And the Bit Goes Down: Revisiting the Quantization of Neural Networks

Pierre Stock[1,2], Armand Joulin[1], Rémi Gribonval[2], Benjamin Graham[1], Hervé Jégou[1]
[1]Facebook AI Research, [2]Univ Rennes, Inria, CNRS, IRISA

Abstract

In this paper, we address the problem of reducing the memory footprint of ResNet-like convolutional network architectures [1]. We introduce a vector quantization method that aims at preserving the quality of the reconstruction of the network outputs and not its weights. The advantage of our approach is that it minimizes the loss reconstruction error for *in-domain* inputs and does not require any labelled data. We also use byte-aligned codebooks to produce compressed networks with efficient inference on CPU. We validate our approach by quantizing a high performing ResNet-50 model [2] to a memory size of 5 MB ($20\times$ compression factor) while preserving a top-1 accuracy of 76.1% on ImageNet object classification and by compressing a Mask R-CNN with a size budget around 6 MB.[1]

1 Introduction

The performance of convolutional networks, or ConvNets, on image classification has steadily improved since the introduction of AlexNet [3]. This progress has been fueled by deeper and richer architectures such as the ResNets [1] and their variants ResNeXts [4] and DenseNets [5]. Those models particularly benefit from the recent progress made with weak supervision [6, 2, 7], which leverage GPUs with more memory and larger training datasets. These improvements are likely to continue and also transfer well to other vision applications [8]. There is a growing need compressing the best ConvNets for applications on embedded devices like robotics or virtual/augmented reality.

Compression of ConvNets has been an active research topic in the recent years, leading to networks with a 71% top-1 accuracy on ImageNet object classification that fits in 1 MB [9]. Most of this progress stems from compression algorithms tailored for mobile-efficient architectures like MobileNets [10] or ShuffleNets [11]. A lot of work focuses on improving the performance of these architectures [12], but they are still behind the best ConvNets by a large margin. For instance, a MobileNet-v2 achieves 72% top-1 accuracy while the latest ResNet-based models are now at 83.1% [13]. Improving the quality of image classifiers on embedded devices may require to depart from these mobile-efficient models and to focus on more traditional ConvNets. A difficulty is that these models have completely different architectures. Compression methods designed for mobile-efficient ConvNets do not transfer convincingly to ResNet-like networks, and vice versa. For example, MobileNets separate convolutions into two distinct operations that minimize the correlation between their weights. This design choice significantly reduces their memory footprint but makes structured-based compression algorithms irrelevant. On the contrary, standard ConvNets are based on standard convolutions where the correlations between the filters can be exploited.

In this work, we propose a compression method particularly adapted to ResNet-like architectures. Our approach takes advantage of the high correlation in the convolutions by the use of a structured quantization algorithm, Product Quantization (PQ) [14]. We focus on byte-aligned indexes (8-bits) as they lead to efficient inference on standard hardware, as opposed to entropy decoders [15].

[1]Code and compressed models available at `http://github.com/facebookresearch/kill-the-bits`

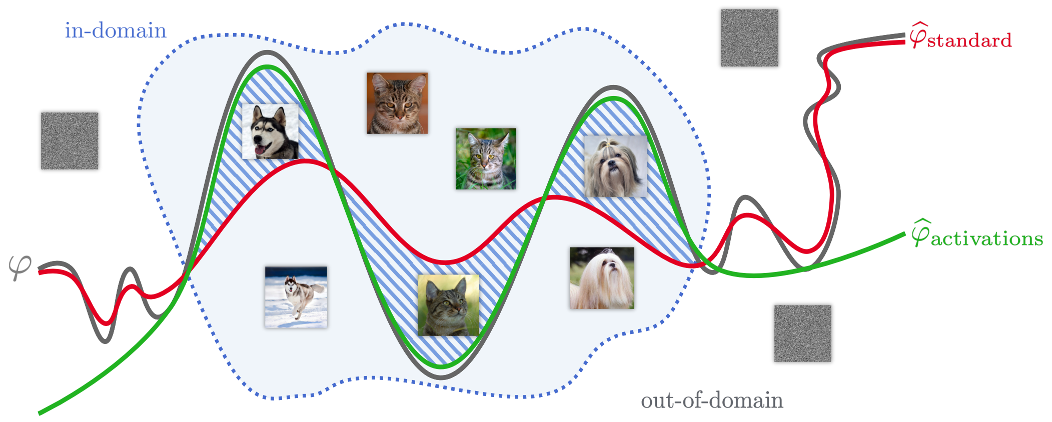
in-domain
$\widehat{\varphi}_{\text{standard}}$
φ
$\widehat{\varphi}_{\text{activations}}$
out-of-domain

