Towards Convolutional Neural Networks Compression via Global&Progressive Product Quantization

Weihan Chen[1,2]
chenweihan2018@ia.ac.cn
Peisong Wang[1]
peisong.wang@nlpr.ia.ac.cn
Jian Cheng[1*]
jcheng@nlpr.ia.ac.cn

[1] NLPR & AIRIA, Institute of Automation, Chinese Academy of Sciences
Beijing, China

[2] School of Artificial Intelligence, University of Chinese Academy of Sciences
Beijing, China

Abstract

In recent years, we have witnessed the great success of convolutional neural networks in a wide range of visual applications. However, these networks are typically deficient due to the high cost in storage and computation, which prohibits their further extensions to resource-limited applications. In this paper, we introduce Global&Progressive Product Quantization(G&P PQ), an end-to-end product quantization based network compression method, to merge the separate quantization and finetuning process into a consistent training framework. Compared to existing two-stage methods, we avoid the time-consuming process of choosing layer-wise finetuning hyperparameters and also make the network capable of learning complex dependencies among layers by quantizing globally and progressively. To validate the effectiveness, we benchmark G&P PQ by applying it to ResNet-like architectures for image classification and demonstrate state-of-the-art tradeoff in terms of model size vs. accuracy under extensive compression configurations compared to previous methods.

1 Introduction

Recently, Convolutional Neural Networks(CNNs) have demonstrated record-breaking results for a wide range of visual applications, including image recognition [9], object detection [16], etc. However, these unprecedented performances are mostly built on top of deeper and wider network architecture with large model size, which leads to very high computation and storage overhead. Consequently, it is intractable to deploy these models on resource-limited hardware such as mobile phones and other embedded devices. Under this circumstance, a variety of methods have been proposed, including knowledge distillation [10], low-precision quantization [8], etc, to achieve CNNs compression and acceleration.

To exploit the spatial redundancy of information inherent to standard convolution filters [6], here we focus on Product Quantization(PQ). The essence of product quantization is to decompose the original high-dimensional space into the Cartesian product of a finite

number of low-dimensional subspaces that are then quantized separately. In this way, the original weight matrix can be represented by the codeword indexes of all the subvectors and the corresponding codebook, which reduce the storage cost of the network significantly yet maintain accuracy close to their uncompressed counterpart.

Given the compression advantages of product quantization, several prior works [7, 18, 20] have introduced it into the regime of network quantization. In summary, the procedure of existing methods consists of two separate stages: **(1)** perform direct quantization on the network sequentially by minimizing the reconstruction error of the weights or activations. **(2)** finetune the codebooks of all the layers globally to compensate for the performance drop due to quantization. *However, we argue that there are two deficiencies in the above procedure.* First, to alleviate the accumulated performance drop while quantizing the deep model sequentially, it is preferable to finetune locally after quantization of each layer or block of layers. As different layer possesses different quantization sensitivity, it is time-consuming to choose proper finetuning hyperparameters for each of them to achieve the best performance. Second, though it is more accessible to handle separately by splitting the whole compression process into two stages and quantizing each layer in sequence, the inconsistency between quantization and finetuning and the imposed constraint on quantization order make it difficult for the network to adapt to global and complex dependencies among layers and hence incline to fall into poor local optimum.

To solve these problems, we introduce Global&Progressive Product Quantization(G&P PQ), an end-to-end product quantization based network compression method that merges the separate quantization and finetuning process into a consistent training framework. Compared to prior works, instead of quantizing each subvector directly, we accomplish the progressive quantization globally during the training process. In addition, as the original task-specific loss function(e.g., classification loss) is kept and will gradually dominate the update of the network as training goes on, the switch between quantization and finetuning is completed progressively and automatically. As a result, our method is capable of modeling global dependency among layers and optimizing quantization and finetuning process coordinately, which leads to better compression performance. Finally, with only introduced hyperparameter of a coefficient that controls the speed of quantization compared to standard training algorithm, it is convenient to implement and deploy it to different network architectures and tasks without the need of tuning hyperparameters for each layer. To validate the effectiveness, we benchmark G&P PQ by applying it to ResNet-like architectures for image classification tasks and demonstrate state-of-the-art tradeoff in terms of model size vs. accuracy.

