COMPRESSING DEEP NEURAL NETWORKS FOR EFFICIENT VISUAL INFERENCE

Shiming Ge[1*], *Zhao Luo*[1,2], *Shengwei Zhao*[1,2], *Xin Jin*[3], *Xiao-Yu Zhang*[1*]

[1]Beijing Key Laboratory of IOT information Security, Institute of Information Engineering, Chinese Academy of Sciences
[2]School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100019, China
[3]Beijing Electronic Science and Technology Institute, Beijing 100070, China

ABSTRACT

The deployments of deep neural network models on mobile or embedded devices have been challenged due to two main reasons: 1) the large model size for storage, and 2) the large memory bandwidth for inference. To address these issues, this paper develops a deep neural network compression framework to reduce the resource usage for efficient visual inference. By reviewing the trained deep model, we propose a hybrid model compression algorithm via four major modules. Approximation module reduces the number of weights in each fully connected layer with low rank approximation. Then, quantization module analyzes weight distribution in each layer and represents them with low precision fixed point, which reduces the bits for storing each weight. After that, pruning module suppresses small weights to further reduce the number of parameters. Finally, coding module joint optimizes the representation and encoding of the sparse structure of the pruned weights with relative index by Huffman coding. Beyond the compression of model size, we propose an adaptive fixed point memory allocation algorithm to reduce memory footprint in inference. The proposed framework, along with the model compression and memory allocation algorithms, can provide 20-30x compression rate with negligible accuracy loss. We conduct an evaluation on two representative models, AlexNet and VGG-16, for object recognition and face verification tasks, which demonstrate the effectiveness of our proposed compression framework.

Index Terms— Deep neural networks, model compression, visual inference, object recognition, face verification

1. INTRODUCTION

With the rapid development of modern computing power and large data collection technique, deep neural networks (DNNs) have pushed artificial intelligence limits in a wide range of inference tasks, including but not limited to visual recognition [1] and face verification [2]. For example, visual recognition method proposed in [3] achieves 3.57% top-5 test error on the ImageNet LSVRL-2012 classification dataset, while face verification system [2] achieves over 99.5% accuracy on the public face benchmark LFW [4]. These powerful methods usually rely on DNNs containing millions or even billions of parameters. For example, "very deep" VGG-16 [5] model, which gives impressive results in ImageNet, contains 138 million parameters and takes more than 500MB in storage.

Beyond the remarkable performance, there is increasing concern that larger number of parameters consumes considerable resources (*e.g.*, storage, computation/energy and memory), which prevents their widespread deployments on mobile devices and cloud. On one hand, the storage bandwidth is very critical both for model size and data computation for DNN usage on mobile. On the other hand, memory bandwidth demand is very important to save transmission and power for DNN usage on cloud. To this end, smaller models via compression at least mean that they (i) are easier to download from App Store, (ii) need less bandwidth to update to an autonomous car, (iii) are easier to deploy on embedded hardware with limited memory, (iv) need less communication across servers during distributed training, and (v) need less energy cost to perform face verification and search.

In this paper, we proposed a DNN model compression framework to facilitate compact models for enabling efficient visual inference on mobile devices. We propose a hybrid four-module compression algorithm and an adaptive fixed point memory allocation algorithm to provide scalable resource control. Experimental results on various visual inference models demonstrate that the proposed framework condenses deep models by 20-30x with negligible accuracy loss. The main contributions of this paper include: 1) We design a hybrid deep model compression framework with negligible accuracy degradation to facilitate a range of visual inference tasks on mobile devices, 2) We propose a scalable memory allocation algorithm to adaptively control the resource usages, which facilitates efficient visual inference tasks, and 3) We apply two representative models to validate our approach and investigate the impact of model compression on both the resource usage and accuracy of two representative visual inference tasks including object recognition and face verification.