Figure 1: Illustration of our method. We approximate a binary classifier φ that labels images as dogs or cats by quantizing its weights. **Standard method**: quantizing φ with the standard objective function (1) leads to a classifier $\widehat{\varphi}_{\text{standard}}$ that tries to approximate φ over the entire input space and can thus perform badly for in-domain inputs. **Our method**: quantizing φ with our objective function (2) leads to a classifier $\widehat{\varphi}_{\text{activations}}$ that performs well for in-domain inputs. Images lying in the hatched area of the input space are correctly classified by $\varphi_{\text{activations}}$ but incorrectly by $\varphi_{\text{standard}}$.

One of the challenges with applying standard sequential compression methods to modern ConvNets is their depth: the errors resulting from the compression accumulate when going up, resulting in a drift in performance. We reduce this issue by guiding the sequential compression with the uncompressed network. This allows for both an efficient layer-by-layer compression procedure and a global fine-tuning of the codewords based on distillation [16].

Finally, our approach departs from traditional scalar [15] and vector [17] quantizers by focusing on the reconstruction of the activations, not of the weights. As depicted by Figure 1, this allows a better *in-domain* reconstruction and does not require any supervision: we only need a set of unlabelled input images to build the internal representation. We show that applying our approach to the semi-supervised ResNet-50 of Yalniz *et al.* [2] leads to a 5 MB memory footprint and a 76.1% top-1 accuracy on ImageNet object classification (hence $20\times$ compression vs. the original model).

2 Related work

As there exists a large body of literature for network compression, we review the work closest to ours and refer the reader to the two recent surveys [18, 19] for a more comprehensive overview.

Low-precision training. Since early works like those of Courbariaux *et al.* [20, 21], researchers have developed various approaches to train networks with low precision weights. Those approaches include training with binary or ternary weights [22, 23, 24, 25], learning a combination of binary bases [26] and quantizing the activations [27, 28, 29]. Some of these methods assume the possibility to employ specialized hardware that speed up inference time and improve power efficiency by replacing most arithmetic operations with bit-wise operations. However, the back-propagation has to be adapted to the case where the weights are discrete using accumulation or projection techniques. Despite some noticeable progress in the context of binary networks, this particular choice of discretization suffers a significant drop in accuracy.

Quantization. Vector Quantization (VQ) and Product Quantization (PQ) have been extensively studied in the context of nearest-neighbor search [30, 31, 32]. The idea is to decompose the original high-dimensional space into a cartesian product of subspaces that are quantized separately (sometimes with a joint codebook). To our knowledge, Gong *et al.* [17] were the first to introduce these stronger quantizers in the context of neural network quantization. As we will see in the remainder of this paper, employing this discretization off-the-shelf does not optimize the right objective function, and leads to a catastrophic drift for deep networks.

Pruning. Network pruning amounts to removing connections according to an importance criteria (typically the magnitude of the weight associated with this connection) until the desired model size/accuracy tradeoff is reached [33]. A natural extension of this work is to prune structural components of the network, for instance by enforcing channel-level sparsity [34]. However, those methods alternate between pruning and re-training steps and thus typically require a long training time.

