Quantized CNN: A Unified Approach to Accelerate and Compress Convolutional Networks

Jian Cheng, Jiaxiang Wu, Cong Leng, Yuhang Wang, and Qinghao Hu

***Abstract*— We are witnessing an explosive development and widespread application of deep neural networks (DNNs) in various fields. However, DNN models, especially a convolutional neural network (CNN), usually involve massive parameters and are computationally expensive, making them extremely dependent on high-performance hardware. This prohibits their further extensions, e.g., applications on mobile devices. In this paper, we present a quantized CNN, a unified approach to accelerate and compress convolutional networks. Guided by minimizing the approximation error of individual layer's response, both fully connected and convolutional layers are carefully quantized. The inference computation can be effectively carried out on the quantized network, with much lower memory and storage consumption. Quantitative evaluation on two publicly available benchmarks demonstrates the promising performance of our approach: with comparable classification accuracy, it achieves 4 to 6× acceleration and 15 to 20× compression. With our method, accurate image classification can even be directly carried out on mobile devices within 1 s.**

***Index Terms*— Acceleration and compression, convolutional neural network (CNN), mobile devices, product quantization.**

I. Introduction

RECENT years' convolutional neural networks (CNNs) [1] have been widely applied in computer vision community, e.g., visual categorization [2], [3], object localization [4], [5], and semantic segmentation [6], [7]. The proposal of deeper networks, from AlexNet [2] (8-layer) to ResNet [8] (152-layer), has dramatically boosted the performance in various visual recognition tasks. Nevertheless, the complexity of networks will increase sharply when it becomes deeper, imposing even higher demand on the computation ability for both model training and deployment.

Take AlexNet as an example [2]; it consists of eight layers involving 60M weights, which needs more than 729M FLOPs[1] for an inference. The deeper 16-layer VGG-16 [3] even contains 138M parameters and requires around 15.5G FLOPs. Although these networks can been efficiently trained with GPU servers, such prohibitive computation complexity is still unaffordable for most PCs and mobile devices. Although mobile applications can greatly benefit from accurate visual recognition, it is often intractable to deploy convolutional networks on mobile devices due to their limited computation resources. Therefore, it is of central importance to accelerate and compress convolutional networks for practical applications.

For most CNNs, the convolutions account for the majority of computational complexity while fully connected parts occupy most of the network parameters. Since these two kinds of layers are distinctly different, most of existing works only focus on addressing either of them. Low-rank decomposition (LRD) was adopted in [9]–[14] to factorize the convolutional layer into multiple smaller layers. Lebedev and Lempitsky [15], and Anwar *et al.* [16] introduced sparsity into convolutional layers to reduce the computation complexity. Matrix and tensor decomposition were explored in [10] and [17] to compress fully connected layers. Other approaches included fixed-point representation [18]–[20], binary-valued network [21]–[24], codebook-based quantization [25]–[27], and compact network design [28]–[31].

In this paper, we present a quantized CNN (Q-CNN), a unified approach to simultaneously compress and speed up CNNs with negligible accuracy loss. For each layer of a CNN, its parameters are split into subvectors and then quantized with the corresponding subcodebooks. After quantization, the layer response can be efficiently calculated using approximate inner product computation. Errors of each layer's approximated response are minimized when learning the parameter quantization, so as to better preserve the model performance. To better quantize multiple layers, a layer-by-layer quantizing manner is presented to efficiently consider previously introduced

Manuscript received June 11, 2016; revised December 15, 2016, September 21, 2017, and November 4, 2017; accepted November 4, 2017. Date of publication December 8, 2017; date of current version September 17, 2018. This work was supported in part by the National Natural Science Foundation of China under Grant 61332016, in part by the Scientific Research Key Program of Beijing Municipal Commission of Education under Grant KZ201610005012, in part by the Fund of Hubei Key Laboratory of Transportation Internet of Things, and in part by the Fund of Jiangsu Key Laboratory of Big Data Analysis Technology. *(Corresponding author: Jian Cheng.)*

J. Cheng is with the Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China, also with the University of Chinese Academy of Sciences, Beijing 100190, China, and also with the CAS Center for Excellence in Brain Science and Intelligence Technology, Beijing 100190, China (e-mail: jcheng@nlpr.ia.ac.cn).

J. Wu was with the Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China, and also with the University of Chinese Academy of Sciences, Beijing 100190, China. He is now with the Tencent AI Lab, Machine Learning Group, Shenzhen 518000, China (e-mail: jonathanwu@tencent.com).

C. Leng and Q. Hu are with the Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China, and also with the University of Chinese Academy of Sciences, Beijing 100190, China (e-mail: cong.leng@nlpr.ia.ac.cn; qinghao.hu@nlpr.ia.ac.cn).

Y. Wang was with the Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China, and also with the University of Chinese Academy of Sciences, Beijing 100190, China. He is now with UISEE Technologies (Beijing) Ltd., Beijing 102402, China (e-mail: yuhang.wang@uisee.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2017.2774288

[1]FLOPs: number of FLoating-point OPerations.

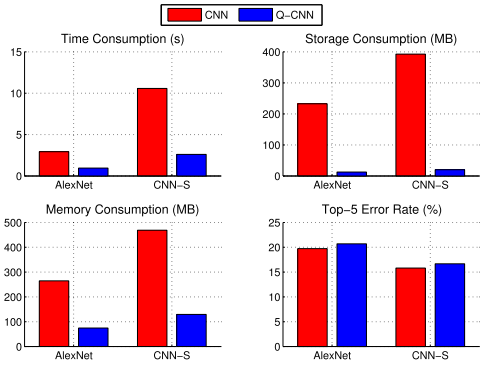


Fig. 1. Performance and resource consumption of the original CNN networks (red bar) and their Q-CNN version (blue bar) on mobile device (Huawei Mate 7).

approximation error, so that the accumulative error can be well suppressed. Our approach enables fast inference, together with significantly smaller storage and memory consumption.

We evaluate the Q-CNN approach on two image classification data sets, MNIST [32] and ILSVRC-12 [33]. On the MNIST data set, the proposed Q-CNN gets more than 12 times compression ratio for multilayer perceptron (MLP) networks, and the accuracy loss is consistently lower than other baselines. For ILSVRC-12, we verify Q-CNN's effectiveness of four classic CNN models: AlexNet [2], CaffeNet [34], CNN-S [35], and VGG-16 [3]. In general, the proposed method yields over 15× compression and 4× speed-up, while the top-5 accuracy loss is strictly controlled within 1%. For further demonstration, the Q-CNN is implemented on mobile devices, which significantly reduce the computational complexity (see Fig. 1). This paper's contributions can be summarized as follows.

1) A Q-CNN approach is proposed to simultaneously compress and speed up CNN models via parameter quantization.
2) We demonstrate that it is more effective to mimic the original network's response during parameter quantization, and the accumulative error can be well suppressed via a layer-by-layer training scheme.
3) The proposed Q-CNN approach obtains over 15× compression and 4× acceleration ratio, with no more than 1% loss in the classification accuracy, which enables complex CNNs affordable for power-constrained equipment.

A preliminary work of this paper has been published in CVPR 2016 [36]. Compared with the preliminary work, this paper provides more theoretical and technical analysis for the quantization process, and new experimental results demonstrate the effectiveness of our error correction training scheme. We also review and compare Q-CNN with more recent methods in accelerating and compressing CNN models.

II. Related Work

CNNs have been largely adopted in many computer vision applications because of powerful feature representation. However, highly redundant network parameters impose severe computation and storage overhead for both training and deployment. Several methods have been proposed to accelerate and compress CNN models, and shall be briefly reviewed in this section, categorized by LRD, connection pruning, low-precision representation, codebook-based quantization, and compact network design.

A. Low-Rank Decomposition

The basic idea is to decompose each convolutional and/or fully connected layer into the combination of two or more smaller layers for more efficient computation and storage.

For the convolutional layer, Jaderberg *et al.* [9] proposed two schemes to replace each convolutional layer by two simpler layers with less computation. Denton *et al.* [10] treated convolutional kernels as a 3-D tensor (by folding the spatial dimension), and approximated it with the sum of multiple rank-1 tensors for speed-up. In the same way, Lebedev *et al.* [11] adopt canonical polyadic (CP) decomposition to approximate the 4-D convolutional kernel tensor with the sum of multiple rank-1 tensors. Zhang *et al.* [12], [13] extended LRD by leveraging subsequent nonlinear activation functions, and minimized the reconstructed error of layer response. Wang and Cheng [14] adopted the tensor block-term decomposition to further speed up convolutional layers.

For the fully connected layer, Denton *et al.* [10] proposed to approximate the weighting matrix as the multiplication of two low-rank matrices, to reduce the parameter redundancy. Novikov *et al.* [17] introduced tensor-train decomposition to compress the weighting matrix, in order to enable fast forward/backward computation and efficient parameter storage.

B. Connection Pruning

By removing certain connections from the network, the computation complexity can be reduced correspondingly. Early exploration includes the combination of the sparsity constraint with unsupervised neural network models, e.g., sparse autoencoder [37] and sparse deep belief network [38].