2 Related Work

There is a large amount of literature on CNN compression and acceleration, here we only review the works related to ours and refer the reader to recent survey [8] for a comprehensive overview.

Low-Precision Quantization As full-precision parameters are not required in achieving high performance in CNNs, low-bit quantization of deep network parameters has recently received increasing interest. By utilizing fixed-point weights representation, the model size can be dramatically reduced by an order of magnitude (up to $\sim 32\times$). [22] proposed to quantize the parameter incrementally and showed that with reduced weight precision to 2-5 bits, classification accuracy on the ImgeNet dataset could be even slightly higher. Furthermore, [3] constrained the weights to binary(e.g., -1 or $+1$) values to obtain acceleration in the

inference stage by replacing many multiply-accumulate operations with simple accumulations. However, acceleration gain via weight quantization is limited because of real-valued intermediate activations. Recently, several works focused on quantizing both weights and activations while minimizing performance degradation, which include [23], etc. Additionally, [4] introduced BinaryNet with binary weights and activations at run-time. Hence during the forward pass, most multiplications can be done by efficient $popcnt - xnor$ operations. [15] improved BinaryNet by introducing scale factors for both weights and activations during the binarization process, which significantly contributes to accuracy improvement. To further compensate for the accuracy loss of binarization, [13] introduced 1-bit CNNs aided with a real-valued shortcut. On the other hand, multi-bit networks [12] decompose a single convolution layer into K binary convolution operations to achieve higher accuracy.

Product Quantization Product Quantization is an effective vector quantization approach that has been extensively studied in the context of fast approximate nearest neighbor search [11]. To reduce the cardinality of the representation space, a high dimensional vector is divided into several low dimensional subvectors, and each of them is vector-quantized to its nearest codeword in a predefined codebook. Afterward, the inner product between two vectors can be efficiently estimated from their codes in the symmetric way or the vector and code in the asymmetric way [11]. To the best of our knowledge, [7] was the first to explore vector quantization methods in CNN compression and focused upon how to compress the fully-connected layers to reduce the storage of neural networks. After that, [20] proposed to quantize by minimizing the estimation error of the activations at each layer, and introduced a unified framework to apply product quantization to both convolutional and fully-connected layers, which simultaneously accelerate and compress CNN models. Recently, [18] shared the similar idea that employed each layer's response reconstruction error as an alternative optimization objective and verified product quantization's effectiveness in ResNet-like networks that with complicated structures through combining other strategies such as codebook finetuning and knowledge distillation.

3 Methodology

3.1 Formulation

Generally speaking, quantization is a destructive process that aims to reduce the cardinality of the representation space while minimizing the task-specific distortion that comes with it. In the context of product quantization, given a D-dimensional random vector $\mathbf{x} \in \mathbb{R}^D$, we first divide it into M contiguous parts of subvectors.

$$\begin{aligned}\mathbf{x} &= [\underbrace{x_1, x_2, \ldots, x_{D/M}}_{\mathbf{x}^{1T}}, \ldots, \underbrace{x_{D-D/M+1}, \ldots, x_D}_{\mathbf{x}^{MT}}]^T \\ &= [\mathbf{x}^{1T}, \ldots, \mathbf{x}^{MT}]^T,\end{aligned} \tag{1}$$

where the m-th subvector is denoted as $\mathbf{x}^m \in \mathbb{R}^{D/M}$ for each $m \in \{1, \ldots, M\}$. To describe formally, we first define a task-specific distortion function $e : \mathbb{R}^{D/M} \times \mathbb{R}^{D/M} \mapsto \mathbb{R}_+$ which measures the distance between the original subvector and quantized one. Then each quantizer consists of two parts:

- a sub-codebook $\mathcal{C}^m = \{\mathbf{c}_k^m\}_{k=1}^K$ for each $m \in \{1, \ldots, M\}$ where we call $\mathbf{c}_k^m \in \mathbb{R}^{D/M}$ as a sub-codeword.

- a quantization function $q : \mathbb{R}^D \mapsto \mathcal{C}^1 \times \cdots \times \mathcal{C}^M$ defined as the concatenation of M sub ones $q^m : \mathbb{R}^{D/M} \mapsto \mathcal{C}^m$.

