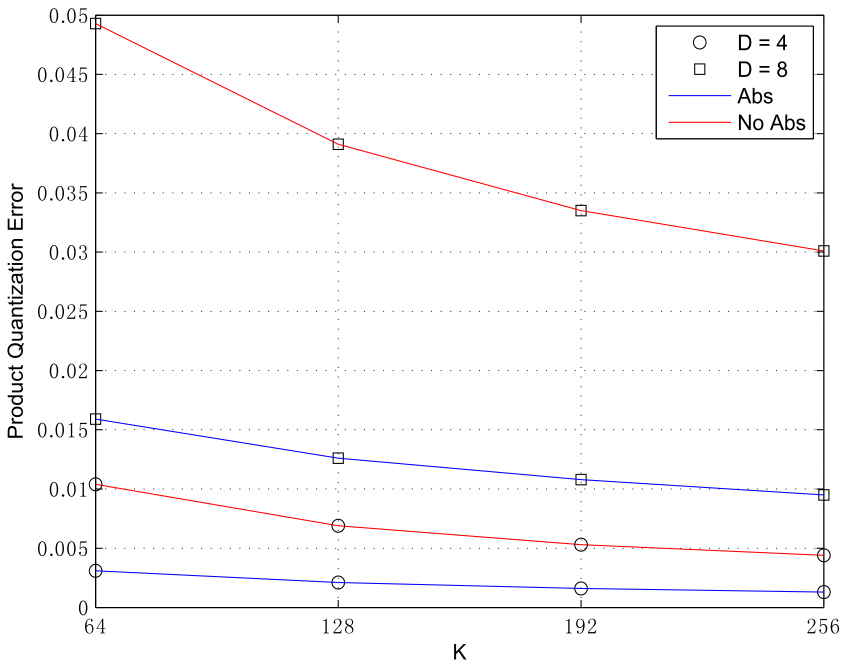


Fig. 8 Comparison of product quantization on absolute values or original values with different dimensionality D of subspace. Product quantization can use fewer quantization centroids with less error on absolute values

eye. We select several samples with obvious differences to show the slight degradation in performance in Fig. 6.

Discussion About Different Configurations

Quantization can only reduce parameter size while pruning can not only reduce parameter size but also improve inference speed. It seems that pruning should be heavily used to get both high compression ratio and speed improvement. However, for facial landmark detection, we find that a higher pruning ratio leads to significant performance degradation. Figure 7 shows the comparison of different pruning ratios on fully connected layers and the pruning method is described in [15]. The results indicate that the method [15] is not suitable for our facial landmark detection and pruning too many connections will severely degrade the performance.

In this case, it must be careful to prune neurons or connections and use product quantization to compress more parameters. 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, it obviously needs a small K and a large D, but the performance will suffer from this. Fortunately, applying product quantization on the absolute values can improve this situation. The only extra expense is that each value needs one bit to record its positive or negative sign. Figure 8 shows the comparison between taking product quantization on the absolute values and original values in fc7 layer. It demonstrates that our method can use fewer quantization centroids with less error and compress the network further.

Conclusion

In this paper, an efficient method is proposed to compress a large DNN model and reduce the runtime cost. First of all, unimportant neurons or connections are pruned based on weight correlations. It can reduce both index storage and inference computation cost. Product quantization is then applied to generate higher compression. Especially, this quantization is taken on the absolute values of the remaining weights. So the same amount of quantization centroids can bring smaller quantization error. Additionally, all pooling layers are removed and the strides of their neighbor convolutional layers are increased during retraining. It accelerates the network further. The experiments of compressing a large dense model for facial landmark detection demonstrate the effectiveness of our proposed method, which achieves 26× compression and 4× acceleration while the mean RMSE increases by just 3.6%. In the future, we will apply our compression and acceleration technique to more existing algorithms for different tasks to further verify the effectiveness and robustness.

Acknowledgments We would like to thank professor Tian Qi (the University of Texas at San Antonio) and Zhou Xi (Chongqing Institute of Green and Intelligent Technology, Chinese Academy of Sciences) for their suggestions to this project.

Funding This work is supported by the National Natural Science Foundation of China (61572307).

Compliance with Ethical Standards This article does not contain any studies with human participants or animals performed by any of the authors.

Conflict of Interest The authors declare that they have no conflict of interest.