This work was partially supported by grants from the National Key Research and Development Plan (No.2016YFC0801005) and the National Natural Science Foundation of China (No.61402463). Corresponding authors: Shiming Ge (email: geshiming@iie.ac.cn) and Xiao-Yu Zhang (email: zhangxiaoyu@iie.ac.cn).

978-1-5090-6067-2/17/$31.00

2. RELATED WORK

Many visual inference services have been developed on the cloud where sufficient resources are provided for running deep models. In these services, the user usually uploads visual data (*e.g.*, images or videos) to the service and requires inference results. However, this process is intelligent but inefficient in cases of the considerable issues including network delay, user privacy and power budget. For example, network delay in the process of object recognition on autonomous car may cause an accident. Therefore, it is necessary to perform DNN model compression to enable efficient and effective visual inference. In the literature of DNN model compression, the methods can mainly be classified into four categories.

Network Distillation. In this category, Hinton *et al.*proposed knowledge distilling idea [6] to train a smaller student network by taking the output of a large, capable, but slow pre-trained teacher network. This idea uses the vast network for the regularization process and facilitates subsequent training operations. However, this method requires a large pre-trained network to begin with which is not always feasible. FitNet [7] extended this idea to allow the training of a student that is deeper and thinner than the teacher, using not only the outputs but also the intermediate representations learned by the teacher as hints to improve the training process and final performance of the student. Inspired by these methods, Luo *et al.* [8] proposed to utilize the learned knowledge of a large teacher network or its ensemble as supervision to train a compact student network. The knowledge is represented by using the neurons at the higher hidden layer, which preserve as much information as the label probabilities, but are more compact. The trained student achieves 51.6x compression ratio and 90x speed-up in inference. SqueezeNet [9] designed a small DNN architecture which achieves AlexNet level accuracy on ImageNet but with 50x fewer parameters 461x smaller model size (<1MB).

Network Pruning. In this category, Han *et al.*developed a method to prune unimportant connection and then retain the weights to reduce storage and computation [10]. Later, they proposed a hybrid method, called Deep Compression, to condense deep neural networks with pruning, quantization and Huffman coding [11], which achieves very impressive results. Polyak and Wolf proposed two compression strategies based on eliminating lowly active channels and coupling pruning with repeated use of already computed elements [12]. Sun *et al.* [13] proposed to iteratively learn sparse ConvNets for face recognition. The trained sparse ConvNet model significantly reduced the parameters to 12%.

Precision Reduction. This category aims to use lower precision to approximate model parameters. Gong *et al.*applied k-means clustering to the weights or conducting product quantization [14], which achieved 16-24x compression of the network with only 1% loss of accuracy on ImageNet classification task. Dettmers proposed 8-bit approximation to make better use of the available bandwidth which obtains a speedup of 50x [15], while Gupta *et al.*used 16-bit fixed-point number representation with stochastic rounding [16]. Wu [17] proposed Quantized CNN to simultaneously speedup the computation and reduce the storage and memory overhead of CNN models. Furthermore, Hubara *et al.* [18] proposed to train binarized neural networks (BNNs) with binary weights and activations at run-time.

Structured Parameters. This category usually condense deep models by exploiting the redundancy of weights. Recent studies applied low rank approximation for compressing convolutional layers [19, 20, 21] or whole network by fine-tuning the compressed network with large data [19]. These methods either introduce whole network compression problem which may degrade the inference accuracy or are impractical due to extremely resource intensive retraining. Figurnov *et al.* [22] used loop perforation technique to eliminate redundant multiplication, which allows to reduce the inference time by 50%. In addition, CNNPack [23] compressed the deep models in frequency domain.

3. ANALYSIS ON DNN MODELS

In this section, we give a brief analysis on the structure of deep model, computation and storage complexity, weight distribution and memory bandwidth, which guides our deep model compression framework. More details are introduced in the supplied material.

