ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/287853408

# Quantized Convolutional Neural Networks for Mobile Devices

**Article** · December 2015

Source: arXiv

CITATIONS
186

READS
1,135

**5 authors**, including:

Jiaxiang Wu
Chinese Academy of Sciences
**14** PUBLICATIONS **821** CITATIONS

SEE PROFILE

Cong Leng
Chinese Academy of Sciences
**17** PUBLICATIONS **795** CITATIONS

SEE PROFILE

**Some of the authors of this publication are also working on these related projects:**

Project Multi-modal Gesture Recognition View project

All content following this page was uploaded by Jiaxiang Wu on 18 March 2016.

The user has requested enhancement of the downloaded file.

arXiv:1512.06473v1 [cs.CV] 21 Dec 2015

# Quantized Convolutional Neural Networks for Mobile Devices

Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, Jian Cheng
National Laboratory of Patter Recognition
Institute of Automation, Chinese Academy of Sciences
{jiaxiang.wu, cong.leng, yuhang.wang, qinghao.hu, jcheng}@nlpr.ia.ac.cn

## Abstract

*Recently, convolutional neural networks (CNN) have demonstrated impressive performance in various computer vision tasks. However, high performance hardware is typically indispensable for the application of CNN models due to the high computation complexity, which prohibits their further extensions. In this paper, we propose an efficient framework, namely Quantized CNN, to simultaneously speed-up the computation and reduce the storage and memory overhead of CNN models. Both filter kernels in convolutional layers and weighting matrices in fully-connected layers are quantized, aiming at minimizing the estimation error of each layer's response. Extensive experiments on the ILSVRC-12 benchmark demonstrate* $4 \sim 6\times$ *speed-up and* $15 \sim 20\times$ *compression with merely one percentage loss of classification accuracy. With our quantized CNN model, even mobile devices can accurately classify images within one second.*

## 1. Introduction

In recent years, we have witnessed the great success of convolutional neural networks (CNN) [18] in a wide range of visual applications, including image classification [15, 26, 24, 22], object detection [9, 8, 20], etc. This success mainly comes from deeper network architectures as well as the tremendous training data. However, as the network grows deeper, the model complexity is also increasing exponentially in both the training and testing stages, which leads to the very high demand in the computation ability. For instance, the 8-layer AlexNet [15] involves 60M parameters and requires over 729M FLOPs[1] to classify a single image. Although the training stage can be offline carried out on high performance clusters with GPU acceleration, the testing computation cost may be unaffordable for common personal computers and mobile devices. Due to the limited computation ability and memory space, mobile devices are almost intractable to run deep convolutional networks. Therefore, it is crucial to accelerate the computation and compress the memory consumption for CNN models.

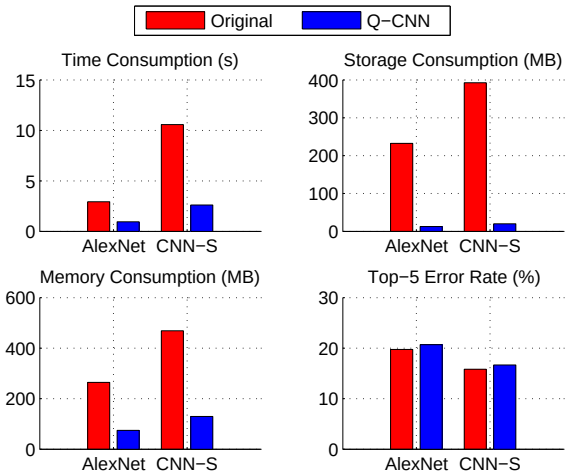


Figure 1: Comparison on the efficiency and classification accuracy between the original and quantized AlexNet [15] and CNN-S [1] on a Huawei® Mate 7 smartphone.

For most CNNs, convolutional layers are the most time-consuming part, while fully-connected layers involve massive network parameters. Due to the intrinsical difference between them, existing works usually focus on improving the efficiency for either convolutional layers or fully-connected layers. In [6, 12, 28, 27, 17, 16], low-rank approximation or tensor decomposition is adopted to speed-up convolutional layers. On the other hand, parameter compression in fully-connected layers is explored in [10, 25, 3, 2, 29, 23]. Overall, the above-mentioned algorithms are able to achieve faster speed or less storage. However, few of them can achieve significant acceleration and compression simultaneously for the whole network.

In this paper, we propose a unified framework for convolutional networks, namely Quantized CNN (Q-CNN), to simultaneously accelerate and compress CNN models with only minor performance degradation. With network pa-

[1] FLOPs: number of FLoating-point OPerations required to classify one image with the convolutional network.

rameters quantized, the response of both convolutional and fully-connected layers can be efficiently estimated via the approximate inner product computation. We minimize the estimation error of each layer's response during parameter quantization, which can better preserve the model performance. In order to suppress the accumulative error while quantizing multiple layers, an effective training scheme is introduced to take previous estimation error into consideration. Our Q-CNN model enables fast test-phase computation, and the storage and memory consumption are also significantly reduced.

We evaluate our Q-CNN framework for image classification on two benchmarks, MNIST [19] and ILSVRC-12 [21]. For MNIST, our Q-CNN approach achieves over $12\times$ compression for two neural networks (no convolution), with lower accuracy loss than several baseline methods. For ILSVRC-12, we attempt to improve the test-phase efficiency of four convolutional networks: AlexNet [15], CaffeNet [14], CNN-S [1], and VGG-16 [22]. Generally, Q-CNN achieves $4\times$ acceleration and $15\times$ compression (sometimes higher) for each network, with less than 1% drop in the top-5 classification accuracy. Moreover, we implement the quantized CNN model on mobile devices, and dramatically improve the test-phase efficiency, as depicted in Figure 1. The main contributions of this paper can be summarized as follows:

- We propose a unified Q-CNN framework to accelerate and compress convolutional networks. We demonstrate that better quantization can be learned by minimizing the estimation error of each layer's response.
- We propose an effective training scheme to suppress the accumulative error while quantizing the whole convolutional network.
- Our Q-CNN framework achieves $4 \sim 6\times$ speed-up and $15 \sim 20\times$ compression, while the classification accuracy loss is within one percentage. Moreover, the quantized CNN model can be implemented on mobile devices and classify an image within one second.