References

1. Yan Y, Ricci E, Subramanian R, et al. A multi-task learning framework for head pose estimation under target motion. IEEE Trans Pattern Anal Mach Intell. 2016;38(6):1070–1083.
2. Carreira J, Agrawal P, Fragkiadaki K, et al. Human pose estimation with iterative error feedback. Proceedings of the IEEE conference on computer vision and pattern recognition; 2016. p. 4733–4742.
3. Dhall A, Goecke R, Joshi J, Sikka K, Gedeon T. Emotion recognition in the wild challenge 2014: baseline, data and protocol. Proceedings of the 16th international conference on multimodal interaction; 2014. p. 461–466.
4. Taigman Y, Yang M, Ranzato MA, Wolf L. Web-scale training for face identification. Proceedings of the IEEE conference on computer vision and pattern recognition; 2015. p. 2746–2754.
5. Chen JC, Patel VM, Chellappa R. Unconstrained face verification using deep cnn features. 2016 IEEE Winter conference on applications of computer vision; 2016. p. 1–9.
6. Sun Y, Wang X, Tang X. Deep convolutional network cascade for facial point detection. Proceedings of the IEEE conference on computer vision and pattern recognition; 2013. p. 3476–3483.
7. Zhang Z, Luo P, Loy CC, Tang X. Facial landmark detection by deep multi-task learning. European conference on computer vision; 2014. p. 94–108.
8. Chen Y, Yang J, Qian J. Recurrent neural network for facial landmark detection. Neurocomputing. 2017:26–38.
9. Jegou H, Douze M, Schmid C. Product quantization for nearest neighbor search. IEEE Trans Pattern Anal Mach Intell. 2011:117–128.
10. Chellapilla K, Puri S, Simard P. High performance convolutional neural networks for document processing. Tenth international workshop on frontiers in handwriting recognition; 2006.
11. Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556. 2014.
12. Xiong X, De la Torre F. Supervised descent method and its applications to face alignment. Proceedings of the IEEE conference on computer vision and pattern recognition; 2013. p. 532–539.
13. Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov RR. Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580. 2012.
14. Han S, Pool J, Tran J, Dally W. Learning both weights and connections for efficient neural network. Advances in neural information processing systems; 2015. p. 1135–1143.
15. Sun Y, Wang X, Tang X. Sparsifying neural network connections for face recognition. arXiv:1512.01891. 2015.
16. Han S, Mao H, Dally WJ. Deep compression: compressing deep neural network with pruning, trained quantization and huffman coding. arXiv:1510.00149. 2015.
17. Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the inception architecture for computer vision. arXiv:1512.00567. 2015.
18. Courbariaux M, Bengio Y. Binarynet: training deep neural networks with weights and activations constrained to $+1$ or -1. arXiv:1602.02830. 2016.
19. Denil M, Shakibi B, Dinh L, de Freitas N. Predicting parameters in deep learning. Advances in neural information processing systems; 2013. p. 2148-2156.
20. Scardapane S, Comminiello D, Hussain A, Uncini A. Group sparse regularization for deep neural networks. arXiv:1607.00485. 2016.
21. Sainath TN, Kingsbury B, Sindhwani V, Arisoy E, Ramabhadran B. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. 2013 IEEE international conference on acoustics, speech and signal processing; 2013. p. 6655–6659.
22. Denton EL, Zaremba W, Bruna J, LeCun Y, Fergus R. Exploiting linear structure within convolutional networks for efficient evaluation. Advances in neural information processing systems; 2014. p. 1269–1277.
23. Gong Y, Liu L, Yang M, Bourdev L. Compressing deep convolutional networks using vector quantization. arXiv:1412.6115. 2014.
24. Han S, Liu X, Mao H, et al. EIE: efficient inference engine on compressed deep neural network. arXiv:1602.01528. 2016.
25. Appuswamy R, Nayak T, Arthur J, et al. Structured convolution matrices for energy-efficient deep learning[j]. arXiv:1606.02407. 2016.
26. Jia Y, Shelhamer E, Donahue J, Karayev S, Long J, Girshick R, ..., Darrell T. Caffe: convolutional architecture for fast feature embedding. Proceedings of the 22nd ACM international conference on multimedia; 2014. p. 675–678.
27. Belhumeur PN, Jacobs DW, Kriegman DJ, Kumar N. Localizing parts of faces using a consensus of exemplars. IEEE Trans Pattern Anal Mach Intell. 2013:2930–2940.
28. Zhu X, Ramanan D. Face detection, pose estimation, and landmark localization in the wild. Computer vision and pattern recognition (CVPR); 2012. p. 2879–2886.
29. Liang L, Xiao R, Wen F, Sun J. Face alignment via component-based discriminative search. European conference on computer vision; 2008. p. 72–85.
30. Sagonas C, Tzimiropoulos G, Zafeiriou S, Pantic M. A semi-automatic methodology for facial landmark annotation. Proceedings of the IEEE conference on computer vision and pattern recognition workshops; 2013. p. 896–903.
31. Zhang Z, Luo P, Loy CC, et al. Learning deep representation for face alignment with auxiliary attributes. IEEE Trans Pattern Anal Mach Intell. 2016:918–930.