Deep Neural Networks. A deep neural network usually contains multiple blocks of convolutional layers, activation layers, and pooling layers, following by multiple fully connected layers. The convolutional layers dominate most of the computational complexity since they needs a lot of *multiplication-and-addition* (MAC) operations to extract local pattern, while they contain less weights due to weight sharing and local connectivity. By contrast, fully connected layers contain most of the weights since dense matrix-vector multiplication is very resource-intense. In addition, an activation layer (such as ReLU) contains a nonlinear function to activate or suppress some neurons. It can make the network more sparse and more robust again to over-fitting while reducing the number of connections. A pooling layer is followed by a convolutional layer and aims to merge semantically similar features together, which can reduce the memory bandwidth.

Complexity Analysis. The complexity of DNNs could be spitted into two parts: 1) the convolutional layers contain more than 90% of the required arithmetic operators, and 2) the fully connected layers take about 90% of the network parameters. For example, in VGG-16 model [5], the thirteen convolutional layers contain 99.2% MAC operations, while the three fully connected layers contains 89.4% parameters.

Weight Analysis. DNNs are known to be over-parameterized

for facilitating convergence to good local minima during model training [24]. Therefore, such redundancy can be removed from the trained models in the inference time. In addition, each layer contains a number of weights near zero-value and the distribution is biased: most of the (quantized) weights on each layer are distributed around the zero-value peak. This observation demonstrates that the weights can be reduced through coding, such as Huffman coding.

Memory Analysis. The memory bandwidth of a deep model refers to the inference processing, and greatly impact the energy consumption, especially when running on mobile devices. The memory of each layer is dependent on bit-width of each activation and the number of activation output. By evaluating the ranges of each layer, we can dynamically control the memory allocation in network layers.

4. OUR DNN MODEL COMPRESSION APPROACH

Fig.1 (1) shows the framework of DNN model compression service, a trained deep model downloaded from training servers or uploaded by users is imported to the service, where compression is carried out to condense it. The compressed deep model, along with the memory allocation engine, then can be deployed on various mobile devices.

4.1. Model Compression

Motivated by the analysis on deep models, we propose a hybrid model compression algorithm to facilitate efficient visual inference on mobile devices. As illustrated in Fig.1 (2), the compression algorithm consists of four sequent modules: 1) the approximation module decomposes the weight matrix of each fully connected layer and then approximates it with low rank version, which reduces the number of weights, 2) the quantization module represents the weights with fixed point low-bit precision and therefore reduces model size, 3) the pruning module suppresses the quantized weights with small values, which further reduces the number of weights, and 4) the coding module encodes the bit-width of representation of each weight, which further reduces model size.

Approximation Module. To avoiding the whole model compression problem and retraining or fine-tuning, we adopt singular value decomposition (SVD)-based low rank approximation method to approximate only the fully connected layers since they contain most of the network parameters. Denote the dense weight matrix in a fully connected layer as $W_{M\times N}$ with M and N units for input layer and output layer respectively, then under SVD it can be represented as:

$$W_{M\times N} = U_{M\times M}\Sigma_{M\times N}V^T_{N\times N} \quad (1)$$

Further, the weight matrix can be approximated by keeping r largest singular values, i.e.,:

$$\hat{W}_{M\times N} = U_{M\times r}\Sigma_{r\times r}V^T_{r\times N} = U_{M\times r}L^T_{r\times N} \quad (2)$$

Next, the weight matrix $W_{M\times N}$ is replaced by the product of new matrices $U_{M\times r}$ and $L^T_{r\times N}$. Note that $r \ll M$ and $r \ll N$, low rank approximation enables drastic reduction of the number of weights. The approximation factors $\{r_l\}$ for all the fully connected layers are selected by evaluating the trade-off between model size and construction error.