## 2. Preliminary

During the test phase of convolutional networks, the computation overhead is dominated by convolutional layers; meanwhile, the majority of network parameters are stored in fully-connected layers. Therefore, for better test-phase efficiency, it is critical to speed-up the convolution computation and compress parameters in fully-connected layers.

Our observation is that the forward-passing process of both convolutional and fully-connected layers is dominated by the computation of inner products. More formally, we consider a convolutional layer with input feature maps $S \in \mathbb{R}^{d_s \times d_s \times C_s}$ and response feature maps $T \in \mathbb{R}^{d_t \times d_t \times C_t}$, where $d_s, d_t$ are the spatial sizes and $C_s, C_t$ are the number of feature map channels. The response at the 2-D spatial position $p_t$ in the $c_t$-th response feature map is computed as:

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

where $W_{c_t} \in \mathbb{R}^{d_k \times d_k \times C_s}$ is the $c_t$-th convolutional kernel and $d_k$ is the kernel size. We use $p_s$ and $p_k$ to denote the 2-D spatial positions in the input feature maps and convolutional kernels, and both $W_{c_t, p_k}$ and $S_{p_s}$ are $C_s$-dimensional vectors. The layer response is the sum of inner products at all positions within the $d_k \times d_k$ receptive field in the input feature maps.

Similarly, for a fully-connected layer, we have:

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

where $S \in \mathbb{R}^{C_s}$ and $T \in \mathbb{R}^{C_t}$ are the layer input and layer response, respectively, and $W_{c_t} \in \mathbb{R}^{C_s}$ is the weighting vector for the $c_t$-th neuron of this layer.

Product quantization [13] is widely used for the approximate nearest neighbor search. The idea is to decompose the feature space as the Cartesian product of multiple subspaces, and then learn sub-codebooks for each subspace. A vector is represented by the concatenation of sub-codewords for efficient distance computation and storage.

In this paper, we leverage product quantization to implement the efficient inner product computation. Let us consider the inner product computation between $x, y \in \mathbb{R}^D$. At first, both $x$ and $y$ are split into $M$ sub-vectors, denoted as $x^{(m)}$ and $y^{(m)}$. Afterwards, each $x^{(m)}$ is quantized with a sub-codeword from the $m$-th sub-codebook, then we have

$$\langle y, x \rangle = \sum\nolimits_m \langle y^{(m)}, x^{(m)} \rangle \approx \sum\nolimits_m \langle y^{(m)}, c^{(m)}_{k_m} \rangle \tag{3}$$

which transforms the $\mathcal{O}(D)$ inner product computation to $M$ addition operations ($M \leq D$), if the inner products between each sub-vector $y^{(m)}$ and all the sub-codewords in the $m$-th sub-codebook have been computed in advance.

Quantization-based approaches have been explored in several works [10, 3, 2, 11]. These approaches mostly focus on compressing parameters in fully-connected layers [10, 3], and none of them can provide acceleration for the test-phase computation. Furthermore, [10, 11] require the network parameters to be re-constructed during the test-phase, which limit the compression to disk storage instead of memory consumption. On the contrary, our approach offers simultaneous acceleration and compression for both convolutional and fully-connected layers, and can reduce the run-time memory consumption dramatically.

## 3. Quantized CNN

In this section, we present our approach for accelerating and compressing convolutional networks. Firstly, we introduce an efficient test-phase computation process with the

network parameters quantized. Secondly, we demonstrate that better quantization can be learned by directly minimizing the estimation error of each layer's response. Finally, we analyze the computation complexity of our quantized CNN model.

### 3.1. Quantizing the Fully-connected Layer

For a fully-connected layer, we denote its weighting matrix as $W \in \mathbb{R}^{C_s \times C_t}$, where $C_s$ and $C_t$ are the dimensions of the layer input and response, respectively. The weighting vector $W_{c_t}$ is the $c_t$-th column vector in $W$.

We evenly split the $C_s$-dimensional space (where $W_{c_t}$ lies in) into $M$ subspaces, each of $C'_s = C_s/M$ dimensions. Each $W_{c_t}$ is then decomposed into $M$ sub-vectors, denoted as $W_{c_t}^{(m)}$. A sub-codebook can be learned for each subspace after gathering all the sub-vectors within this subspace. Formally, for the $m$-th subspace, we optimize:

$$\begin{aligned} \min_{D^{(m)}, B^{(m)}} & \left\| D^{(m)} B^{(m)} - W^{(m)} \right\|_F^2 \\ s.t.\ & D^{(m)} \in \mathbb{R}^{C'_s \times K}, B^{(m)} \in \{0,1\}^{K \times C_t} \end{aligned} \tag{4}$$

where $W^{(m)} \in \mathbb{R}^{C'_s \times C_t}$ consists of the $m$-th sub-vectors of all weighting vectors. The sub-codebook $D^{(m)}$ contains $K$ sub-codewords, and each column in $B^{(m)}$ is an indicator vector (only one non-zero entry), specifying which sub-codeword is used to quantize the corresponding sub-vector. The optimization can be solved via k-means clustering.

The layer response is approximately computed as:

$$\begin{aligned} T(c_t) = \sum\nolimits_m \langle W_{c_t}^{(m)}, S^{(m)} \rangle &\approx \sum\nolimits_m \langle D^{(m)} B_{c_t}^{(m)}, S^{(m)} \rangle \\ &= \sum\nolimits_m \langle D_{k_m(c_t)}^{(m)}, S^{(m)} \rangle \end{aligned} \tag{5}$$

where $B_{c_t}^{(m)}$ is the $c_t$-th column vector in $B^{(m)}$, and $S^{(m)}$ is the $m$-th sub-vector of the layer input. $k_m(c_t)$ is the index of the sub-codeword used to quantize the sub-vector $W_{c_t}^{(m)}$.

In Figure 2, we depict the parameter quantization and test-phase computation process of the fully-connected layer. By decomposing the weighting matrix into $M$ sub-matrices, $M$ sub-codebooks can be learned, one per subspace. During the test-phase, the layer input is split into $M$ sub-vectors, denoted as $S^{(m)}$. For each subspace, we compute the inner products between $S^{(m)}$ and every sub-codeword in $D^{(m)}$, and store the results in a look-up table. Afterwards, only $M$ addition operations are required to compute each response. As a result, the overall time complexity can be reduced from $\mathcal{O}(C_s C_t)$ to $\mathcal{O}(C_s K + C_t M)$. On the other hand, only sub-codebooks and quantization indices need to be stored, which can dramatically reduce the storage consumption.