Finally, we describe the quantization formally as solving the following optimization problem.

$$\min_{\{\mathcal{C}^m, q^m\}_{m=1}^M} \quad E_{\mathbf{x}}[e(\mathbf{x}, q(\mathbf{x}))] = \int_{\mathbf{x}} p(\mathbf{x}) e(\mathbf{x}, q(\mathbf{x})) d\mathbf{x}, \tag{2}$$

where $p(\mathbf{x})$ denotes the probability density function of $\mathbf{x}$. In practice, the above objective function is approximated through Monte Carlo sampling according to the Law of Large Numbers. *Another thing that should be noted is that if $\mathcal{C}^i = \mathcal{C}^j$ for all $i \neq j$ and $i, j \in \{1, \ldots, M\}$ then we use a joint codebook for all the subvectors, which is the default setting of this paper.*

3.2 Compression for Single Layer

Here we focus on the fully-connected and convolutional layer as they dominate the weights of convolutional networks. Without loss of generality, we omit to quantize the bias as its storage cost is negligible.

Fully-Connected Layer For a fully-connected layer, we denote its weight matrix as $W \in \mathbb{R}^{C_{in} \times C_{out}}$, where C_{in} and C_{out} are the dimensions of layer input and output, respectively. The weight vector W^p is the p-th column of vector in W. Then we split each $W^p, p \in 1, \ldots, C_{out}$ evenly into M contiguous subvectors and learn a joint codebook $\mathcal{C} = \{\mathbf{c}_k\}_{k=1}^K$ on the resulting $M \times C_{out}$ subvectors. For simplicity, we suppose that C_{in} is a multiple of M, *i.e.*, all the subvectors have the same dimension $d = C_{in}/M$. Consequently, we denote the quantized weight vector as

$$q(W^p) = [\mathbf{c}_{\psi(W_1^p)}{}^T, \ldots, \mathbf{c}_{\psi(W_M^p)}{}^T]^T, \tag{3}$$

where $\psi(W_m^p) : \mathbb{R}^d \mapsto \{1, \ldots, K\}$ denotes the index of codeword in the codebook for the mth subvector in W^p. More specifically, in this paper we utilize *Mean Square Error* as our distortion function, then we can minimize the objective of (2) alternatively via k-means algorithm.

Convolutional Layer We denote $W \in \mathbb{R}^{C_{out} \times C_{in} \times k_h \times k_w}$ as the weight matrix of a convolutional layer with C_{out} output channles, C_{in} input channels, and $k_h \times k_w$ kernel size. There are many ways to split a $4D$ matrix into a set of vectors and apply product quantization. In this paper, we follow [7] that first reshape W into a $2D$ matrix $\hat{W}$ of size $(C_{in} \times k_h \times k_w) \times C_{out}$ and then split along each column evenly into contiguous subvectors. As a result, we can deal with the convolutional layer in the same way as the fully-connected layer.

Storage Analysis To employ the network after product quantization, we need to store the codeword indexes and codebook of each quantized layer. As an example, for a fully-connected layer with weight matrix $W \in \mathbb{R}^{C_{in} \times C_{out}}$ and original size of $\mathbf{32} \times \boldsymbol{C_{in}} \times \boldsymbol{C_{out}}$ *bit*, the storage cost of codeword indexes and codebook are $\boldsymbol{M} \times \boldsymbol{C_{out}} \times \boldsymbol{log_2(K)}$ *bit* and $\mathbf{16} \times \boldsymbol{(C_{in}/M)} \times \boldsymbol{K}$ *bit*, respectively. Here we adopt length-fixed coding for codeword indexes and store the codebook in half precision following the prior work [18].