Quantization Module. Similar with [25], we adopt dynamical fixed point approach to carry out quantization. In dynamical fixed point, for a L-layer network, each weight in l-th layer can be represented as $w = (-1)^{sb}2^{-f_l}\sum_{i=0}^{n_l-2}2^i x_i$. Here quantization factor n_l donates the bit-width, $sb \in \{0, 1\}$ is the sign bit, f_l is the fractional length, and x is the mantissa bits. The workflow is shown in Fig.1 (3). The dynamic range of weights is analyzed to find a good fixed point representation for each layer. Next we run multiple images in forward testing to analyze the activation range to determine the memory bandwidth. Then a search is performed to determine the optimal number of bits for convolutional weights, fully connected weights and layer outputs by evaluating an optimal trade-off between small number representation and the construction accuracy of the last fully connected layer. Finally, we get the initial bit-widths m^0_l and n^0_l for representing memory and weight in l-th layer respectively. They are updated in memory allocation module and the detail is introduced in 4.2.

Pruning Module. This module prunes the connections with small weight values by simply thresholding since most of the quantized weights are around the zero-value peak. A weight will be set to zero if its amplitude is below a pruning factor t. This process can form the weights to a sparse structure.

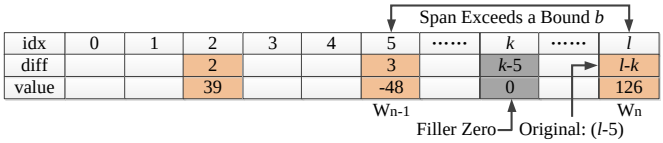

Fig. 2. Representing the sparse structure of the pruned weights with relative index (diff). Filler zero is inserted when span exceeding a bound b to prevent overflow. The optimal bound is determined by minimizing total code length with Huffman coding on both sparse weights and relative index.

Coding Module. After quantization and pruning, the weights can be represented as $\{0, ..., w_1, 0, ..., w_k, 0, ..., w_a, ...\}$. A direct way is to store the a non-zero weights in two-tuples manner: $\{(i_1, w_1), ...(i_k, w_k), ..., (i_a, w_a)\}$, where i_k is the absolute position index of k-th non-zero weight. Note that the absolute index values span too long, as thus we can use relative index instead as $S = \{(d_1, w_1), ...(d_k, w_k), ..., (d_a, w_a)\}$ where $d_k = i_k - i_{k-1}$. Also note that most of $\{d_k\}$ are below a value, we constraints their span by inserting filler zeros when a relative index is larger than a bound b, as shown in Fig.2. To this end, the sparse structure can be represented