### 3.2. Quantizing the Convolutional Layer

Unlike the 1-D weighting vector in the fully-connected layer, each convolutional kernel is a 3-dimensional tensor:

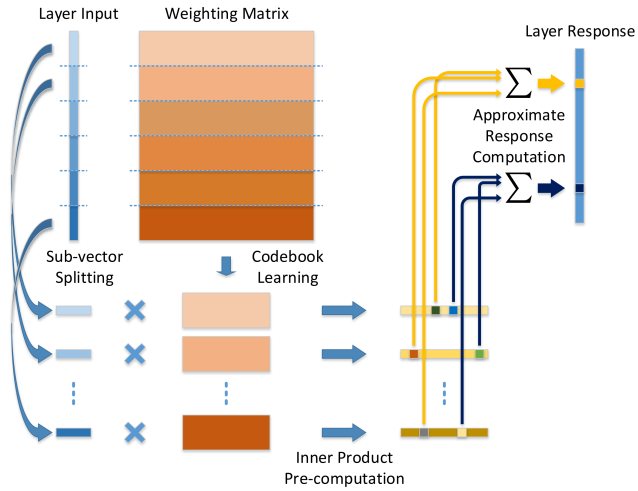


Figure 2: The parameter quantization and test-phase computation process of the fully-connected layer.

$W_{c_t} \in \mathbb{R}^{d_k \times d_k \times C_s}$. Before quantization, we need to determine how to split it into sub-vectors, i.e. apply subspace splitting to which dimension. During the test phase, the input feature maps are traversed by each convolutional kernel with a sliding window in the spatial domain. Since these sliding windows are partially overlapped, we split each convolutional kernel along the dimension of feature map channels, so that the pre-computed inner products can be reused at multiple spatial locations. Specifically, we learn the quantization in each subspace by:

$$\begin{aligned} \min_{D^{(m)}, \{B_{p_k}^{(m)}\}} & \sum\nolimits_{p_k} \left\| D^{(m)} B_{p_k}^{(m)} - W_{p_k}^{(m)} \right\|_F^2 \\ s.t.\ & D^{(m)} \in \mathbb{R}^{C'_s \times K}, B_{p_k}^{(m)} \in \{0,1\}^{K \times C_t} \end{aligned} \tag{6}$$

where $W_{p_k}^{(m)} \in \mathbb{R}^{C'_s \times C_t}$ contains the $m$-th sub-vectors of all convolutional kernels at position $p_k$. The optimization can also be solved by k-means clustering in each subspace.

With the convolutional kernels quantized, we approximately compute the response feature maps by:

$$\begin{aligned} T_{p_t}(c_t) &= \sum\nolimits_{(p_k, p_s)} \sum\nolimits_m \langle W_{c_t, p_k}^{(m)}, S_{p_s}^{(m)} \rangle \\ &\approx \sum\nolimits_{(p_k, p_s)} \sum\nolimits_m \langle D^{(m)} B_{c_t, p_k}^{(m)}, S_{p_s}^{(m)} \rangle \\ &= \sum\nolimits_{(p_k, p_s)} \sum\nolimits_m \langle D_{k_m(c_t, p_k)}^{(m)}, S_{p_s}^{(m)} \rangle \end{aligned} \tag{7}$$

where $S_{p_s}^{(m)}$ is the $m$-th sub-vector at position $p_s$ in the input feature maps, and $k_m(c_t, p_k)$ is the index of the sub-codeword to quantize the $m$-th sub-vector at position $p_k$ in the $c_t$-th convolutional kernel.

Similar to the fully-connected layer, we pre-compute the look-up tables of inner products with the input feature maps.

Then, the response feature maps are approximately computed with (7), and both the time and storage complexity can be greatly reduced.

### 3.3. Quantization with Error Correction

So far, we have presented an intuitive approach to quantize parameters and improve the test-phase efficiency of convolutional networks. However, there are still two critical drawbacks. First, minimizing the quantization error of model parameters does not necessarily give the optimal quantized network for the classification accuracy. In contrast, minimizing the estimation error of each layer's response is more closely related to the network's classification performance. Second, the quantization of one layer is independent of others, which may lead to the accumulation of error when quantizing multiple layers. The estimation error of the network's final response is very likely to be quickly accumulated, since the error introduced by the previous quantized layers will also affect the following layers.

To overcome these two limitations, we introduce the idea of error correction into the quantization of network parameters. This improved quantization approach directly minimizes the estimation error of the response at each layer, and can compensate the error introduced by previous layers. With the error correction scheme, we can quantize the network with much less performance degradation than the original quantization method.

#### 3.3.1 Error Correction for the Fully-connected Layer

Suppose we have $N$ images to learn the quantization of a fully-connected layer, and the layer input and response of image $I_n$ are denoted as $S_n$ and $T_n$. In order to minimize the estimation error of the layer response, we optimize:

$$\min_{\{D^{(m)}\},\{B^{(m)}\}} \sum\nolimits_n \left\| T_n - \sum\nolimits_m (D^{(m)} B^{(m)})^T S_n^{(m)} \right\|_F^2 \tag{8}$$

where the first term in the Frobenius norm is the desired layer response, and the second term is the approximated layer response computed via the quantized parameters.

A block coordinate descent approach can be applied to minimize this objective function. For the $m$-th subspace, its residual error is defined as:

$$R_n^{(m)} = T_n - \sum\nolimits_{m' \neq m} (D^{(m')} B^{(m')})^T S_n^{(m')} \tag{9}$$

and then we attempt to minimize the residual error of this subspace, which is:

$$\min_{D^{(m)},B^{(m)}} \sum\nolimits_n \left\| R_n^{(m)} - (D^{(m)} B^{(m)})^T S_n^{(m)} \right\|_F^2 \tag{10}$$

and the above optimization can be solved by alternatively updating the sub-codebook and sub-codeword assignment.