Ciresan *et al.* [39] presented a CNN variant with sparse random connectivity for more efficient computation and fewer parameters. Lebedev and Lempitsky [15] proposed the group-wise brain damage (GBD) approach to impose group-sparsity regularization to convolutional kernels during training. As a result, convolutions were reduced to multiplications of thinned dense matrices for speeding up. Srinivas and Babu [40] proposed a data-free parameter pruning method by detecting and removing similar neurons, instead of individual connections, for better acceleration. Structured sparsity was explored in [16] at various scales (channel/kernel/intrakernel) for more advantageous computational resource saving. Liu *et al.* [41] exploited the interchannel and intrachannel redundancy via the two-stage decompositions. Han *et al.* [42] proposed a three-stage pruning method: first identify important connections in the network, then remove unimportant connection, and finally retrain the network with those remaining connections. Sun *et al.* [43] presented a layerwise training scheme to introduce sparsity

into the network, and obtained good performance for face recognition with a moderately sparse network.

C. Low-Precision Representation

Currently, 32-b floating-point numbers are widely used to represent connection weights in CNN models. However, arithmetic operations with fixed-point numbers are usually more friendly for hardware acceleration.

Gupta *et al.* [18] demonstrated that it was possible to train deep networks with 16-b fixed-point number representation, while the classification accuracy degradation was negligible. In [19], deep neural networks were trained with low-precision multipliers and high-precision accumulators. A dynamic fixed-point format was proposed to allow various data ranges of connection weights for more accurate approximation. Logarithmic data representation was studied in [20] to better fit the nonuniform distribution of network parameters and activations. For low-power embedded devices, such as FPGAs and microcontrollers, fixed-point representation has been extensively studied to improve the computational efficiency [44]–[46].

Another trend is to train neural networks with binarized weights and/or activations. In [21] and [22], network weights were binarized to convert multiplications to additions during the forward and backward propagations. Courbariaux and Bengio [23] further restricted both weights and activations to be binary-valued to fully exploit the efficient bitwise operation. Rastegari *et al.* [24] introduced a scaling factor to the binarization, and evaluated two binary-valued networks (weights only/weights and activations) on ImageNet.

D. Codebook-Based Quantization

Similar to the above fixed-point and binarization methods, codebook-based quantization can also reduce the bit length to represent each network weight. The basic idea is to quantize each network parameter (or vector of parameters) as an element drawn from a finite set, i.e., the codebook, so as to discard the original value for model compression.

Hashing is a representative method of efficient approximate nearest neighbor search for large-scale databases [47], [48], and has been extended into the compression of neural networks. Chen *et al.* [26] proposed to compress neural networks by randomly grouping weights of networks into different hash buckets, and then encode all weights in the same bucket into a single value. In [49], this idea was further extended by grouping parameters in the frequency domain. Han *et al.* [27] adopted a three-stage approach to model compression, i.e., connection pruning followed by quantization and Huffman encoding.

The above methods quantized network parameters one at a time, while in [25], network parameters were first concatenated into subvectors and then quantized. This approach is similar to ours, but still differs in several aspects. In [25], the quantization was only applied to fully connected layers, and the inference computation was based on the reconstructed parameters with no acceleration. In contrast, our approach is applicable to the whole network including fully connected and convolutional layers. Moreover, the inference computation can be directly carried out on quantized parameters, which is more efficient than the original one. In fact, the method by Gong and Lazebnik [50] can be considered as a special case of our methods, i.e., without handling convolutional layers and no accelerated computation based on the quantized parameters.

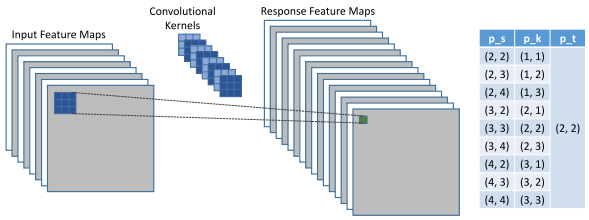


Fig. 2. Computation process of one response neuron located at p_t in the convolutional layer. The response value is determined by the sum of inner products of input feature map vectors and convolutional kernel vectors at a series of spatial positions, as specified by p_s and p_k. Here, we demonstrate 3×3 convolution with no padding and the stride size is 1.

E. Compact Network Design

Apart from accelerating and compressing a network without changing its overall structure, some methods attempt to train a smaller network, while still preserving comparable performance with the larger one.

Ba and Caruana [28] extended the idea of model compression in [51], and demonstrated that the shallow network could be more accurately trained by mimicking the output of a deep network (or an ensemble of deep networks). Similar idea was adopted in [29], where the knowledge hidden in an ensemble of networks was distilled into a single model. Romero *et al.* [30] used both the teacher network's outputs and intermediate layers' activations as the supervise information to train a deeper but faster student network. Iandola *et al.* [31] designed an extremely small network ($50\times$ fewer parameters than AlexNet), but still achieved the same accuracy level.

III. Preliminary

In this section, we analyze the computation operations of the convolutional layer as well as the fully connected layer. Then, we demonstrate that product quantization can be used to speed up the inner product computation, which is the most time-consuming operation of these two layers.

A. Convolutional Neural Networks

For most CNN models, its major computation time is spent on convolutional layers while most parameters come from fully connected layers. Hence, for efficient inference, it is crucial to accelerate computation in the convolutional layers and, at the same time, reduce parameters in fully connected layers.

In fact the computation process of all layers can be considered as inner product computation. To see this, we denote $S \in \mathbb{R}^{d_s \times d_s \times C_s}$ input feature maps of one convolutional layer and $T \in \mathbb{R}^{d_t \times d_t \times C_t}$ the response feature maps, where d_s and d_t are the spatial sizes and C_s and C_t are the corresponding channel quantities. As depicted in Fig. 2, for the 2-D spatial

region p_t in the c_tth channel of response feature maps, its response is computed as

$$T_{p_t}(c_t) = \sum_{(p_k, p_s)} \langle W_{c_t, p_k}, S_{p_s} \rangle \tag{1}$$

where $W_{c_t} \in \mathbb{R}^{d_k \times d_k \times C_s}$ denotes the c_tth kernel of this convolutional layer and d_k the kernel size.[2] p_s and p_k are the 2-D spatial positions in the feature maps and convolutional kernels, respectively. Please note that both W_{c_t, p_k} and S_{p_s} have dimensionality C_s. Then the response of layer is actually the summation of the inner product at every spatial regions in this $d_k \times d_k$ receptive field of input feature maps.

In a similar way, the fully connected layer can be formulated as follows:

$$T(c_t) = \langle W_{c_t}, S \rangle \tag{2}$$

where $S \in \mathbb{R}^{C_s}$ and $T \in \mathbb{R}^{C_t}$ are the input and response vectors of this layer. $W_{c_t} \in \mathbb{R}^{C_s}$ is the weight corresponding to the c_tth position of the layer's response.

B. Product Quantization for Inner Product Computation

Product quantization [52] is a commonly used technique in approximate nearest neighbor search, which can usually offer much higher retrieval accuracy than approaches based on hashing [53], [54]. Intrinsically, product quantization treats the feature space as the Cartesian product space composed by multiple subspaces. In each subspace, we learn a subcodebook. Thus, vectors in the original feature space can be approximately represented by the concatenation of subcodewords, which facilitates efficient similarity computation and parameter storage.

Here, product quantization is introduced to accelerate the computation of inner products. Consider the inner products between vector y and a collection of vectors $\{x_i\}_{i=1}^N$, where each vector is of D-dimension. First, each vector is parted into M subvectors

$$y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(M)} \end{bmatrix}, \quad x_i = \begin{bmatrix} x_i^{(1)} \\ \vdots \\ x_i^{(M)} \end{bmatrix} \tag{3}$$

where each subvector is of D/M-dimension. Afterward, we quantize the mth subvector in x_i by a subcodeword selected from the corresponding pretrained subcodebook $\mathcal{C}^{(m)}$. After that, the inner product can be approximately computed as

$$\begin{aligned} \langle y, x_i \rangle &= \sum_m \langle y^{(m)}, x_i^{(m)} \rangle \\ &\approx \sum_m \left\langle y^{(m)}, c_{k_{i,m}}^{(m)} \right\rangle, \quad \text{s.t. } c_{k_{i,m}}^{(m)} \in \mathcal{C}^{(m)}. \end{aligned} \tag{4}$$

Following this procedure, computation of inner product, which asks for $\mathcal{O}(D)$ multiplication and addition, is reduced to M addition operations, given that all the inner products between subcodewords in $\mathcal{C}^{(m)}$ and subvector $y^{(m)}$ have already been computed and stored in advance. The overall computation complexity is reduced from $\mathcal{O}(ND)$ to $\mathcal{O}(KD + NM)$, where K is subcodeword quantity in each subcodebook. This indicates that with proper parameters (M and K) chosen, product quantization can indeed speed up the inner product computation.

[2]For simplicity, here, we assume that no feature map grouping is used, so that the number of channels in the convolutional kernel and input feature maps are the same. Our approach remains valid when feature map grouping is used (as in AlexNet and CaffeNet).