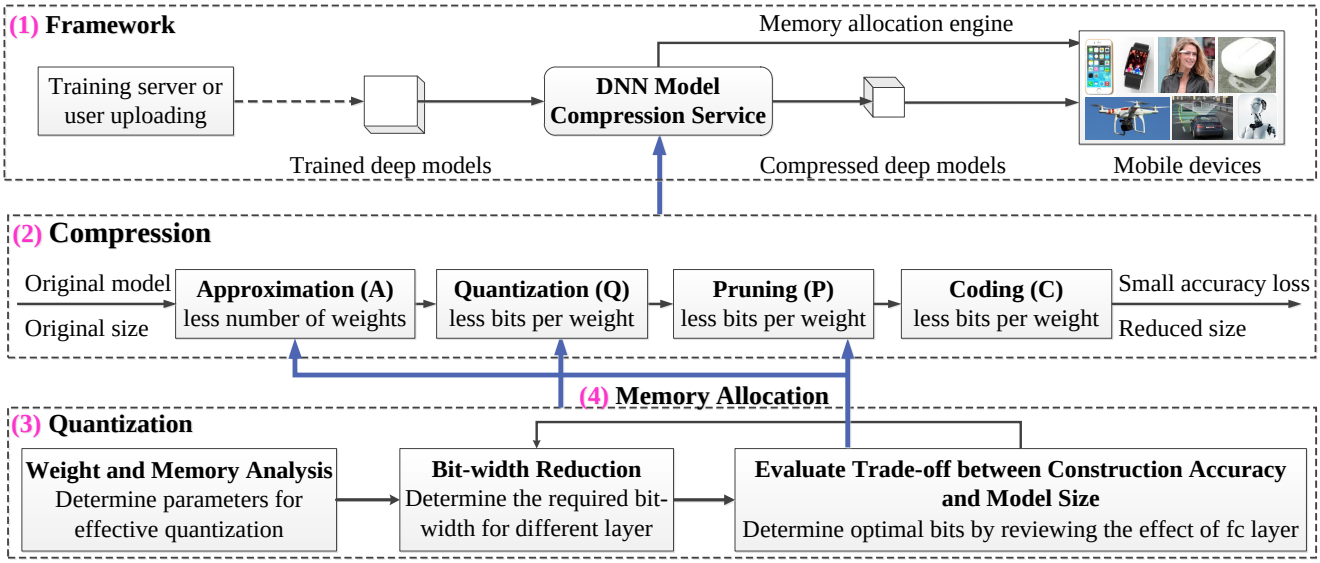

Fig. 1. Our deep model compression approach. (1) The compression framework carries out compression service to the trained deep models and then deploys the compressed deep models along with the memory allocation engine to various mobile devices. (2) The hybrid deep compression algorithm consists of four modules. (3) The quantization module performs adaptive bit allocation by evaluating an optimal trade-off between small number representation and the construction accuracy of the last fully connected layer. (4) The memory allocation module also performs the evaluation.

as sparse weights $K = \{w_1, ..., 0, ..., w_a\}$ and relative index $I = \{d_1, ..., b, ..., d_a\}$. To encode K and I for minimal storing, the minimization problem can be formulated as:

$$\ell_{min}(b_o) = \arg\min_{\{b,K,I\}}\{\ell(b, \hbar(K)) + \ell(b, \hbar(I))\} \quad (3)$$

where ℓ_{min} is the minimal total code length with the optimal bound b_o, $\ell(b, \hbar(*))$ is the code length to represent $*$ with Huffman coding $\hbar(*)$. The problem can be solved with an exhaustive search algorithm.

4.2. Memory Allocation

Beyond the compression of model size, we further proposed an adaptive memory allocation algorithm to reduce the memory usage in a fixed point and shared way. As shown in Fig.1 (3) and (4), both weight and memory are first analyzed, and then jointly optimized by evaluating the trade-off between construction accuracy and model size to achieve optimal bit-widths. The evaluation is carried out three times: 1) for directing rank selection in the approximation module; 2) for guiding reduced precision quantization in the quantization module, and 3) for optimizing memory allocation in compression module when the bit-width of each layer weight is determined. Similar with the dynamical fixed point representation for each weight, the activation in l-th layer can be represented as $a = 2^{-g_l}\sum_{i=0}^{m_l-2} 2^i x_i$ where m_l is the number of bits for l-th layer activations and sign bit sb is ignored since $a \geq 0$. For the l-th layer, the bit-width of $(l+1)$-th layer is

$$m_{l+1} = m_l + n_l + log_2(o_l) \quad (4)$$

where o_l is the multiplication number with non-zero weights.

The distribution of bit-widths across all the layers can be adjusted via fixed point algorithm, which is formulate as an energy minimization problem for optimal bit-width configure:

$$\{m_l^*, n_l^*\} = \arg\min_{\{m_l,n_l\}}\{\lambda_m E_m + \lambda_w E_w + \lambda_c E_c\}, \quad s.t. \sum_l m_l^* = \sum_l m_l^0 \quad (5)$$

where E_m, E_w and E_c are memory bandwidth, weight bits and construction error respectively. λ_m, λ_w and λ_c are balance factors. λ_m and λ_w are set to 0 in the approximation and quantization modules respectively. E_m is defined as the maximal shared layer memory:

$$E_m = \max_l\{m_l s_l\} \quad (6)$$

where s_l is the size of l-th layer output and 0 for input layer. Once $\{m_l, n_l\}$ is fixed, E_c is measured by performing inference on multiple images with original and compressed models, and then computing the average difference in the last fully connected layer. To solve Eq.5 effectively, we fix $n_l = n_l^0$ and then search the optimal memory configure $\{m_l\}$.