**Update $D^{(m)}$.** We fix the sub-codeword assignment $B^{(m)}$, and define $L_k = \{c_t | B^{(m)}(k, c_t) = 1\}$. The optimization in (10) can be re-formulated as:

$$\min_{\{D_k^{(m)}\}} \sum\nolimits_{n,k} \sum\nolimits_{c_t \in L_k} [R_n^{(m)}(c_t) - D_k^{(m)^T} S_n^{(m)}]^2 \tag{11}$$

which implies that the optimization over one sub-codeword does not affect other sub-codewords. Hence, for each sub-codeword, we construct a least square problem from (11) to update it.

**Update $B^{(m)}$.** With the sub-codebook $D^{(m)}$ fixed, it is easy to discover that the optimization of each column in $B^{(m)}$ is mutually independent. For the $c_t$-th column, its optimal sub-codeword assignment is given by:

$$k_m^*(c_t) = \arg\min_k \sum\nolimits_n [R_n^{(m)}(c_t) - D_k^{(m)^T} S_n^{(m)}]^2 \tag{12}$$

#### 3.3.2 Error Correction for the Convolutional Layer

We adopt the similar idea to minimize the estimation error of the convolutional layer's response feature maps, that is:

$$\min_{\{D^{(m)}\},\{B_{p_k}^{(m)}\}} \sum_{n,p_t} \left\| T_{n,p_t} - \sum_{(p_k,p_s)} \sum_m (D^{(m)} B_{p_k}^{(m)})^T S_{n,p_s}^{(m)} \right\|_F^2 \tag{13}$$

The optimization also can be solved by block coordinate descent. More details on solving this optimization can be found in the appendix.

#### 3.3.3 Error Correction for Multiple Layers

The above quantization method can be sequentially applied to each layer in the CNN model. One concern is that the estimation error of layer response caused by the previous layers will be accumulated and affect the quantization of the following layers. Here, we propose an effective training scheme to address this issue.

We consider the quantization of a specific layer, assuming its previous layers have already been quantized. The optimization of parameter quantization is based on the layer input and response of a group of training images. To quantize this layer, we take the layer input in the quantized network as $\{S_n\}$, and the layer response in the original network (not quantized) as $\{T_n\}$ in Eq. (8) and (13). In this way, the optimization is guided by the actual input in the quantized network and the desired response in the original network. The accumulative error introduced by the previous layers is explicitly taken into consideration during optimization. In consequence, this training scheme can effectively suppress the accumulative error for the quantization of multiple layers.

Another possible solution is to adopt back-propagation to jointly update the sub-codebooks and sub-codeword assignments in all quantized layers. However, since the sub-codeword assignments are discrete, the gradient-based optimization can be quite difficult, if not entirely impossible. Therefore, back-propagation is not adopted here, but could be a promising extension for future work.

### 3.4. Computation Complexity

Now we analyze the test-phase computation complexity of convolutional and fully-connected layers, with or without parameter quantization. For our proposed Q-CNN model, the forward-passing through each layer mainly consists of two procedures: pre-computation of inner products, and approximate computation of layer response. Both sub-codebooks and sub-codeword assignments are stored for the test-phase computation. We report the detailed comparison on the computation and storage overhead in Table 1 and 2.

Table 1: Comparison on the computation overhead of convolutional and fully-connected layers.

| Layer | Model | FLOPs |
|---|---|---|
| Conv. | CNN | $d_t^2 C_t d_k^2 C_s$ |
| | Q-CNN | $d_s^2 C_s K + d_t^2 C_t d_k^2 M$ |
| FCnt. | CNN | $C_s C_t$ |
| | Q-CNN | $C_s K + C_t M$ |

Table 2: Comparison on the storage overhead of convolutional and fully-connected layers.

| Layer | Model | Bytes |
|---|---|---|
| Conv. | CNN | $4 d_k^2 C_s C_t$ |
| | Q-CNN | $4 C_s K + \frac{1}{8} d_k^2 M C_t \log_2 K$ |
| FCnt. | CNN | $4 C_s C_t$ |
| | Q-CNN | $4 C_s K + \frac{1}{8} M C_t \log_2 K$ |

As we can see from Table 1 and 2, the reduction in the computation and storage overhead largely depends on two hyper-parameters, $M$ (number of subspaces) and $K$ (number of sub-codewords in each subspace). Large values of $M$ and $K$ lead to more fine-grained quantization, but is less efficient in the computation and storage consumption. In practice, we can vary these two parameters to balance the trade-off between the test-phase efficiency and accuracy loss of the quantized CNN model.

## 4. Related Work

There have been a few attempts in accelerating the test-phase computation of convolutional networks, and most of them are inspired from the low-rank decomposition. Denton et al. [6] presented a series of low-rank decomposition designs for convolutional kernels. Similarly, a low-rank CP-decomposition was adopted in [16] to transform a convolutional layer into multiple layers of lower complexity. Zhang et al. [28, 27] considered the subsequent nonlinear units while learning the low-rank decomposition. [17] applied group-wise pruning to the convolutional tensor to decompose it into the multiplications of thinned dense matrices.

Another parallel research trend is to compress parameters in fully-connected layers. Ciresan et al. [4] reduced network parameters by randomly removing connections before training. [5] decomposed the weighting matrix into two low-rank matrices, and singular value decomposition was adopted in [6] for the matrix factorization. [7] trained a distilled model to mimic the response of a larger and well-trained network. Srinivas et al. [23] proposed to remove redundant neurons instead of network connections. In [25], fully-connected layers were replaced by a single "Fastfood" layer for end-to-end training with convolutional layers. Chen et al. [3] randomly grouped connection weights into hash buckets, and then fine-tuned network parameters with back-propagation. Gong et al. [10] showed that vector quantization had a clear advantage over matrix factorization methods in compressing fully-connected layers.

## 5. Experiments

In this section, we evaluate our quantized CNN framework on two image classification benchmarks, MNIST [19] and ILSVRC-12 [21]. For the acceleration of convolutional layers, we compare with:

- **CPD** [16]: CP-Decomposition;
- **GBD** [17]: Group-wise Brain Damage;
- **LANR** [28, 27]: Low-rank Approximation of Nonlinear Responses.

and for the compression of fully-connected layers, we compare with the following approaches:

- **RER** [4]: Random Edge Removal;
- **LRD** [5]: Low-Rank Decomposition;
- **DK** [7]: Dark Knowledge;
- **HashNet** [3]: Hashed Neural Nets;
- **DPP** [23]: Data-free Parameter Pruning;
- **SVD** [6]: Singular Value Decomposition;
- **DFC** [25]: Deep Fried Convnets.