3.3 Global&Progressive Product Quantization

Given a convolutional neural network of N layers with task-specific loss function f and training images X, we denote its parameters set as $\{W^{(l)}\}_{l=1}^{N}$. Here we only consider weights of fully-connected and convolutional layers, as explained earlier. Then we can formulate product quantization on CNN as the following constrained optimization problem

$$\begin{aligned}
&\min_{\{\{W_j^{(l)}\}_{j=1}^{M^{(l)}},\mathcal{I}^{(l)}\}_{l=1}^{N}} f(\{\{q(W_j^{(l)})\}_{j=1}^{M^{(l)}}\}_{l=1}^{N};X) \\
&\text{s.t.} \qquad q(W_j^{(l)}) = \mathbf{c}^{(l)}_{\psi^{(l)}(W_j^{(l)})} \\
&\mathbf{c}_k^{(l)} = \arg\min_{\mathbf{c}} \frac{1}{|\mathcal{I}^{(l)}(k)|} \sum_{i\in\mathcal{I}^{(l)}(k)} e(W_i^{(l)},\mathbf{c}) \\
&l \in \{1,\dots,N\}, j \in \{1,\dots,M^{(l)}\}, k \in \{1,\dots,K^{(l)}\}
\end{aligned} \tag{4}$$

where we denote $W_j^{(l)}$, $M^{(l)}$, $K^{(l)}$ as the j-th subvector and the number of subvectors, codewords of layer l. Besides, we denote $\mathcal{I}^{(l)}(k)$ as the index set of subvectors quantized to the k-th codeword in layer l, based on which we can get $\psi^{(l)}$. Other notations follow the definitions mentioned earlier. We only consider *Mean Square Error* as the distortion function and hence rewrite (4) as

$$\mathbf{c}_k^{(l)} = \frac{1}{|\mathcal{I}^{(l)}(k)|} \sum_{i\in\mathcal{I}^{(l)}(k)} W_i^{(l)}. \tag{5}$$

To solve the above problem, prior works adopt a two-stage pipeline, which first performs direct quantization layer-by-layer and then finetunes the quantized weights to compensate for performance drop. Due to the deficiencies mentioned earlier, we propose to merge the separate stage into a consistent training framework. Specifically, we start from a pre-trained model and calculate the index set $\mathcal{I}^{(l)}(k)$ for each layer through k-means algorithm. Then instead of quantizing each subvector directly according to the cluster index, we employ the following gradient-based update rule to train the network end-to-end.

$$\begin{aligned}
W_j^{(l)} &\leftarrow W_j^{(l)} - \eta \Delta W_j^{(l)} \\
\Delta W_j^{(l)} &= \frac{1}{|\mathcal{I}^{(l)}(\psi^{(l)}(W_j^{(l)}))|} \sum_{i\in\mathcal{I}^{(l)}(\psi^{(l)}(W_j^{(l)}))} \frac{\partial f}{\partial W_i^{(l)}} \\
&\quad + \mu\left(W_j^{(l)} - \mathbf{c}^{(l)}_{\psi^{(l)}(W_j^{(l)})}\right),
\end{aligned} \tag{6}$$

where η and μ denote the learning rate and introduced hyperparameter that controls the speed of quantization, respectively. To understand how the above rule works, let us split the update term $\Delta W_j^{(l)}$ into the former and latter part and analyze independently. First, we define the quantization loss of network as

$$\Gamma = \frac{1}{N}\sum_{l=1}^{N}\Big(\frac{1}{M^{(l)}}\sum_{j=1}^{M^{(l)}} \big\|W_j^{(l)} - \mathbf{c}^{(l)}_{\psi^{(l)}(W_j^{(l)})}\big\|^2\Big). \tag{7}$$

Base on the above update formula and assume learning rate $\eta = 1$, it is easy to derive that the quantization loss of the t-th and $(t+1)$-th iteration satisfy

$$\Gamma^{t+1} = (1-\mu)^2 \times \Gamma^t. \tag{8}$$

Therefore, with a properly chosen μ, we are mainly quantizing all the weights globally and progressively in the initial phase of training. Then as the training goes on, on the one hand, the decreased quantization loss causes the latter part of the update term infinitely small. And on the other hand, task-specific loss(e.g., classification loss) increases gradually due to the progressive quantization. Both of these lead the former part to dominates the update process. Consequently, the update term is nearly equivalent to

$$\begin{aligned}
\Delta W_j^{(l)} &= \frac{1}{|\mathcal{I}^{(l)}(\psi^{(l)}(W_j^{(l)}))|} \sum_{i \in \mathcal{I}^{(l)}(\psi^{(l)}(W_j^{(l)}))} \frac{\partial f}{\partial W_i^{(l)}} \\
&= \frac{1}{|\mathcal{I}^{(l)}(\psi^{(l)}(W_j^{(l)}))|} \sum_{i \in \mathcal{I}^{(l)}(\psi^{(l)}(W_j^{(l)}))} \frac{\partial f}{\partial q(W_i^{(l)})} \\
&= \frac{\partial f}{\partial \mathbf{c}^{(l)}_{\psi^{(l)}(W_j^{(l)})}},
\end{aligned} \tag{9}$$