5. EVALUATION

5.1. Experiment Settings

To evaluate our method, we select two representative deep models, AlexNet [1] and VGG-16 [5], for benchmarking. In the experiments, we evaluate the performance on both the inference accuracy and resource usage. In particular, we evaluate the impact of each stage in our DNN model compression approach on the performance, which demonstrates the effectiveness of the compression approach. In addition, two representative visual inference tasks are studies: 1) object recognition on the ImageNet LSVRC-2012 dataset which contains 50,000 validation examples, and 2) face verification in LFW testing dataset [4] which includes 6,000 face pairs.

5.2. AlexNet on ImageNet

AlexNet model on ImageNet has 61M floating point parameters and achieved a top-1 accuracy of 57.2% and a top-5 accuracy of 80.3%. Our compressed version achieves the Top-1 accuracy and Top-5 accuracy of 55.4% and 78.6% respectively. First, approximation module reduces the number of the weights to 9.41%. Then, after quantization and pruning, the model parameters can be reduced to 4M. Finally, with the coding, the average weight bit is reduced to 5.39 compared to original 32, which condenses the AlexNet model to 4.37% of its original size (23x compression rate). The detail of compression statistics is shown in the supplied materials.

5.3. VGG-16 on ImageNet

With promising results on AlexNet, we further studied a larger network, VGG-16 model on the same dataset. Following a similar methodology, convolutional and fully connected layers are aggressively compressed with approximation, quantization, pruning and coding, to achieve a significant reduction in the very number of effective weights. Our method can condense the model to 4.27% of its original size. The supplied materials show the compression detail.

5.4. VGG-16 on LFW

Beyond object recognition, we evaluate the compressed VGG-16 model (VGG-Face) on face verification. The original model [26] achieves an accuracy of 97.27% on public LFW benchmark dataset [4] without embedding comparison. To get more effective face representation, we applied the output of the first fully connected layer fc6 to form a face feature. As thus, we compress the thirteen convolutional layers and one fully connected layer in a similar way. Our method can condense the model to 17MB at the accuracy of 96.88%, which achieves 32x compression rate compared to 553MB of original model with negligible accuracy loss (less than 0.4%).

Table 1. Resource usage statistics of storage, the number of MAC operations and MAC reduction rate before (b) and after (a) model compression for AlexNet on ImageNet, VGG-16 on ImageNet and VGG-Face on LFW.

Model	Data	Storage (b/a, MB)	#MAC (b/a, B)	Red. (↓%)
AlexNet	ImageNet	241/11	1.14/0.38	66.4
VGG-16	ImageNet	537/24	15.5/5.83	62.3
VGG-Face	LFW	553/17	15.5/5.82	62.4

Table 2. Compression performance comparison with recent methods. It shows that our approach achieve larger compression rate. (VQ: vector quantization, P: pruning, Q: quantization, TuD: Tucker decomposition, TD: tensor decomposition)

Method	Year	Idea	Retrain?	AlexNet	VGG-16
Gong [14]	2015	VQ	N	16-20x	–
Han [10]	2015	P	Y	9x	13x
Wu [17]	2016	Q	N	15-20x	–
Kim [27]	2016	TuD	Y	5.46x	1.09x
Alvarez [21]	2016	TD	Y	12.5x	12.5x
Han [11]	2016	Hybrid	Y	35x	49x
Our	–	Hybrid	N	21.9x	22.4x

5.5. Resource Usage and Energy Efficiency

Beyond the compression rate, we also looked at the resource usage of the compressed model. Our framework is targeting extremely energy-hungry applications running on mobile devices, which requires fast visual inference, such as face recognition on an embedded processor inside an smart glasses. As shown in Tab.1, 95.4%, 95.5% and 96.9% of model storages can be saved with our approach for AlexNet, VGG-16 and VGG-Face respectively, while the MAC operations are reduced about 2/3. Additionally, our MAC operations are performed with fixed point arithmetic other than float point arithmetic in other methods. Therefore, we can compress the deep models to facilitate mobile deployment.