For all above baseline methods, we use their reported results under the same experiment setting for fair comparison.

Our approaches are denoted as "Q-CNN" and "Q-CNN (EC)", where the former one adopts the naive product quantization, and the latter one learns product quantization with error correction. We implement the optimization process of parameter quantization in MATLAB, and the fine-tuning of CNN models is based on the Caffe [14] framework. Additional results of our approach can be found in the appendix.

### 5.1. Results on MNIST

The MNIST dataset contains 70k images of hand-written digits, 60k used for training and 10k for testing. To evaluate the compression performance, we pre-train two neural networks, one is 3-layer and another one is 5-layer, where each hidden layer contains 1000 units. Different compression techniques are then adopted to compress these two network, and the results are as depicted in Table 3.

Table 3: Comparison on the compression rates and classification error on MNIST, based on a 3-layer network (784-1000-10) and a 5-layer network (784-1000-1000-1000-10).

| Method | 3-layer | | 5-layer | |
|---|---|---|---|---|
| | Compr. | Error | Compr. | Error |
| Original | - | 1.35% | - | 1.12% |
| RER [4] | 8× | 2.19% | 8× | 1.24% |
| LRD [5] | 8× | 1.89% | 8× | 1.77% |
| DK [7] | 8× | 1.71% | 8× | 1.26% |
| HashNets [3] | 8× | 1.43% | 8× | 1.22% |
| Q-CNN | 12.1× | 1.42% | 13.4× | 1.34% |
| Q-CNN (EC) | 12.1× | **1.39%** | 13.4× | **1.19%** |

In our Q-CNN framework, the trade-off between accuracy and efficiency is controlled by $M$ (number of subspaces) and $K$ (number of sub-codewrods in each subspace). Since $M = C_s/C'_s$ is determined once $C'_s$ is given, we tune $(C'_s, K)$ to adjust the quantization precision. In Table 3, we set the hyper-parameters as $C'_s = 4$ and $K = 32$.

From Table 3, we observe that our Q-CNN (EC) approach offers higher compression rates with less performance degradation than all baselines for both networks. The error correction scheme is effective in reducing the accuracy loss, especially for deeper networks (5-layer). Also, we find the performance of both Q-CNN and Q-CNN (EC) quite stable, as the standard deviation of five random runs is merely 0.05%. Therefore, we report the single-run performance in the remaining experiments.

### 5.2. Results on ILSVRC-12

The ILSVRC-12 benchmark consists of over one million training images drawn from 1000 categories, and a disjoint validation set of 50k images. We report both the top-1 and top-5 classification error rates on the validation set, using single-view testing (central patch only).

We demonstrate our approach on four convolutional networks: AlexNet [15], CaffeNet [14], CNN-S [1], and VGG-16 [22]. The first two models have been adopted in several related works, and therefore are included for comparison. CNN-S and VGG-16 use a either wider or deeper structure for better classification accuracy, and are included here to prove the scalability of our approach. We compare all these networks' computation and storage overhead in Table 4, together with their classification error rates on ILSVRC-12.

Table 4: Comparison on the test-phase computation overhead (FLOPs), storage consumption (Bytes), and classification error rates (Top-1/5 Err.) of AlexNet, CaffeNet, CNN-S, and VGG-16.

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

#### 5.2.1 Quantizing the Convolutional Layer

To begin with, we quantize the second convolutional layer of AlexNet, which is the most time-consuming layer during the test-phase. In Table 5, we report the performance under several $(C'_s, K)$ settings, comparing with two baseline methods, CPD [16] and GBD [17].

Table 5: Comparison on the speed-up rates and the increase of top-1/5 error rates for accelerating the second convolutional layer in AlexNet, with or without fine-tuning (FT). The hyper-parameters of Q-CNN, $C'_s$ and $K$, are as specified in the "Para." column.

| Method | Para. | Speed-up | Top-1 Err. ↑ | | Top-5 Err. ↑ | |
|---|---|---|---|---|---|---|
| | | | No FT | FT | No FT | FT |
| CPD | - | 3.19× | - | - | 0.94% | 0.44% |
| | - | 4.52× | - | - | 3.20% | 1.22% |
| | - | 6.51× | - | - | 69.06% | 18.63% |
| GBD | - | 3.33× | 12.43% | 0.11% | - | - |
| | - | 5.00× | 21.93% | 0.43% | - | - |
| | - | 10.00× | 48.33% | 1.13% | - | - |
| Q-CNN | 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 (EC) | 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% |

From Table 5, we discover that with a large speed-up rate (over 4×), the performance loss of both CPD and GBD become severe, especially before fine-tuning. The naive parameter quantization method also suffers from the similar problem. By incorporating the idea of error correction, our Q-CNN model achieves up to 6× speed-up with merely 0.6% drop in accuracy, even without fine-tuning. The accuracy loss can be further reduced after fine-tuning the subsequent layers. Hence, it is more effective to minimize the estimation error of each layer's response than minimize the quantization error of network parameters.

Next, we take one step further and attempt to speed-up all the convolutional layers in AlexNet with Q-CNN (EC).

Table 6: Comparison on the speed-up/compression rates and the increase of top-1/5 error rates for accelerating all the convolutional layers in AlexNet and VGG-16.

| Model | Method | Para. | Speed-up | Compression | Top-1 Err. ↑ | | Top-5 Err. ↑ | |
|---|---|---|---|---|---|---|---|---|
| | | | | | No FT | FT | No FT | FT |
| AlexNet | Q-CNN (EC) | 4/64 | 3.32× | 12.46× | 1.33% | - | 0.94% | - |
| | | 6/64 | 4.32× | 18.55× | 2.32% | - | 1.90% | - |
| | | 6/128 | 3.71× | 15.25× | 1.44% | 0.13% | 1.16% | 0.36% |
| | | 8/128 | 4.27× | 20.26× | 2.25% | 0.99% | 1.64% | 0.60% |
| VGG-16 | LANR [27] | - | 4.00× | 2.73× | - | - | 0.95% | 0.35% |
| | Q-CNN (EC) | 6/128 | 4.06× | 23.68× | 3.04% | 1.06% | 1.83% | 0.45% |

We fix the quantization hyper-parameters $(C'_s, K)$ across all layers. From Table 6, we observe that the loss in accuracy grows mildly than the single-layer case. The speed-up rates reported here are consistently smaller than those in Table 5, since the acceleration effect is less significant for some layers (i.e. "conv_4" and "conv_5"). For AlexNet, our Q-CNN model ($C'_s = 8, K = 128$) can accelerate the computation of all the convolutional layers by a factor of 4.27×, while the increase in the top-1 and top-5 error rates are no more than 2.5%. After fine-tuning the remaining fully-connected layers, the performance loss can be further reduced to less than 1%.