which actually finetunes the codeword by averaging the gradients of all the subvectors quantized to it and compensates for the performance drop. Hence we can see the switch between quantization and finetuning is completed progressively and automatically, and the relative speed is controlled by choosing different μ. Once the training terminates, we get the quantized network without the need for further finetuning. We will discuss the training dynamics in detail in section 4.3. Besides, we can integrate the update rule into any other gradient-based optimization algorithms without affecting the intrinsic mechanism. In this paper, we combine the above formula with momentum to enhance its performance in deep network.

Architecture	Compression Regime	$d_{3\times3}$	$d_{1\times1}$	d_{fc}	$K_{3\times3}$	$K_{1\times1}$	K_{fc}
ResNet-18	Small blocks	9	4	4	256	256	2048
	Large blocks	18	4				
ResNet-50	Small blocks	9	4	4	256	256	1024
	Large blocks	18	8				

Table 1: Compression configurations details. Here the $3^{th}-5^{th}$ and $6^{th}-8^{th}$ columns denote the subvector and codebook size of 3×3 conv, 1×1 conv and fully-connected layers, respectively.

4 Experiments

4.1 Experimental Setup

To verify the effectiveness of the proposed method, we quantize standard ResNet-18 and ResNet-50 [9] architectures with pre-trained weights. Unless explicit mention of the con-

trary, the models are pre-trained on ImageNet dataset [5] and taken from the Pytorch model zoo*. We detail our experimental setup below.

Compression Regimes To explore the tradeoff between compression ratio and accuracy, we vary between small block size and large block size compression regimes. In the large blocks regime, we utilize a larger subvector size of d for each layer, which leads to less storage cost of codeword indexes and higher compression ratios. Except for subvector size, we also adopt the codebook of different sizes for various architectures and types of layers. We summarize the details of compression configurations in Table 1 as a reference. Note that following [18] we also clamp the number of codewords to $\min(K, C_{out} \times M/4)$ for stability and skip the quantization of the first convolutional layer of kernel size 7×7 as it represents less than 0.1%(resp., 0.05%) of the weights of ResNet-18(resp., ResNet-50).

Training Hyparameters We train each network under different compression configurations for 30 epochs with weight decay of $1e-8$, momentum of 0.9, and initial learning rate of 0.01. For memory reason, the batch size of ResNet-18 and ResNet-50 are 256 and 128, respectively. As the method is robust to the value of μ in a broad range(see Table 5), we set it to $1e-3$ by default. Besides, we utilize the ReduceLROnPlateau scheduler based on PyTorch [14], which reduces the learning rate by $10\times$ whenever the validation loss plateaus.

Compression Regime	Method	Size ratio	Model size	Top-1 (%)
Small blocks	[18]	29×	1.54 MB	65.81
	G&P PQ(Ours)			**66.75**
Large blocks	[18]	43×	1.03 MB	61.10
	G&P PQ(Ours)			**63.31**
Original	-	1×	44.6 MB	69.76

Table 2: Compression Results for vanilla ResNet-18 with $k = 256$ centroids

Compression Regime	Method	Size ratio	Model size	Top-1 (%)
Small blocks	[18]	19×	5.09 MB	73.79
	G&P PQ(Ours)			**75.22**
Large blocks	[18]	31×	3.19 MB	68.21
	G&P PQ(Ours)			**72.10**
Original	-	1×	97.5 MB	76.15

Table 3: Compression Results for vanilla ResNet-50 with $k = 256$ centroids

4.2 Classification Results

In this section, we compare our results against [18], the state of the art that applies product quantization to CNNs compression with the two-stage pipeline, and some other low-precision quantization methods .