5.6. Comparison with other methods

We compare the performance with other recent methods and the results are shown in Table.2. Quantization based methods [14, 17] achieved the compression rates of 16-20x and 15-20x on the AlexNet on ImageNet respectively. Han *et al.* [10] pruned the number of parameters of AlexNet by 9x and VGG-16 by 13x respectively, while Kim *et al.* [27] achieved 5.46x and 1.09x compression with Tucker decomposition for the two models respectively. DecomposeMe [21] used tensor decomposition to reduce 92% of the weights for VGG-16 model. By contrast, our hybrid approach achieves larger compression rate. In addition, different from [11], our approach does not need retraining or fine-turning.

6. CONCLUSION

In this work, we studied the deep neural network (DNN) model compression problem and proposed a framework to reduce the resource usage in visual inference. Our findings indicate that large deep models can be compressed to deploy on mobile devices by exploiting the redundance of parameters. By analyzing the features of the trained deep models, we developed an approach to effectively reduce the network parameters as well as bit-width for the representations of parameters and memory, which enables to provide energy-effective visual inference service on mobile devices. Our experimental results demonstrate that one can easily condense large deep models with negligible loss of inference accuracy.

7. REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[2] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *IEEE CVPR*, 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016.

[4] E. Miller, G. Huang, A. Chowdhury, and *et al.*, "Labeled faces in the wild: A survey," *Advances in Face Detection and Facial Image Analysis*, 2016.

[5] K Simonyan and A Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[6] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Workshop*, 2014.

[7] A. Romero, N. Ballas, S. Kahou, and *et al.*, "Fitnets: Hints for thin deep nets," in *ICLR*, 2015.

[8] P. Luo, Z. Zhu, Z. Liu, and *et al.*, "Face model compression by distilling knowledge from neurons," in *AAAI*, 2016.

[9] F. Iandola, M. Moskewicz, K. Ashraf, and *et al.*, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," in *ECCV*, 2016.

[10] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural networks," in *NIPS*, 2015.

[11] S. Han, H. Mao, and W. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *ICLR*, 2016.

[12] A. Polyak and L. Wolf, "Channel-level acceleration of deep face representations," in *IEEE*, 2015.

[13] Y. Sun, X. Wang, and X. Tang, "Sparsifying neural network connections for face recognition," in *IEEE CVPR*, 2016.

[14] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," in *ICLR*, 2015.

[15] T. Dettmers, "8-bit approximations for parallelism in deep learning," in *ICLR*, 2016.

[16] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Arvix*, 2015.

[17] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *IEEE CVPR*, 2016.

[18] I. Hubara, M. Courbariaux, D. Soudry, and *et al.*, "Binarized neural networks," in *NIPS*, 2016.

[19] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *BMVC*, 2014.

[20] C. Tai, T. Xiao, Y. Zhang, and *et al.*, "Convolutional neural networks with lowrank regularization," in *ICLR*, 2016.

[21] J. Alvarez and L. Petersson, "Decomposeme: Simplifying convnets for end-to-end learning," in *Arvix*, 2016.

[22] M. Figurnov, D. Vetrov, and P. Kohli, "Perforatedcnns: Accelerateion through elimination of redundant convolutions," in *ICLR*, 2016.

[23] Y. Wang, C. Xu, S. You, and *et al.*, "Cnnpack: Packing convolutional neural networks in the frequency domain," in *NIPS*, 2016.

[24] D. Misha, S. Babak, D. Laurent, and *et al.*, "Predicting parameters in deep learning," in *NIPS*, 2013.

[25] G. Philipp, M. Mohammad, and G. Soheil, "Hardware-oriented approximation of convolutional neural networks," in *ICLR Workshop*, 2016.

[26] O. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *BMVC*, 2015.

[27] Y. Kim, E. Park, S. Yoo, and *et al.*, "Compression of deep convolutional neural networks for fast and low power mobile applications," in *ICLR*, 2016.