In Table 6, we also report the comparison against LANR [27] on VGG-16. For the similar speed-up rate (4×), their approach outperforms ours in the top-5 classification error (an increase of 0.95% against 1.83%). After fine-tuning, the performance gap is narrowed down to 0.35% against 0.45%. At the same time, our approach offers over 23× compression of parameters in convolutional layers, much larger than theirs 2.7× compression[2]. Therefore, our approach is effective in accelerating and compressing networks with many convolutional layers, with only minor performance loss.

### 5.2.2 Quantizing the Fully-connected Layer

For demonstration, we first compress parameters in a single fully-connected layer. In CaffeNet, the first fully-connected layer possesses over 37 million parameters ($9216 \times 4096$), more than 60% of whole network parameters. Our Q-CNN approach is adopted to quantize this layer and the results are as reported in Table 7. The performance loss of our Q-CNN model is negligible (within 0.4%), which is much smaller than baseline methods (DPP and SVD). Furthermore, error correction can help Q-CNN to be more effective in preserving the classification accuracy, especially under a higher compression rate.

Now we evaluate our approach's performance for compressing all the fully-connected layers in CaffeNet in Table 8. The third layer is actually the combination of 1000 classifiers, and is more critical to the classification accuracy. Hence, we adopt a much more fine-grained hyper-parameter setting ($C'_s = 1, K = 16$) for this layer. Although the speed-up effect no longer exists, we can still achieve around 8× compression for the last layer.

From Table 8, we discover that our Q-CNN models achieve high compression rates with less than 1% drop in

Table 7: Comparison on the compression rates and the increase of top-1/5 error rates for compressing the first fully-connected layer in CaffeNet, without fine-tuning.

| Method | Para. | Compression | Top-1 Err. ↑ | Top-5 Err. ↑ |
|---|---|---|---|---|
| DPP | - | 1.19× | 0.16% | - |
| | - | 1.47× | 1.76% | - |
| | - | 1.91× | 4.08% | - |
| | - | 2.75× | 9.68% | - |
| SVD | - | 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 | 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 (EC) | 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 8: Comparison on the compression rates and the increase of top-1/5 error rates for compressing all the fully-connected layers in CaffeNet. Both SVD and DFC are fine-tuned, while Q-CNN and Q-CNN (EC) are not fine-tuned.

| Method | Para. | Compression | Top-1 Err. ↑ | Top-5 Err. ↑ |
|---|---|---|---|---|
| SVD | - | 1.26× | 0.14% | - |
| | - | 2.52× | 1.22% | - |
| DFC | - | 1.79× | -0.66% | - |
| | - | 3.58× | 0.31% | - |
| Q-CNN | 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 (EC) | 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% |

[2]The compression effect of their approach was not explicitly discussed in the paper; we estimate the compression rate based on their description.

Table 9: The speed-up/compression rates and the increase of top-1/5 error rates for the whole CNN model. Particularly, for the quantization of the third fully-connected layer in each network, we let $C'_s = 1$ and $K = 16$.

| Model | Para. | | Speed-up | Compression | Top-1 Err. ↑ | Top-5 Err. ↑ |
|---|---|---|---|---|---|---|
| | Conv. | FCnt. | | | | |
| AlexNet | 8/128 | 3/32 | 4.05× | 15.40× | 1.38% | 0.84% |
| | 8/128 | 4/32 | 4.15× | 18.76× | 1.46% | 0.97% |
| CaffeNet | 8/128 | 3/32 | 4.04× | 15.40× | 1.43% | 0.99% |
| | 8/128 | 4/32 | 4.14× | 18.76× | 1.54% | 1.12% |
| CNN-S | 8/128 | 3/32 | 5.69× | 16.32× | 1.48% | 0.81% |
| | 8/128 | 4/32 | 5.78× | 20.16× | 1.64% | 0.85% |
| VGG-16 | 6/128 | 3/32 | 4.05× | 16.55× | 1.22% | 0.53% |
| | 6/128 | 4/32 | 4.06× | 20.34× | 1.35% | 0.58% |

accuracy. In contrast, the compression rates of the baselines (SVD[3]and DFC) are much smaller (less than 4×). Again, Q-CNN with error correction is consistently more effective than the naive Q-CNN approach as adopted in [10].

#### 5.2.3 Quantizing the Whole Network

So far, we have evaluated the performance of CNN models with either convolutional or fully-connected layers quantized. Now we demonstrate the quantization of the whole network with a three-stage strategy. Firstly, we quantize all the convolutional layers with error correction, while fully-connected layers remain untouched. Secondly, we fine-tune fully-connected layers in the quantized network with the ILSVRC-12 training set to restore the classification accuracy. Finally, fully-connected layers in the fine-tuned network are quantized with error correction. We report the performance of our Q-CNN models in Table 9.

For convolutional layers, we let $C'_s = 8$ and $K = 128$ for AlexNet, CaffeNet, and CNN-S, and let $C'_s = 6$ and $K = 128$ for VGG-16, to ensure roughly $4 \sim 6\times$ speed-up for each network. Then we vary the hyper-parameter settings in fully-connected layers for different compression levels. For the former two networks, we achieve 18× compression with about 1% loss in the top-5 classification accuracy. For CNN-S, we achieve 5.78× speed-up and 20.16× compression, while the top-5 classification accuracy drop is merely 0.85%. The result on VGG-16 is even more encouraging: with 4.06× speed-up and 20.34×, the increase of top-5 error rate is only 0.58%. Hence, our proposed Q-CNN framework can improve the efficiency of convolutional networks with minor performance loss, which is acceptable in many applications.

[3]In Table 7, SVD means replacing the weighting matrix with the multiplication of two low-rank matrices; in Table 8, SVD means fine-tuning the network after the low-rank matrix decomposition.