Vanilla ResNet-18 and ResNet-50 First, we compress standard ResNet-18 and ResNet-50 network with different configurations detailed in Table 1. The comparison results against

*https://pytorch.org/docs/stable/torchvision/models

[18] are summarized in Table 2 and 3. As shown in the tables, our method outperforms [18] in all the configurations to a different extent. Besides, we can observe a general rule from the results that the deeper the network and the higher the compression ratio are, the larger the improvement is. In fact, the largest improvement comes from large blocks regime in ResNet-50, in which we get a 3.9% absolute boost of Top-1 accuracy. As mentioned earlier, we argue that the inconsistency between quantization and finetuning process and the imposed constraint on quantization order of two-stage method make it difficult for the network to adapt to global and complex dependencies among layers and hence incline to fall into poor local optimum. Intuitively, these deficiencies will be further enlarged for deeper networks with a higher compression ratio. Therefore we believe the improvements owe to the fusion of quantization and finetuning so that we can optimize both processes globally and coordinately.

To justify the method in a broader range of compression regimes, we vary the codebook size of the convolutional layer among $\{256, 512, 1024, 2048\}$ in ResNet-18/50, and keep other configurations the same for large blocks and small blocks regimes, respectively. The results are summarized in Figure 1. It is clearly shown that our method exhibits superior performance consistently under such extensive compression configurations. Also as shown in 1, we compare our results with other low-precision quantization methods, which include Trained Ternary Quantization [24], LR-Net [17], ABC-Net [12], Binary Weight Networks [15], Deep Compression (DC)[2], Hardware-Aware Automated Quantization (HAQ)][1], to demonstrate our advantages in the tradeoff between model size and accuracy compared to other network compression methods. We believe these advantages owe to the exploitation of spatial redundancy inherent to standard convolution filters through product quantization.

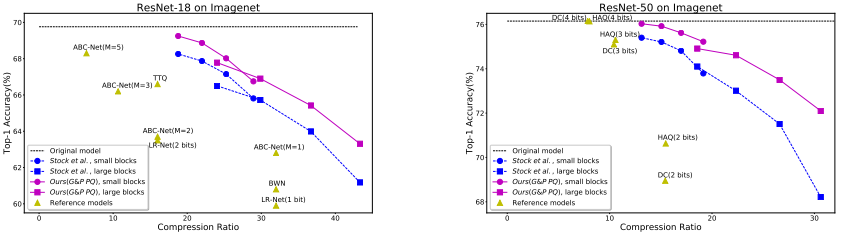


Figure 1: Compression results for ResNet-18 and ResNet-50 architectures.

Semi-supervised ResNet-50 Recently, Yalniz et al. [21] use the publicly available YFCC-100M dataset [19] to train a semi-supervised ResNet-50 that reaches 79.23% top-1 accuracy on the standard validation set of ImageNet. We test our method on this particular model in the small block regime with a 19× compression ratio. As shown in Table 4, our compressed semi-supervised ResNet-50 reaches 77.55% top-1 accuracy, which outperforms the performance of a vanilla and non-compressed ResNet-50 significantly, with the model size of only 5MB.

4.3 Method Analysis

Sensitivity Study Compared to standard network training, there is only one introduced hyperparameter that controls the speed of quantization. We conduct a series of comparative experiments on the large blocks regime of ResNet-18 with different values of μ. The results

Method	Size ratio	Model size	Top-1 (%)
[18]	19×	5.09 MB	76.12
G&P PQ(Ours)			**77.55**
Uncompressed	1×	97.5 MB	79.23

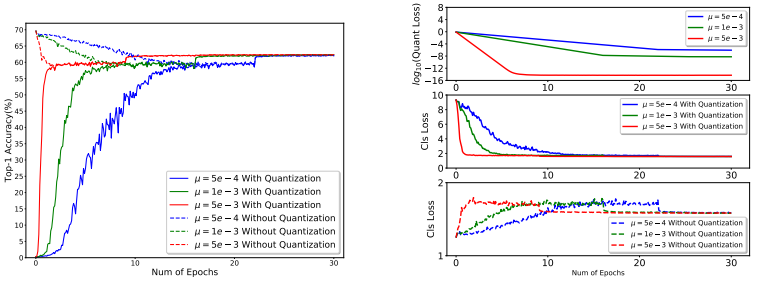
Table 4: Compression Results for semi-supervised ResNet-50

Value of μ	Top-1 (%)	Top-5 (%)
$1e-4$	51.19	76.10
$5e-4$	63.20	85.10
$\mathbf{1e-3}$	**63.31**	**85.11**
$5e-3$	63.26	85.06
$1e-2$	62.22	85.06

Table 5: Ablation study for the effect of different values of μ.

are summarized in Table 5. Our conclusion consists of two parts. On the one hand, improper selection of μ(e.g., $1e-4$) would lead the network converges to a suboptimal accuracy. On the other hand, our method still demonstrates robustness as it is able to achieve significant improvement compared to the prior method for different μ(e.g., from $1e-3$ to $1e-2$) in a broad range.