IV. Proposed Q-CNN

Here, a novel Q-CNN approach is presented to compress and accelerate CNNs. We will present an efficient inference process once model parameters have been quantized. Afterward, we show that directly minimizing the approximation error of each layer's response leads to better quantization. Finally, we present the complexity analysis on the time and memory overhead.

A. Quantization of Fully Connected Layer

We use C_s and C_t to denote inputs and outputs of a fully connected layer. The weight matrix is written as $W \in \mathbb{R}^{C_s \times C_t}$. The weight vector W_{c_t}, which accounts for the c_tth response neuron, is the c_tth column in W. To obtain the layer response, one needs to compute the inner products between the layer input S and all the weighting vectors $\{W_{c_t}\}$. As we have already shown above, such computation can be accelerated by quantizing $\{W_{c_t}\}$ in advance.

Specifically, we evenly split the C_s-dimensional space into M subspaces, each of $C_s' = C_s/M$ dimensions. Each weighting vector W_{c_t} is correspondingly parted into M subvectors, where the mth subvector is denoted as $W_{c_t}^{(m)}$. For each subspace, a subcodebook is learned by clustering all elements(subvectors) in that subspace. We can formulate the process as follows:

$$\begin{aligned} &\min_{D^{(m)}, B^{(m)}} \|D^{(m)} B^{(m)} - W^{(m)}\|_F^2 \\ &\text{s.t. } D^{(m)} \in \mathbb{R}^{C_s' \times K}, \quad B^{(m)} \in \{0, 1\}^{K \times C_t} \end{aligned} \tag{5}$$

where $W^{(m)} \in \mathbb{R}^{C_s' \times C_t}$ is comprised of the mth subvectors of the original weight vectors. $D^{(m)}$, the subcodebook, consists of K subcodewords, and $B^{(m)}$ is the indexing matrix, each column of whom only has one nonzero entry, specifying the quantization relationship between subvector and subcodewords. The optimization is equivalent to k-means clustering.

After the subspace decomposition, the layer response can be computed as

$$T(c_t) = \langle W_{c_t}, S \rangle = \sum_m \langle W_{c_t}^{(m)}, S^{(m)} \rangle \tag{6}$$

where $S^{(m)}$ denotes the mth input subvector of this layer. It is obvious that there are many replicate elements in $\{W_{c_t}^{(m)}\}_{c_t=1}^{C_t}$ after the parameter quantization. Therefore, it is unwise to compute the inner products in a one-by-one style. Instead, we first calculate the inner product between each subcodeword in subcodebook $D^{(m)}$ and input subvectors $S^{(m)}$

$$L_k^{(m)} = \langle D_k^{(m)}, S^{(m)} \rangle \tag{7}$$

to constitute the lookup table $L^{(m)} \in \mathbb{R}^K$. Afterward, the response of the layer is roughly equivalent to

$$\begin{aligned} T(c_t) &\approx \sum_m \left\langle D^{(m)} B_{c_t}^{(m)}, S^{(m)} \right\rangle = \sum_m \left\langle B_{c_t}^{(m)}, D^{(m)^T} S^{(m)} \right\rangle \\ &= \sum_m \left\langle B_{c_t}^{(m)}, L^{(m)} \right\rangle \end{aligned} \tag{8}$$

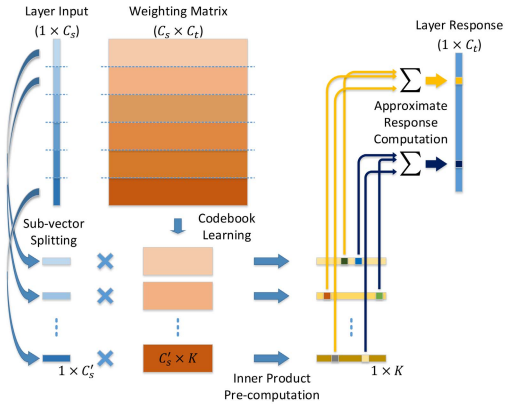


Fig. 3. Quantization and acceleration computation processes for a fully connected layer. During training, weights are split into submatrices, and then quantized with the subcodebook in the corresponding subspace. In the inference stage, input should be split into subvectors, in alignment with the partition of weights, and then multiplied by the corresponding subcodebook to obtain lookup tables. Finally, the layer response is approximately computed by accumulating elements in lookup tables, guided by quantization indicators.

where $B^{(m)}_{c_t}$ is the c_tth column in indexing $B^{(m)}$. The first line in (8) holds since the weighting subvector $W^{(m)}_{c_t}$ is quantized as $D^{(m)}B^{(m)}_{c_t}$. Because $B^{(m)}_{c_t}$ is the indicator vector who has only one nonzero entry, its inner product with $L^{(m)}$ can be directly obtained via a lookup operation. Hence, the approximate layer response computation only involves M addition operations.

In Fig. 3, we demonstrate the fully connected layer's parameter quantization and the inference computation process. After the weight matrix is decomposed into M submatrices, one subcodebook can be learned for each subspace. During the inference stage, the input vector of this layer is also parted into M subvectors. In each subspace, the inner products between the input subvector and all subcodewords of the corresponding subcodebook are computed and stored in a lookup table, which allows each response value to be approximately computed with M addition operations.

B. Quantizing the Convolutional Layer

To compute a response value in the fully connected layer, a 1-D weighting vector is involved. However, in the convolutional layer, a response value is determined by a 3-D convolutional kernel, as depicted in Fig. 2. Therefore, we need to determine to which dimension the subspace splitting should be applied, so as to minimize the computation cost.

During the forward-passing process, convolutional kernels sweep through the input feature maps spatially. We denote the c_tth convolutional kernel as $W_{c_t} \in \mathbb{R}^{d_k \times d_k \times C_s}$, where d_k is the kernel size, and C_s denotes the quantity of input feature map channels. The selected feature patch (with size $d_k \times d_k \times C_s$) is correspondingly decomposed into subvectors to compute the inner product lookup table. We choose to perform the subspace splitting along the input feature map channels dimension, resulting in that the whole input feature maps can be decomposed into subvectors ahead to compute a unified lookup table. Since the sliding windows are partially overlapped in the spatial domain, the lookup table can be reused for different feature patches to avoid redundant computation. Fig. 4 visualizes the quantization and computation process in the convolutional layer.

Formally, quantization can be achieved in each subspace by minimizing the following objective:

$$\min_{D^{(m)},\{B^{(m)}_{p_k}\}} \sum_{p_k} \left\| D^{(m)}B^{(m)}_{p_k} - W^{(m)}_{p_k} \right\|_F^2$$
$$\text{s.t. } D^{(m)} \in \mathbb{R}^{C'_s \times K}, \quad B^{(m)}_{p_k} \in \{0,1\}^{K \times C_t} \tag{9}$$

where $W^{(m)}_{p_k} \in \mathbb{R}^{C'_s \times C_t}$ consists of all the mth subvectors in each kernels at spatial position p_k. Optimization is realized by clustering of k-means on all the subvectors of convolutional kernels that belong to the mth subspace, regardless of their spatial positions.

For the c_tth response feature map, its response value at position p_t can be computed as the sum of multiple inner products of subvectors

$$T_{p_t}(c_t) = \sum_{(p_k,p_s)} \langle W_{c_t,p_k}, S_{p_s} \rangle = \sum_{(p_k,p_s)} \sum_m \left\langle W^{(m)}_{c_t,p_k}, S^{(m)}_{p_s} \right\rangle \tag{10}$$

where $S^{(m)}_{p_s}$ denotes the mth subvector of input feature maps at the spatial position p_s. There will be a massive number of replicate subvectors in convolutional kernels after the parameter quantization. Similarly, we calculate and store in advance an inner product between $S^{(m)}_{p_s}$ and each subcodebook of $D^{(m)}$

$$L^{(m)}_{p_s,k} = \left\langle D^{(m)}_k, S^{(m)}_{p_s} \right\rangle \tag{11}$$

to constitute the lookup table $L^{(m)}_{p_s} \in \mathbb{R}^K$. Then, the response value can be approximately estimated as

$$\begin{aligned} T_{p_t}(c_t) &\approx \sum_{(p_k,p_s)} \sum_m \left\langle D^{(m)}B^{(m)}_{c_t,p_k}, S^{(m)}_{p_s} \right\rangle \\ &= \sum_{(p_k,p_s)} \sum_m \left\langle B^{(m)}_{c_t,p_k}, D^{(m)^T}S^{(m)}_{p_s} \right\rangle \\ &= \sum_{(p_k,p_s)} \sum_m \left\langle B^{(m)}_{c_t,p_k}, L^{(m)}_{p_s} \right\rangle \end{aligned} \tag{12}$$

where the approximation in the first line in (12) is due to quantization. Since $B^{(m)}_{c_t,p_k}$ contains only one nonzero entry, its inner product with $L^{(m)}_{p_s}$ is obtained with a single lookup operation. In summary, the approximate computation of each response value requires $d_k^2 M$ additions, where d_k^2 is contributed by the first sum operator in (12).