### 5.3. Results on Mobile Devices

We have developed an Android application to fulfill CNN-based image classification on mobile devices, based on our Q-CNN framework. The experiments are carried out on a Huawei® Mate 7 smartphone, equipped with an 1.8GHz Kirin 925 CPU. The test-phase computation is carried out on a single CPU core, without GPU acceleration.

In Table 10, we compare the computation efficiency and classification accuracy of the original and quantized CNN models. Our Q-CNN framework achieves 3× speed-up for AlexNet, and 4× speed-up for CNN-S. What's more, we compress the storage consumption by 20 ×, and the required run-time memory is only one quarter of the original model. At the same time, the loss in the top-5 classification accuracy is no more than 1%. Therefore, our proposed approach improves the run-time efficiency in multiple aspects, making the deployment of CNN models become tractable on mobile platforms.

Table 10: Comparison on the time, storage, memory consumption, and top-5 classification error rates of the original and quantized AlexNet and CNN-S.

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

## 6. Conclusion

In this paper, we propose a unified framework to simultaneously accelerate and compress convolutional neural networks. We quantize network parameters to enable efficient test-phase computation. Extensive experiments are conducted on MNIST and ILSVRC-12, and our approach achieves outstanding speed-up and compression rates, with only negligible loss in the classification accuracy.

## References


[1] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference (BMVC)*, 2014. 1, 2, 6

[2] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing convolutional neural networks. *CoRR*, abs/1506.04449, 2015. 1, 2

[3] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning (ICML)*, pages 2285–2294, 2015. 1, 2, 5, 6

[4] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. High-performance neural networks for visual object classification. *CoRR*, abs/1102.0183, 2011. 5, 6

[5] M. Denil, B. Shakibi, L. Dinh, M. A. Ranzato, and N. de Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2148–2156. 2013. 5, 6

[6] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1269–1277, 2014. 1, 5

[7] J. D. Geoffrey Hinton, Oriol Vinyals. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. 5, 6

[8] R. B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. 1

[9] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587, 2014. 1

[10] Y. Gong, L. Liu, M. Yang, and L. D. Bourdev. Compressing deep convolutional networks using vector quantization. *CoRR*, abs/1412.6115, 2014. 1, 2, 5, 8

[11] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR*, abs/1510.00149, 2015. 2

[12] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Conference (BMVC)*, 2014. 1

[13] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 33(1):117–128, 2011. 2

[14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093, 2014. 2, 5, 6

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1106–1114, 2012. 1, 2, 6

[16] V. Lebedev, Y. Ganin, M. Rakhuba, I. V. Oseledets, and V. S. Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. In *International Conference on Learning Representations (ICLR)*, 2015. 1, 5, 6

[17] V. Lebedev and V. S. Lempitsky. Fast convnets using group-wise brain damage. *CoRR*, abs/1506.02515, 2015. 1, 5, 6

[18] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. 1

[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 2, 5

[20] S. Ren, K. He, R. B. Girshick, and J. Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. 1

[21] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, pages 1–42, 2015. 2, 5

[22] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015. 1, 2, 6

[23] S. Srinivas and R. V. Babu. Data-free parameter pruning for deep neural networks. *CoRR*, abs/1507.06149, 2015. 1, 5

[24] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015. 1

[25] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. J. Smola, L. Song, and Z. Wang. Deep fried convnets. *CoRR*, abs/1412.7149, 2014. 1, 5

[26] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision (ECCV)*, pages 818–833, 2014. 1

[27] X. Zhang, J. Zou, K. He, and J. Sun. Accelerating very deep convolutional networks for classification and detection. *CoRR*, abs/1505.06798, 2015. 1, 5, 7

[28] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun. Efficient and accurate approximations of nonlinear convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1984–1992, 2015. 1, 5

[29] S. Zhou and J. Wu. Compression of fully-connected layer in neural network by kronecker product. *CoRR*, abs/1507.05775, 2015. 1

## Appendix A: Additional Results

In the submission, we report the performance after quantizing all the convolutional layers in AlexNet, and quantizing all the full-connected layers in CaffeNet. Here, we present experimental results for some other settings.

### Quantizing Convolutional Layers in CaffeNet

We quantize all the convolutional layers in CaffeNet, and the results are as demonstrated in Table 11. Furthermore, we fine-tune the quantized CNN model learned with error correction ($C'_s = 8, K = 128$), and the increase of top-1/5 error rates are 1.15% and 0.75%, compared to the original CaffeNet.

Table 11: Comparison on the speed-up rates and the increase of top-1/5 error rates for accelerating all the convolutional layers in CaffeNet, without fine-tuning.

| Method | Para. | Speed-up | Top-1 Err. ↑ | Top-5 Err. ↑ |
|---|---|---|---|---|
| Q-CNN | 4/64 | 3.32× | 18.69% | 16.73% |
| | 6/64 | 4.32× | 32.84% | 33.55% |
| | 6/128 | 3.71× | 20.08% | 18.31% |
| | 8/128 | 4.27× | 35.48% | 37.82% |
| Q-CNN (EC) | 4/64 | 3.32× | 1.22% | 0.97% |
| | 6/64 | 4.32× | 2.44% | 1.83% |
| | 6/128 | 3.71× | 1.57% | 1.12% |
| | 8/128 | 4.27× | 2.30% | 1.71% |

### Quantizing Convolutional Layers in CNN-S

We quantize all the convolutional layers in CNN-S, and the results are as demonstrated in Table 12. Furthermore, we fine-tune the quantized CNN model learned with error correction ($C'_s = 8, K = 128$), and the increase of top-1/5 error rates are 1.24% and 0.63%, compared to the original CNN-S.

Table 12: Comparison on the speed-up rates and the increase of top-1/5 error rates for accelerating all the convolutional layers in CNN-S, without fine-tuning.

| Method | Para. | Speed-up | Top-1 Err. ↑ | Top-5 Err. ↑ |
|---|---|---|---|---|
| Q-CNN | 4/64 | 3.69× | 19.87% | 16.77% |
| | 6/64 | 5.17× | 45.74% | 48.67% |
| | 6/128 | 4.78× | 27.86% | 25.09% |
| | 8/128 | 5.92× | 46.18% | 50.26% |
| Q-CNN (EC) | 4/64 | 3.69× | 1.60% | 0.92% |
| | 6/64 | 5.17× | 3.49% | 2.32% |
| | 6/128 | 4.78× | 2.07% | 1.32% |
| | 8/128 | 5.92× | 3.42% | 2.17% |

### Quantizing Fully-connected Layers in AlexNet

We quantize all the fully-connected layers in AlexNet, and the results are as demonstrated in Table 13.