Training Dynamics We perform a quantitative study on the training dynamics of ResNet-18 with large block compression regime and summarize the results in Figure 2. As the top part of Figure 2(b) shows, quantization loss declines exponentially at the very beginning of training, which conforms with the formula (8), and converges to infinitely small finally. Furthermore, during the training process, we quantize the weights of the network and evaluate its accuracy, which denoted as **With Quantization**, to compare with the accuracy of original weights, which denoted as **Without Quantization**. As depicted in Figure 2(a), due to the progressive quantization, With Quantization accuracy increases quickly at the beginning and Without Quantization one decreases instead. As the quantization is completed progressively, these two kinds of accuracy converge to the same. Meanwhile, we switch to the finetuning process gradually and then compensate for the accuracy drop.



(a) The accuracy of network as training goes on (b) The losses of network as training goes on

Figure 2: Training dynamics of G&P PQ

5 Conclusion

We introduce Global&Progressive Product Quantization(G&P PQ), an end-to-end product quantization based network compression method that merges the separate quantization and finetuning process into a single training process. To verify its advantages, we apply it to ResNet-like architectures for image classification task and demonstrated state-of-the-art tradeoff in terms of model size vs. accuracy.

Acknowledgement

This work was supported in part by National Natural Science Foundation of China (No.61906193).

References

[1] *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, 2018. Computer Vision Foundation / IEEE. URL http://openaccess.thecvf.com/CVPR2019.py.

[2] Yoshua Bengio and Yann LeCun, editors. *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL https://iclr.cc/archive/www/doku.php%3Fid=iclr2016:accepted-main.html.

[3] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 3123–3131, 2015. URL http://papers.nips.cc/paper/5647-binaryconnect-training-deep-neural-networks-with-binary-

[4] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255, 2009. doi: 10.1109/CVPRW.2009.5206848. URL https://doi.org/10.1109/CVPRW.2009.5206848.

[6] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277, 2014.

[7] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[8] Yunhui Guo. A survey on methods and theories of quantized neural networks. *arXiv preprint arXiv:1808.04752*, 2018.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778, 2016.

[10] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. URL http://arxiv.org/abs/1503.02531.

[11] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33 (1):117–128, 2010.

[12] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 344–352, 2017.

[13] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XV*, pages 747–763, 2018. doi: 10.1007/978-3-030-01267-0_44. URL https://doi.org/10.1007/978-3-030-01267-0_44.

[14] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[15] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, pages 525–542, 2016. doi: 10.1007/978-3-319-46493-0_32. URL https://doi.org/10.1007/978-3-319-46493-0_32.

[16] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 91–99, 2015. URL http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-re

[17] Oran Shayer, Dan Levi, and Ethan Fetaya. Learning discrete weights using the local reparameterization trick. *arXiv preprint arXiv:1710.07739*, 2017.

[18] Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. And the bit goes down: Revisiting the quantization of neural networks. *arXiv preprint arXiv:1907.05686*, 2019.

[19] Bart Thomee, David A Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. Yfcc100m: The new data in multimedia research. *Communications of the ACM*, 59(2):64–73, 2016.

[20] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[21] I Zeki Yalniz, Hervé Jégou, Kan Chen, Manohar Paluri, and Dhruv Mahajan. Billion-scale semi-supervised learning for image classification. *arXiv preprint arXiv:1905.00546*, 2019.

[22] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. In *International Conference on Learning Representations (ICLR)*, 2017. URL http://arxiv.org/abs/1702.03044.

[23] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv*, abs/1606.06160, 2016. URL http://arxiv.org/abs/1606.06160.

[24] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.