C. Discussion

So far, we have presented a naive approach to quantize parameters for more efficient computation in the inference stage. However, the above quantization method may not be optimal due to two critical issues. First, the minimization of network parameters' quantization error may not necessarily lead to the optimal network in terms of performance. In contrast, optimization of the approximation error of all the layers' response seems to be more relevant to the classification accuracy. Second, each layer's parameters are quantized

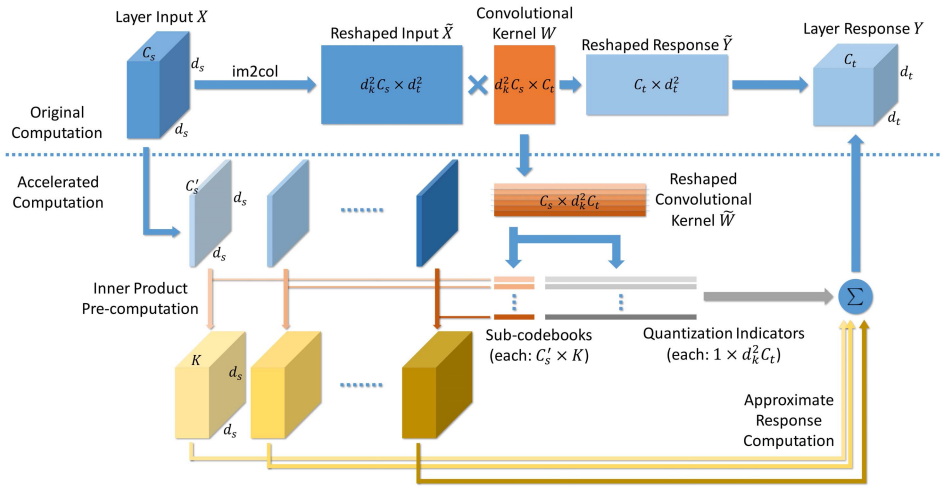


Fig. 4. Quantization and acceleration computation processes for the convolutional layer. For the original computation, the layer input X is first reshaped into the matrix form, and then multiplied by the convolutional kernel W to obtain the layer response Y. For the accelerated computation, the convolutional kernel W is reshaped and split into submatrices along the C_s dimension, and quantized with the subcodebook in each subspace. In the test-phase, layer input X is also split along the C_s dimension, and each resulting part is multiplied by the corresponding subcodebook to obtain a lookup table. Finally, the layer response is approximated by accumulating elements in lookup tables, guided by quantization indicators.

independently, which could result in the accumulative error if multiple layers are quantized. Due to the layer-by-layer accumulation of the error, the final error of the whole network is probably out of scope.

To overcome the above drawbacks, error correction is proposed for better network parameters' quantization. In short, we directly minimize the approximation error of the layer response instead of the quantization error of layer parameters. This approach can easily consider the error caused by previous quantized layers, so as to compensate such an error when quantizing the following layers. Next, we present details how error correction can be carried out in both fully connected and convolutional layers, and how the accumulative error can be suppressed when multiple layers are quantized.

D. Fully Connected Layer With Error Correction

We can control the approximated layer error by mimicking the input-response relationship of a set of N images. For the nth image, its layer input is denoted as S_n and its layer response is denoted as T_n. The response's approximation error is minimized via

$$\min_{\{D^{(m)}\},\{B^{(m)}\}} \sum_n \left\| T_n - \sum_m (D^{(m)} B^{(m)})^T S_n^{(m)} \right\|_F^2 \tag{13}$$

where the former term T_n is the desired layer response, while the latter one is the approximated response after quantization. The idea is to minimize the overall difference between the desired and approximated layer responses, measured by the Frobenius norm.

The above optimization problem can be efficiently solved by a block coordinate descent approach, i.e., alternatively updating the quantization parameters in each subspace. The residual error of the mth subspace for the nth image can be formulated as

$$R_n^{(m)} = T_n - \sum_{m' \neq m} (D^{(m')} B^{(m')})^T S_n^{(m')}. \tag{14}$$

With parameters in all the other subspaces fixed, the optimal subcodebook and quantization indicators in this subspace should minimize the overall residual error

$$\min_{D^{(m)}, B^{(m)}} \sum_n \left\| R_n^{(m)} - (D^{(m)} B^{(m)})^T S_n^{(m)} \right\|_F^2 \tag{15}$$

here, we utilize the alternative updating approach to calculate the subcodebook $D^{(m)}$ as well as quantization indicators $B^{(m)}$.

Update $D^{(m)}$: Fixing the quantization indicators $B^{(m)}$, we assume $H_k = \{c_t | B^{(m)}(k, c_t) = 1\}$, which is the collection of indexes that the corresponding subvectors are quantized with the kth subcodeword in $D^{(m)}$. The objective of (15) can then be converted into

$$\min_{\{D_k^{(m)}\}} \sum_{n,k} \sum_{c_t \in H_k} \left[R_n^{(m)}(c_t) - D_k^{(m)^T} S_n^{(m)} \right]^2 \tag{16}$$

which indicates independence in the optimization of different subcodewords. Hence, we have

$$\min_{D_k^{(m)}} \sum_{n} \sum_{c_t \in H_k} \left[R_n^{(m)}(c_t) - D_k^{(m)^T} S_n^{(m)} \right]^2 \tag{17}$$

which is a least square problem that can be easily solved.

Update $B^{(m)}$: Fixing $D^{(m)}$, it is obvious that the column of $B^{(m)}$ can be optimized independently. For the c_tth one, we have

$$\min_{B_{c_t}^{(m)}} \sum_{n} \left[R_n^{(m)}(c_t) - B_{c_t}^{(m)^T} D^{(m)^T} S_n^{(m)} \right]^2 \tag{18}$$

under the constraint that $B_{c_t}^{(m)}$ is an indicator vector. This can be solved by exhaustively checking all K possibilities, and then choosing the one with the smallest error.

E. Convolutional Layer With Error Correction

A similar idea can be applied to the quantization of convolutional layer's parameters, which is

$$\min_{\substack{\{D^{(m)}\}\\ \{B_{p_k}^{(m)}\}}} \sum_{n,p_t} \left\| T_{n,p_t} - \sum_{(p_k,p_s)} \sum_{m} \left(D^{(m)} B_{p_k}^{(m)}\right)^T S_{n,p_s}^{(m)} \right\|_F^2. \quad (19)$$

Again, we sequentially update parameters in each subspace to solve the above problem. In the mth subspace, we first define the residual error on the nth image's response feature maps at position p_t as

$$R_{n,p_t}^{(m)} = T_{n,p_t} - \sum_{(p_k,p_s)} \sum_{m' \neq m} \left(D^{(m')} B_{p_k}^{(m')}\right)^T S_{n,p_s}^{(m')} \quad (20)$$

and then the optimization is converted into the following:

$$\min_{D^{(m)},\{B_{p_k}^{(m)}\}} \sum_{n,p_t} \left\| \sum_{(p_k,p_s)} \left(D^{(m)} B_{p_k}^{(m)}\right)^T S_{n,p_s}^{(m)} - R_{n,p_t}^{(m)} \right\|_F^2. \quad (21)$$

Update $D^{(m)}$: With the quantization indicators fixed, we define $H_{k,p_k} = \{c_t | B_{p_k}^{(m)}(k,c_t) = 1\}$, which includes indexes of all the subvectors that are currently quantized with the kth subcodeword in the subcodebook $D^{(m)}$. Afterward, we greedily update each subcodeword in a sequential style. For the kth subcodeword, we compute its residual term as

$$Q_{n,p_t,k}^{(m)}(c_t) = R_{n,p_t}^{(m)}(c_t) - \sum_{(p_k,p_s)} \sum_{k' \neq k} \sum_{c_t \in H_{k',p_k}} D_{k'}^{(m)^T} S_{n,p_s}^{(m)} \quad (22)$$

and then update it with

$$\min_{D_k^{(m)}} \sum_{n,p_t} \left\| \sum_{(p_k,p_s)} \sum_{c_t \in H_{k,p_k}} D_k^{(m)^T} S_{n,p_s}^{(m)} - Q_{n,p_t,k}^{(m)}(c_t) \right\|_F^2 \quad (23)$$

which is also equivalent to a least square problem.

Update $\{B_{p_k}^{(m)}\}$: By fixing the subcodebook, we sequentially update the quantization indicator at each spatial position in the convolutional kernel. For the spatial position p_k, we compute the corresponding residual error as

$$P_{n,p_t,p_k}^{(m)} = R_{n,p_t}^{(m)} - \sum_{\substack{(p_k',p_s')\\ p_k' \neq p_k}} \left(D^{(m)} B_{p_k'}^{(m)}\right)^T S_{n,p_s'}^{(m)} \quad (24)$$

which leads to the following optimization problem:

$$\min_{B_{p_k}^{(m)}} \sum_{n,p_t} \left\| \left(D^{(m)} B_{p_k}^{(m)}\right)^T S_{n,p_s}^{(m)} - P_{n,p_t,p_k}^{(m)} \right\|_F^2. \quad (25)$$

Similarly, since $B_{p_k}^{(m)}$ is an indicator vector, we exhaustively evaluate each subcodeword and select the optimal one that minimizes the objective function.