Table 13: Comparison on the compression rates and the increase of top-1/5 error rates for compressing all the fully-connected layers in AlexNet, without fine-tuning.

| Method | Para. | Compression | Top-1 Err. ↑ | Top-5 Err. ↑ |
|---|---|---|---|---|
| Q-CNN | 2/16 | 13.96× | 0.25% | 0.27% |
| | 3/16 | 19.14× | 0.77% | 0.64% |
| | 3/32 | 15.25× | 0.54% | 0.33% |
| | 4/32 | 18.71× | 0.71% | 0.69% |
| Q-CNN (EC) | 2/16 | 13.96× | 0.14% | 0.20% |
| | 3/16 | 19.14× | 0.40% | 0.22% |
| | 3/32 | 15.25× | 0.40% | 0.21% |
| | 4/32 | 18.71× | 0.46% | 0.38% |

### Quantizing Fully-connected Layers in CNN-S

We quantize all the fully-connected layers in CNN-S, and the results are as demonstrated in Table 14.

Table 14: Comparison on the compression rates and the increase of top-1/5 error rates for compressing all the fully-connected layers in CNN-S, without fine-tuning.

| Method | Para. | Compression | Top-1 Err. ↑ | Top-5 Err. ↑ |
|---|---|---|---|---|
| Q-CNN | 2/16 | 14.37× | 0.22% | 0.07% |
| | 3/16 | 20.15× | 0.45% | 0.22% |
| | 3/32 | 15.79× | 0.21% | 0.11% |
| | 4/32 | 19.66× | 0.35% | 0.27% |
| Q-CNN (EC) | 2/16 | 14.37× | 0.36% | 0.14% |
| | 3/16 | 20.15× | 0.43% | 0.24% |
| | 3/32 | 15.79× | 0.29% | 0.11% |
| | 4/32 | 19.66× | 0.56% | 0.27% |

## Appendix B: Optimization in Section 3.3.2

Assume we have $N$ images to learn the quantization of a convolutional layer. For image $I_n$, we denote its input feature maps as $S_n \in \mathbb{R}^{d_s \times d_s \times C_s}$ and response feature maps as $T_n \in \mathbb{R}^{d_t \times d_t \times C_t}$, where $d_s, d_t$ are the spatial sizes and $C_s, C_t$ are the number of feature map channels. We use $p_s$ and $p_t$ to denote the spatial location in the input and response feature maps. The spatial location in the convolutional kernels is denoted as $p_k$.

To learn quantization with error correction for the convolutional layer, we attempt to optimize:

$$\min_{\{D^{(m)}\},\{B^{(m)}_{p_k}\}} \sum_{n,p_t} \left\| \sum_{(p_k,p_s)} \sum_{m} (D^{(m)} B^{(m)}_{p_k})^T S^{(m)}_{n,p_s} - T_{n,p_t} \right\|_F^2 \tag{14}$$

where $D^m$ is the $m$-th sub-codebook, and $B^{(m)}_{p_k}$ is the corresponding sub-codeword assignment indicator for the convolutional kernels at spatial location $p_k$.

Similar to the fully-connected layer, we adopt a block coordinate descent approach to solve this optimization problem. For the $m$-th subspace, we firstly define its residual

feature map as:

$$R_{n,p_t}^{(m)} = T_{n,p_t} - \sum_{(p_k,p_s)} \sum_{m' \neq m} (D^{(m')} B_{p_k}^{(m')})^T S_{n,p_s}^{(m')} \quad (15)$$

and then the optimization in the $m$-th subspace can be reformulated as:

$$\min_{D^{(m)},\{B_{p_k}^{(m)}\}} \sum_{n,p_t} \left\| \sum_{(p_k,p_s)} (D^{(m)} B_{p_k}^{(m)})^T S_{n,p_s}^{(m)} - R_{n,p_t} \right\|_F^2 \quad (16)$$

**Update $\boldsymbol{D^{(m)}}$.** With the assignment indicator $\{B_{p_k}^{(m)}\}$ fixed, we let:

$$L_{k,p_k} = \{c_t | B_{p_k}^{(m)}(k, c_t) = 1\} \quad (17)$$

We greedily update each sub-codeword in the $m$-th sub-codebook $D^{(m)}$ in a sequential style. For the $k$-th sub-codeword, we compute the corresponding residual feature map as:

$$Q_{n,p_t,k}^{(m)}(c_t) = R_{n,p_t}^{(m)}(c_t) - \sum_{(p_k,p_s)} \sum_{k' \neq k} \sum_{c_t \in L_{k',p_k}} D_{k'}^{(m)^T} S_{n,p_s}^{(m)} \quad (18)$$

and then we can alternatively optimize:

$$\min_{D_k^{(m)}} \sum_{n,p_t} \left\| \sum_{(p_k,p_s)} \sum_{c_t \in L_{k,p_k}} D_k^{(m)^T} S_{n,p_s}^{(m)} - Q_{n,p_t,k}^{(m)}(c_t) \right\|_F^2 \quad (19)$$

which can be transformed into a least square problem. By solving it, we can update the $k$-th sub-codeword.

**Update $\{\boldsymbol{B_{p_k}^{(m)}}\}$.** We greedily update the sub-codeword assignment at each spatial location in the convolutional kernels in a sequential style. For the spatial location $p_k$, we compute the corresponding residual feature map as:

$$P_{n,p_t,p_k}^{(m)} = R_{n,p_t}^{(m)} - \sum_{\substack{(p'_k,p'_s) \\ p_k \neq p_k}} (D^{(m)} B_{p'_k}^{(m)})^T S_{n,p'_s}^{(m)} \quad (20)$$

and then the optimization can be re-written as:

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

Since $B_{p_k}^{(m)} \in \{0,1\}^K$ is an indicator vector (only one non-zero entry), we can exhaustively try all sub-codewords and select the optimal one that minimize the objective function.

View publication stats