F. Accumulative Error in Multiple Layers

Multiple layers can be independently quantized with the above quantization methods. Nevertheless, such an approach fails to capture the interlayer relationship and may suffer from severe accumulative error when more layers are quantized. The quantization of previous layers will change the input to subsequent layers, so that the layer response will deviate from the original one even if the parameters are not quantized. With parameter quantization, the deviation from the desired layer response can be even larger.

In this section, a layer-by-layer learning manner is presented to solve the above-mentioned problem. For a certain layer, we assume that its upstreaming layers have been quantized. For the nth training image, we denote its layer's input of this quantized network as S_n, and the response of this layer in the original network (without quantization) as T_n. The quantized parameters can be learned via solving the optimization in (13) and (19), so that the quantized layer's response can maximally approximate the original layer response.

In this way, the quantized network attempts to recover the response of each layer in the original model with the quantized parameters, which means that the error caused by previous approximation is considered when quantizing the current layer. In consequence, the accumulative error can be effectively suppressed to allow more accurate quantization for multiple layers.

An alternative choice is to apply back propagation to simultaneously update quantization indicators and subcodebooks for all quantized layers. However, due to the $0/1$ constraint on the quantization indicators, the gradient-based optimization may be challenging. If we only update the subcodebooks during the backpropagation, while the quantization indicators remain unchanged, we may not be able to obtain a sufficiently good approximation to the original network. Therefore, we choose to adopt the former learning scheme to quantize multiple layers.

G. Complexity Analysis

Next, we try to analyze and compare the inference computational complexity of the original CNNs and its quantized counterpart.

For Q-CNN models, the computation in a layer mostly consists of two parts: the computation of the inner product and the approximated layer response computation. For each layer, both quantization indicators and subcodebooks, instead of the original parameters, are kept to reduce the network's storage overhead. The test-phase computation can be directly carried out on the quantized parameters, so the run-time memory consumption is also much smaller than the original network. The inner product lookup table does require extra memory, which is relatively small and can be reused by all layers to save space.[3]

Tables I and II depict the detailed complexity analysis of the original CNN and its quantized version on the storage

[3]The run-time memory consumption also involves other temporary variables, i.e., feature maps of each layer. These are omitted in the comparison, since they are the same for both CNN and Q-CNN models.

TABLE I

COMPUTATION AND STORAGE COMPLEXITY IN THE CONVOLUTIONAL LAYER OF CNN AND Q-CNN

CNN	FLOPs	$d_t^2 C_t d_k^2 C_s$	
	Bytes	$4 d_k^2 C_s C_t$	
Q-CNN	FLOPs	inner product	$d_s^2 C_s K$
		approx. response	$d_t^2 C_t d_k^2 M$
	Bytes	sub-codebooks	$4 C_s K$
		quan. indicators	$\frac{1}{8} d_k^2 M C_t \log_2 K$
		look-up tables[4]	$4 d_s^2 M K$

TABLE II

COMPUTATION AND STORAGE COMPLEXITY IN THE FC LAYER OF THE CNN AND THE Q-CNN

CNN	FLOPs	$C_s C_t$	
	Bytes	$4 C_s C_t$	
Q-CNN	FLOPs	inner product	$C_s K$
		approx. response	$C_t M$
	Bytes	sub-codebooks	$4 C_s K$
		quan. indicators	$\frac{1}{8} M C_t \log_2 K$
		look-up tables	$4 M K$

and computation. For the Q-CNN, its computation and storage overhead is largely relevant to the number of subcodewords in each subspace (K) and the number of slitted subspaces (M). Larger K and M may result in better quantization, yet less efficient. In practice, the two hyperparameters can be tuned to keep a balance between the effectiveness and efficiency of the quantized networks. We shall discuss the acceleration and compression effects under various hyperparameter settings in the following experiments.

V. EXPERIMENTS

To verify its effectiveness, the proposed Q-CNN approach is evaluated on two image classification benchmarks, MNIST [32] and ILSVRC-12 [33]. For evaluating the acceleration of convolutional layers, we compare the Q-CNN with the following:

1) *CPD [11]:* canonical polyadic decomposition;
2) *GBD [15]:* groupwise brain damage;
3) *LRR [55]:* low-rank regularization;
4) *LANR [13]:* low-rank approximation of nonlinear responses.

Then, to compare the fully connected layers' compression, the following baseline methods are considered:

1) *RER [39]:* random edge removal;
2) *LRD [56]:* low-rank decomposition;
3) *DK [29]:* dark knowledge;
4) *NN-ES [26]:* neural network with equivalent size;
5) *HashNet [26]:* hashed neural nets;
6) *DPP [40]:* data-free parameter pruning;
7) *SVD [10]:* singular-value decomposition;
8) *DFC [57]:* deep fried convnets.

To investigate the whole network acceleration and/or compression, we compare with the following methods:

1) *BC [21]:* binaryconnect;
2) *BWN [24]:* binary weight network;
3) *DC [27]:* deep compression.

For the above compared algorithms, performance with the same setup in original literatures as ours is compared. Here, we only compare the theoretical acceleration ratio, since the realistic acceleration could be relevant to diverse factors, e.g., cache, CPU, and RAM. In Section V-D, we will further analyze the realistic and theoretical acceleration, and discuss the acceleration effect of the BLAS library [58].

TABLE III

COMPARISON OF 3-LAYER MLP (784-1000-10) AND 5-LAYER MLP (784-1000-1000-1000-10) ON MNIST

Method	3-layer		5-layer	
	Compression	Error	Compression	Error
Original	-	1.35%	-	1.12%
RER [39]	8.0×	2.19%	8.0×	1.24%
LRD [56]	8.0×	1.89%	8.0×	1.77%
DK [29]	8.0×	1.71%	8.0×	1.26%
NN-ES [26]	8.0×	1.69%	8.0×	1.35%
HashNets [26]	8.0×	1.43%	8.0×	1.22%
Q-CNN[1]	10.9×	1.42%	13.0×	1.34%
Q-CNN[2]	10.9×	1.39%	13.0×	1.19%

We evaluate two variants of our Q-CNN framework: "**Q-CNN**[1]" and "**Q-CNN**[2]." The difference lies in the optimization objective: the former one aims at minimizing the approximation error of model parameters, while the latter one tries to optimize the layers' response error. We implement the parameter quantization process in MATLAB, and the resulting model can be converted into the Caffe-compatible [34] format for network fine-tuning (FT). The test-phase computation is implemented in C++ for better run-time efficiency, and the code is now publicly available on GitHub at https://github.com/jiaxiang-wu/quantized-cnn.

A. Comparison on MNIST

The MNIST [32] is a benchmark including 60k training images and 10k testing images on handwritten digits. We train one 3-layer and one 5-layer neural network (MLP) as baseline models, of which each hidden layer consists of 1k neurons. Different compression algorithms are tested for these two networks, and the comparison results are reported in Table III.

For the Q-CNN, hyperparameters M (number of subspaces) and K (number of subcodewords per subspace) control the model efficiency and effectiveness. Since $M = C_s / C_s'$ is determined once C_s' is given (C_s is the original network's hyperparameter), we choose a unified (C_s', K) for all layers to ease the parameter tuning workload. In Table III, we have $C_s' = 4$ and $K = 32$.

In Table III, we observe that Q-CNN[2] can achieve consistently highest compression ratios while keeping least performance drop among all competitors. The results demonstrate that our error correction method can eliminate effectively performance degradation, especially when the network goes deeper. The standard deviation of classification error ratios averaging five random runs are only 0.05% for Q-CNN[1] and Q-CNN[2]. For simplicity, it is reasonable to only report single-run results in the next experiments.

Discussion: Here, we discuss a few variants of our Q-CNN method. Although the quantization indicators are discrete, backpropagation can still be used to learn subcodebooks jointly in the whole network. We evaluate this learning strategy on MNIST, and report the results in Fig. 5. Methods with the "BP" suffix are trained via backpropagation. Q-CNN[0]-BP

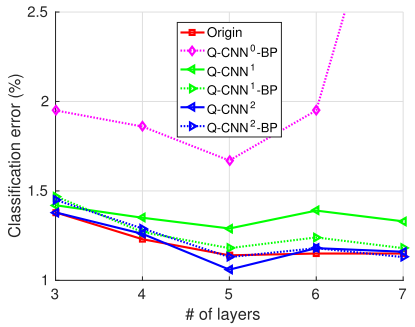


Fig. 5. Comparison on the classification error on MNIST, based on networks with various depths and quantization learning strategies.

TABLE IV
COMPRESSION RATES AND CLASSIFICATION ERROR ON MNIST FOR VARIOUS MODEL DEPTHS (EACH HIDDEN LAYER CONTAINS 1000 NEURONS). "Q-CNN-S" SHARES ONE UNIFIED SUBCODEBOOK ACROSS ALL SUBSPACES, WHILE "Q-CNN" DOES NOT

Model	Method	Para.	Compression	Error
3-layer	Original	-	-	1.37%
	Q-CNN	4/32	10.9×	1.95%
	Q-CNN-s	4/32	24.8×	22.16%
4-layer	Original	-	-	1.22%
	Q-CNN	4/32	12.5×	1.86%
	Q-CNN-s	4/32	25.2×	20.41%
5-layer	Original	-	-	1.13%
	Q-CNN	4/32	13.0×	1.67%
	Q-CNN-s	4/32	25.3×	19.17%

is randomly initialized, while Q-CNN^1-BP and Q-CNN^2-BP are initialized with Q-CNN^1 and Q-CNN^2, respectively.

We observe that directly training via backpropagation from scratch is always inferior to other competitors. After obtaining an initial solution from Q-CNN^1, backpropagation can consistently improve the classification accuracy (see the comparison between Q-CNN^1 and Q-CNN^1-BP). However, this effect is much less significant for Q-CNN^2, as the error correction training scheme is already quite effective. Still, updating subcodebooks via backpropagation shows promising results, especially for deeper networks.

Another possible modification to Q-CNN is to force subcodebooks to be shared across all subspaces, which leads to further model compression. In Table IV, we compare the compression rates and classification error, with subcodebooks shared or not. All the networks are trained from scratch, i.e., updating subcodebooks via backpropagation while keeping quantization indicators fixed. When the subcodebook is shared as in Q-CNN-s, the classification error is much higher than that of the standard Q-CNN, indicating that such a constraint may greatly hurt the network's performance.

B. Comparison on ILSVRC-12

ILSVRC-12 [33] contains 1k categories of images with more than 1 million labelled instances, as well as 50 000 images as validation set. Here, for quantitative comparison, classification errors of both top-1 and top-5 are investigated on the validation set.

We evaluate the proposed Q-CNN approach on four networks: AlexNet [2], CaffeNet [34], CNN-S [35], and VGG-16 [3]. AlexNet and CaffeNet have been investigated in literatures, so that their results will be directly cited for comparison. CNN-S and VGG-16 are relatively complex and representative networks, so we quantize them to demonstrate our approach's scalability. First of all, their basic information is reported in Table V.

TABLE V
COMPARISON OF FOUR NETWORKS ON COMPUTATION COMPLEXITY (FLOPS), STORAGE OVERHEAD (BYTES), AND CLASSIFICATION ERROR (TOP-1/TOP-5)

Model	FLOPs	Bytes	Top-1 Err.	Top-5 Err.
AlexNet [2]	7.29e+8	2.44e+8	42.78%	19.74%
CaffeNet [34]	7.27e+8	2.44e+8	42.53%	19.59%
CNN-S [35]	2.94e+9	4.12e+8	37.31%	15.82%
VGG-16 [3]	1.55e+10	5.53e+8	28.89%	10.05%

TABLE VI
COMPARISON ON ACCELERATION OF THE SECOND CONVOLUTIONAL LAYER OF ALEXNET WITH OR WITHOUT FT. FOR Q-CNN, WE SPECIFY HYPERPARAMETERS (C_s'/K) IN "PARA." COLUMN

Method	Para.	Speed-up	Top-1 Err. ↑		Top-5 Err. ↑	
			No FT	FT	No FT	FT
CPD [11]	-	3.19×	-	-	0.94%	0.44%
	-	4.52×	-	-	3.20%	1.22%
	-	6.51×	-	-	69.06%	18.63%
GBD [15]	-	3.33×	12.43%	0.11%	-	-
	-	5.00×	21.93%	0.43%	-	-
	-	10.00×	48.33%	1.13%	-	-
Q-CNN^1	4/64	3.70×	10.55%	1.63%	8.97%	1.37%
	6/64	5.36×	15.93%	2.90%	14.71%	2.27%
	6/128	4.84×	10.62%	1.57%	9.10%	1.28%
	8/128	6.06×	18.84%	2.91%	18.05%	2.66%
Q-CNN^2	4/64	3.70×	0.35%	0.20%	0.27%	0.17%
	6/64	5.36×	0.64%	0.39%	0.50%	0.40%
	6/128	4.84×	0.27%	0.11%	0.34%	0.21%
	8/128	6.06×	0.55%	0.33%	0.50%	0.31%

1) Quantization of the Convolutional Layer: Regarding of the convolutional layer, it is more critical to improve its test-phase computation speed, rather than reducing the storage consumption. Thus, we focus on investigating acceleration of different methods in this section.

For demonstration, the second convolutional layer in AlexNet is quantized, which consumes a large portion of computation in inference. We evaluate the effectiveness of the error correction training scheme by comparing two Q-CNN variants under different hyperparameter settings, as depicted in Fig. 6.

From Fig. 6, we discover that different parameter settings lead to different speed-up rates, ranging from 3× to 7×. Both lower subspace dimensions (C_s') and more subcodewords per subspace (K) reduce the speed-up rates, but the resulting quantization is of higher quality and the loss in the classification accuracy is smaller. By comparing the response approximation and classification performance, we find that in order to better preserve the network's performance, it is more effective to approximate the layer response rather than its parameters. The error correction scheme in Q-CNN^2 can indeed reduce the approximation error of layer response, so the classification performance is much closer to the original network.

In Table VI, we compare our method with two baselines, CPD [11] and GBD [15], in speeding up the computation of the second convolutional layer of AlexNet.

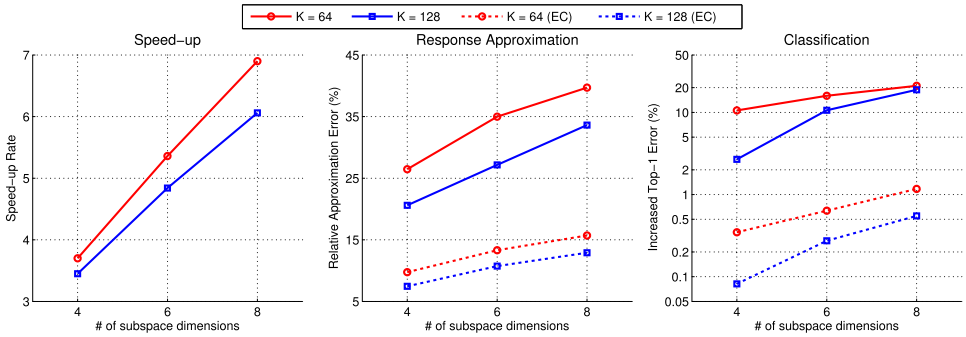


Fig. 6. Comparison on acceleration, response approximation error, and classification performance for the second convolutional layer in AlexNet, under various hyperparameter settings. Since error correction does not affect the acceleration rates, we omit Q-CNN's error correction variant in the first figure.

TABLE VII
COMPARISON OF ACCELERATION OF ALL CONVOLUTIONAL LAYERS IN ALEXNET AND VGG-16, WITH OR WITHOUT FT (FT/NO FT)

Model	Method	Para.	Speed-up	Compression	Top-1 Err. ↑		Top-5 Err. ↑	
					No FT	FT	No FT	FT
AlexNet	LRR [55]	-	5.27×	5.00×	-	-	-	0.37%
	Q-CNN^2	4/64	3.32×	10.58×	1.33%	-	0.94%	-
		6/64	4.32×	14.32×	2.32%	-	1.90%	-
		6/128	3.71×	10.27×	1.44%	0.13%	1.16%	0.36%
		8/128	4.27×	12.08×	2.25%	0.99%	1.64%	0.60%
VGG-16	LRR [55]	-	3.10×	2.75×	-	-	-	0.29%
	LANR [13]	-	4.00×	2.73×	-	-	0.95%	0.35%
	Q-CNN^2	8/128	4.95×	16.75×	2.54%	0.80%	1.35%	0.30%

From Table VI, we can see that sharp acceleration (over 4×) will lead to significant degeneration on the classification precision for CPD, GBD, as well as our Q-CNN^1, especially without fine-tuning. However, with the help of layer response error correction, Q-CNN^2 can achieve up to six-time acceleration with a mild prediction accuracy drop (0.6%). It is worth noting that the accuracy drop can be further decreased if we fine-tune subsequent layers while keeping the quantized parameters untouched.

Next, we attempt to quantize the whole convolution operations for AlexNet and VGG-16 with Q-CNN^2. We use a unified hyperparameter setting (C_s', K) for all layers for simplicity. From Table VII, we can see that the classification precision drops mildly, comparing with the single-layer case. The accumulative error of multiple quantized layers is well suppressed by the error correction training scheme.

For AlexNet, overall acceleration is not as significant as the result shown in Table VI, since some layers, e.g., "conv_4" and "conv_5" are less sensitive. In summary, Q-CNN^2 can accelerate the whole convolutions by around 4.27× ($C_s' = 8$ and $K = 128$), while the performance drop in the top-1/5 is controlled in 2.5%. With fine-tuning, the drop is further suppressed to no more than 1%.

As shown in Table VII, Q-CNN^2 is also compared with LRR [55] and LANR [13]. The classification error rates of all the three methods are very close. Q-CNN^2 offers slightly higher speed-up rates, especially for the deeper VGG-16 network. Meanwhile, our approach provides over 10× compression of convolutional kernels, much larger than those of baseline methods. In conclusion, our Q-CNN^2 approach is effective in accelerating and compressing the convolutional layer(s), and the performance degradation in the classification accuracy is well suppressed when multiple layers are quantized.

2) Quantizing the Fully Connected Layer: To reduce the storage overhead, it is of top priority to decrease weights in fully connected layers that contribute more than 90% parameters of the whole network. To begin with, we demonstrate the parameter compression for a single layer here. For example, the first fully connected layer of CaffeNet contains around 37 million weights that accounts for 60% weights of the model. Therefore, we compare Q-CNN^1 and Q-CNN^2 to verify the effectiveness of the error correction scheme for this layer, and the results are reported in Fig. 7.

As shown in Fig. 7, we observe that lower subspace dimensions (C_s') and more subcodewords per subspace (K) lead to low compression rates but high quantization quality, similar to the convolutional layer. The improvement brought by the error correction scheme is less significant compared with that by the convolutional layer, but is still critical to the classification accuracy, especially for higher compression rates.

In Table VIII, we compare two Q-CNN variants with the SVD-based approach [10] for compressing this layer. Both Q-CNN^1 and Q-CNN^2 offer 15 ∼ 22× compression with an ignorable degradation of performance, while SVD [10] tends to fail when the compression rate is over 10×. This is reasonable, since it is difficult to approximate the 9216×4096 weight matrix by multiplication of two rank-256 matrices, for 11.08× compression.

Then, we investigate the proposed approach for compressing the whole fully connected layers of CaffeNet in Table IX. Note that the third fully connected layer is in fact a combination of 1k linear classifiers, which is crucial to the final results. Thus, a more fine-grained quantization setting ($C_s' = 1$ and

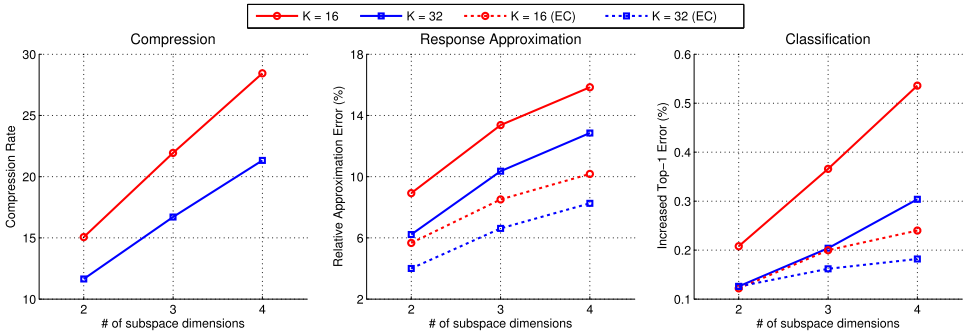


Fig. 7. Comparison on compression, response approximation error, and error of classification by quantizing the first fully connected layer in CaffeNet, under various hyperparameter settings. Since error correction does not affect the compression rates, we omit Q-CNN's error correction variant in the first figure.

TABLE VIII

COMPARISON ON COMPRESSING THE FIRST FULLY CONNECTED LAYER OF CAFFENET WITHOUT FINE-TUNING

Method	Para.	Compression	Top-1 Err. ↑	Top-5 Err. ↑
SVD [10]	-	1.38×	0.03%	-0.03%
	-	2.77×	0.07%	0.07%
	-	5.54×	0.36%	0.19%
	-	11.08×	1.23%	0.86%
Q-CNN^1	2/16	15.06×	0.19%	0.19%
	3/16	21.94×	0.35%	0.28%
	3/32	16.70×	0.18%	0.12%
	4/32	21.33×	0.28%	0.16%
Q-CNN^2	2/16	15.06×	0.10%	0.07%
	3/16	21.94×	0.18%	0.03%
	3/32	16.70×	0.14%	0.11%
	4/32	21.33×	0.16%	0.12%

TABLE IX

COMPARISON ON COMPRESSING ALL THE FULLY CONNECTED LAYERS OF CAFFENET. HERE, SVD AND DFC ARE FINE-TUNED, WHILE DPP, Q-CNN^1, AND Q-CNN^2 ARE NOT

Method	Para.	Compression	Top-1 Err. ↑	Top-5 Err. ↑
SVD [10]	-	1.26×	0.14%	-
	-	2.52×	1.22%	-
DPP [40]	-	1.54×	2.24%	-
	-	1.65×	3.92%	-
	-	1.77×	7.12%	-
	-	1.92×	13.28%	-
DFC [57]	-	1.79×	-0.66%	-
	-	3.58×	0.31%	-
Q-CNN^1	2/16	13.96×	0.28%	0.29%
	3/16	19.14×	0.70%	0.47%
	3/32	15.25×	0.44%	0.34%
	4/32	18.71×	0.75%	0.59%
Q-CNN^2	2/16	13.96×	0.31%	0.30%
	3/16	19.14×	0.59%	0.47%
	3/32	15.25×	0.31%	0.27%
	4/32	18.71×	0.57%	0.39%

$K = 16$) is adopted for the third layer. With this setting, the acceleration is not obvious, but this layer still enjoys nearly 8× compression.

From Table IX, at an expense of around 1% degradation in classification, Q-CNN can achieve pretty high compression ($13 \sim 20\times$), and significantly outperforms all the baselines. With the error correction scheme as adopted in Q-CNN^2, the classification accuracy can be slightly improved, especially under high compression rates.

3) Quantizing the Whole Network: Overall, we quantize the entire CNN model by three steps. First, the convolutional layers are quantized. Second, we fine-tune fully connected layers using the ILSVRC-12 to recover classification accuracy of the network. Final, we quantize the fine-tuned fully connected layers.

We report the results of AlexNet, CaffeNet, CNN-S, and VGG-16 in Table X. For all four networks, we quantize their convolutional layers with $C'_s = 8$ and $K = 128$ to obtain around $4 \sim 6$ times acceleration. Two hyperparameter settings are adopted for fully connected layers to achieve various compression rates.

For all models, Q-CNN^2 achieves $4 \sim 6$ times acceleration and $15 \sim 20$ times compression, at the expense of 1% loss in top-5 classification accuracy. Notably, the accuracy loss in the deepest VGG-16 network is the smallest, indicating that deeper networks may be less sensitive to parameter quantization.

Two binarization-based networks, BC [21] and BWN [24], achieve high compression rates for AlexNet, but the speed-up is relatively low and the performance loss is large. DC [27] can dramatically reduce the disk storage (35× for AlexNet and 49× for VGG-16), but the test-phase computation requires model decompression, so the reduction in run-time memory is smaller (9× and 13×). Also, the speed-up effect of DC [27] is only reported for fully connected layers, but remains unknown for the more important convolutional layers.

C. Results on Mobile Devices

To further investigate the proposed Q-CNN in resource-limited devices, experiments of image classification are conducted on a smartphone,i.e., Huawei Mate 7 equipped with a 1.8-GHz CPU. For simplicity, we only use a single CPU core for inference. The results are depicted in Table XI.

For mobile devices, the proposed Q-CNN approach gets 3× acceleration for AlexNet, and 4× for CNN-S. Besides, the storage consumption is reduced by 20×, and the memory consumption is less than one-third of their original networks. The experimental results demonstrate that our approach can significantly improve the inference efficiency, making it possible to deploy CNN networks on resource-limited devices.

D. Realistic Versus Theoretical Acceleration

In Table XII, we show the realistic and theoretical acceleration for AlexNet. Experiments are implemented on desk-

TABLE X

COMPARISON OF ACCELERATION AND COMPRESSION ON Q-CNN NETWORKS. HERE, SET $C'_s = 1$ AND $K = 16$ FOR THE QUANTIZATION OF THE THIRD FULLY CONNECTED LAYER

Model	Method	Para.		Speed-up	Compression	Top-1 Err. ↑	Top-5 Err. ↑
		Conv.	FCnt.				
AlexNet	BC [21]	-	-	2.00×	32.00×	21.20%	19.20%
	BWN [24]	-	-	2.00×	32.00×	2.80%	3.20%
	DC [27]	-	-	-	9.00 / 35.00×	0.00%	-0.03%
	Q-CNN^2	**8/128**	**3/32**	**4.05×**	**15.10×**	**1.38%**	**0.84%**
		8/128	**4/32**	**4.15×**	**18.32×**	**1.46%**	**0.97%**
CaffeNet	Q-CNN^2	**8/128**	**3/32**	**4.04×**	**15.10×**	**1.43%**	**0.99%**
		8/128	**4/32**	**4.16×**	**18.32×**	**1.54%**	**1.12%**
CNN-S	Q-CNN^2	**8/128**	**3/32**	**5.69×**	**15.93×**	**1.48%**	**0.81%**
		8/128	**4/32**	**5.78×**	**19.57×**	**1.64%**	**0.85%**
VGG-16	DC [27]	-	-	-	13.00 / 49.00×	-0.33%	-0.41%
	Q-CNN^2	**8/128**	**3/32**	**4.92×**	**16.06×**	**1.02%**	**0.38%**
		8/128	**4/32**	**4.94×**	**19.60×**	**1.13%**	**0.45%**

TABLE XI

RESOURCE CONSUMPTION AND PERFORMANCE COMPARISON FOR ALEXNET AND CNN-S ON MOBILE DEVICE

Model	Method	Time	Storage	Memory	Top-5 Err.
AlexNet	CNN	2.93s	232.56MB	264.74MB	19.74%
	Q-CNN^2	0.95s	12.60MB	74.65MB	20.70%
CNN-S	CNN	10.58s	392.57MB	468.90MB	15.82%
	Q-CNN^2	2.61s	20.13MB	129.49MB	16.68%

TABLE XII

THEORETICAL AND REALISTIC ACCELERATION ON ALEXNET WITH SINGLE-THREADED CPU CORE. THE DEFAULT BLAS CHOICE, ATLAS LIBRARY, IS USED IN CAFFE LIKE [34]

BLAS	FLOPs		Time (ms)		Speed-up	
	CNN	Q-CNN	CNN	Q-CNN	Theo.	Real.
Off	7.29e+8	1.75e+8	321.10	75.62	4.15×	4.25×
On			167.79	55.35		3.03×

top PC, equipped with an Intel Core i7-4790K CPU and 32-GB RAM. All the results are measured with a single-thread mode on the CPU core.

Caffe [34] uses BLAS [58] to facilitate matrix multiplication, but it is unclear that BLAS is still effective for ARM CPU. Thus, we compare the two settings with or without BLAS optimization, respectively. The final acceleration rate, however, is mildly lower when BLAS is used, indicating that BLAS is not friendly to our Q-CNN. There are some other libraries, e.g., AVX [59], SIMD, and SSE, maybe more friendly to Q-CNN, which will be investigated in future work.

VI. CONCLUSION

A quantization approach, i.e., Q-CNN, is proposed to simultaneously accelerate and compress the convolutional networks. Our approach enables efficient inference based on the highly compressed network parameters. Extensive experiments demonstrate that the proposed Q-CNN can achieve impressive performance on compression and acceleration for CNN models.

REFERENCES

[1] Y. LeCun *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2012, pp. 1106–1114.

[3] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015.

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2014, pp. 580–587.

[5] R. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, Sep. 2015.

[6] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3431–3440.

[7] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder-decoder architecture for image segmentation," *CoRR*, vol. abs/1511.00561, Nov. 2015.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, Dec. 2015.

[9] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *Proc. Brit. Mach. Vis. Conf. (BMVC)*, 2014.

[10] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2014, pp. 1269–1277.

[11] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned CP-decomposition," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–18.

[12] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun, "Efficient and accurate approximations of nonlinear convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 1984–1992.

[13] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *CoRR*, vol. abs/1505.06798, Nov. 2015.

[14] P. Wang and J. Cheng, "Accelerating convolutional neural networks for mobile applications," in *Proc. ACM Multimedia Conf.*, 2016, pp. 541–545.

[15] V. Lebedev and V. S. Lempitsky, "Fast ConvNets using group-wise brain damage," *CoRR*, vol. abs/1506.02515, Dec. 2015.

[16] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *CoRR*, vol. abs/1512.08571, Dec. 2015.

[17] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2015, pp. 442–450.

[18] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2015, pp. 1737–1746.

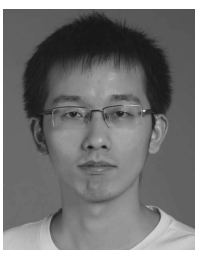
[19] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015.

[20] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," *CoRR*, vol. abs/1603.01025, Mar. 2016.

[21] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2015, pp. 3123–3131.
[22] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural networks with few multiplications," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016.
[23] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to $+1$ or -1," *CoRR*, vol. abs/1602.02830, Feb. 2016.
[24] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," *CoRR*, vol. abs/1603.05279, Aug. 2016.
[25] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," *CoRR*, vol. abs/1412.6115, Dec. 2014.
[26] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2015, pp. 2285–2294.
[27] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016.
[28] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2014, pp. 2654–2662.
[29] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *Proc. Deep Learn. Workshop Adv. Neural Inf. Process. Syst. (NIPSW)*, 2014, pp. 1–9.
[30] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "FitNets: Hints for thin deep nets," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015.
[31] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size," *CoRR*, vol. abs/1602.07360, Nov. 2016.
[32] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[33] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.
[34] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," *CoRR*, vol. abs/1408.5093, Jun. 2014.
[35] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," in *Proc. Brit. Mach. Vis. Conf. (BMVC)*, 2014.
[36] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 4820–4828.
[37] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2007, pp. 153–160.
[38] H. Lee, C. Ekanadham, and A. Y. Ng, "Sparse deep belief net model for visual area V2," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2008, pp. 873–880.
[39] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "High-performance neural networks for visual object classification," *CoRR*, vol. abs/1102.0183, Feb. 2011.
[40] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," in *Proc. Brit. Mach. Vis. Conf. (BMVC)*, 2015, pp. 31.1–31.12.
[41] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Penksy, "Sparse convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 806–814.
[42] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2015, pp. 1135–1143.
[43] Y. Sun, X. Wang, and X. Tang, "Sparsifying neural network connections for face recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 4856–4864.
[44] A. W. Savich, M. Moussa, and S. Areibi, "The impact of arithmetic representation on implementing MLP-BP on FPGAs: A study," *IEEE Trans. Neural Netw.*, vol. 18, no. 1, pp. 240–252, Jan. 2007.
[45] D. L. Ly and P. Chow, "High-performance reconfigurable hardware architecture for restricted Boltzmann machines," *IEEE Trans. Neural Netw.*, vol. 21, no. 11, pp. 1780–1792, Nov. 2010.
[46] F. Ortega-Zamorano, J. M. Jerez, D. U. Muñoz, R. M. Luque-Baena, and L. Franco, "Efficient implementation of the backpropagation algorithm in FPGAs and microcontrollers," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 9, pp. 1840–1850, Sep. 2016.
[47] C. Leng, J. Wu, J. Cheng, X. Bai, and H. Lu, "Online sketching hashing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 2503–2511.
[48] C. Leng, J. Wu, J. Cheng, X. Zhang, and H. Lu, "Hashing for distributed data," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2015, pp. 1642–1650.
[49] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen, "Compressing convolutional neural networks," *CoRR*, vol. abs/1506.04449, Jun. 2015.
[50] Y. Gong and S. Lazebnik, "Iterative quantization: A procrustean approach to learning binary codes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2011, pp. 817–824.
[51] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proc. Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2006, pp. 535–541.
[52] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–128, Jan. 2011.
[53] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. Annu. Symp. Comput. Geometry (SCG)*, 2004, pp. 253–262.
[54] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 12, pp. 2916–2929, Dec. 2013.
[55] C. Tai, T. Xiao, Y. Zhang, X. Wang, and W. E, "Convolutional neural networks with low-rank regularization," *CoRR*, vol. abs/1511.06067, Nov. 2015.
[56] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2013, pp. 2148–2156.
[57] Z. Yang *et al.*, "Deep fried convnets," *CoRR*, vol. abs/1412.7149, Dec. 2014.
[58] R. C. Whaley and A. Petitet, "Minimizing development and maintenance costs in supporting persistently optimized BLAS," *Softw., Pract. Exper.*, vol. 35, no. 2, pp. 101–121, Feb. 2005.
[59] Intel Cooperation. (Feb. 2016). "Intel architecture instruction set extensions programming reference." Intel Corp., Mountain View, CA, USA, Tech. Rep. 319433-030. [Online]. Available: https://software.intel.com/en-us/isa-extensions

Jian Cheng received the B.S. and M.S. degrees in mathematics from Wuhan University, Wuhan, China, in 1998 and in 2001, respectively, and the Ph.D. degree from the Institute of Automation, Chinese Academy of Sciences, Beijing, China.

He is currently a Professor with the Institute of Automation, Chinese Academy of Sciences. His current research interests include image and video indexing and search, deep learning, and recommender systems.



Jiaxiang Wu received the B.E. degree in automation from the Beijing Institute of Technology, Beijing, China, in 2012, and the Ph.D. degree in computer science from the Institute of Automation, Chinese Academy of Sciences, Beijing, in 2017.

His current research interests include deep learning, image retrieval, and recommender systems.

Cong Leng received the B.E. degree in automation from Central South University, Changsha, China, in 2011, and the Ph.D. degree in pattern recognition and intelligent systems from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2016.

His current research interests include machine learning, deep learning, and their applications in computer vision and data mining.

Yuhang Wang received the B.E. degree from Zhejiang University, Hangzhou, China, in 2012, and the Ph.D. degree from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2017.

His current research interests include deep learning, image or video segmentation, and object detection.

Qinghao Hu received the B.E. degree in computer science from Northwestern Polytechnical University, Xi'an, China, in 2014. He is currently pursuing the Ph.D. degree with the National Laboratory of Pattern Recognition, Image and Video Analysis Group, Institute of Automation, Chinese Academy of Sciences, Beijing, China.

His current research interests include computer vision and multimedia, especially deep learning and hashing.