Architectures. Architectures such as SqueezeNet [35], NASNet [36], ShuffleNet [11, 37] or MobileNet [10, 38] typically rely on a combination of depth-wise and point-wise convolutional filters, sometimes along with channel splitting or shuffling. The architecture is either designed by hand or using the framework of architecture search [39]. Those architectures trade off a low memory footprint against a loss of accuracy. For instance, the respective model size and test top-1 accuracy of ImageNet of a Mobilenet are 13.4 MB for 71.9%, to be compared with a vanilla ResNet-50 with size 97.5 MB for a top-1 of 76.2%. Moreover, larger models such as ResNets can benefit from large-scale weakly- or semi-supervised learning to reach better performance [6, 2].

Combining some of the mentioned approaches yields high compression factors as demonstrated by Han *et al.* with Deep Compression (DC) [15] or more recently by Tung & Mori [40] where the authors successively prune and quantize various architectures. Moreover and from a practical point of view, the process of compressing networks depends on the type of hardware on which the networks will run. Recent work directly quantizes to optimize energy-efficiency and latency time on a specific hardware [41]. Finally, the memory overhead of storing the full activations is negligible compared to the storage of the weights for two reasons. First, in realistic real-time inference setups, the batch size is almost always equal to one. Second, a forward pass only requires to store the activations of the *current* layer –which are often smaller than the size of the input– and not the whole activations of the network.

3 Our approach

In this section, we describe our strategy for network compression. We first introduce our approach in the simple case of a fully-connected layer and then apply it to a convolutional layer. Finally, we show how to extend our approach to quantize a modern ConvNet architecture.

3.1 Layer quantization

In this section, we describe our layer quantization method in the case of a fully-connected layer and a convolutional layer. The specificity of our approach is that it aims at a small reconstruction error for the outputs of the layer, not the layer weights themselves.

3.1.1 Fully-connected layer

In this section, we consider a fully-connected layer with weights $\mathbf{W} \in \mathbf{R}^{C_{\text{in}} \times C_{\text{out}}}$ and, without loss of generality, we omit the bias since it does not impact reconstruction error.

Product Quantization (PQ). Applying the PQ algorithm to the columns of $\mathbf{W}$ consists in evenly splitting each column into m contiguous subvectors and learning a codebook on the resulting mC_{out} subvectors. Then, a column of $\mathbf{W}$ is quantized by mapping each of its subvector to its nearest codeword in the codebook. For simplicity, we assume that C_{in} is a multiple of m, *i.e.*, that all the subvectors have the same dimension $d = C_{\text{in}}/m$.

More formally, the codebook $\mathcal{C} = \{\mathbf{c}_1, \dots, \mathbf{c}_k\}$ contains k codewords of dimension d. Any column $\mathbf{w}_j$ of $\mathbf{W}$ is mapped to its quantized version $\mathbf{q}(\mathbf{w}_j) = (\mathbf{c}_{i_1}, \dots, \mathbf{c}_{i_m})$ where i_1 denotes the index of the codeword assigned to the first subvector of $\mathbf{w}_j$, and so forth. The codebook is then learned by minimizing the following objective function:

$$\|\mathbf{W} - \widehat{\mathbf{W}}\|_2^2 = \sum_j \|\mathbf{w}_j - \mathbf{q}(\mathbf{w}_j)\|_2^2, \tag{1}$$

where $\widehat{\mathbf{W}}$ denotes the quantized weights. This objective can be efficiently minimized with k-means. When m is set to 1, PQ is equivalent to vector quantization (VQ) and when m is equal to C_{in}, it is the scalar k-means algorithm. The main benefit of PQ is its expressivity: each column $\mathbf{w}_j$ is mapped to a vector in the product $\mathcal{C} = \mathcal{C} \times \cdots \times \mathcal{C}$, thus PQ generates an implicit codebook of size k^m.

Our algorithm. PQ quantizes the weight matrix of the fully-connected layer. However, in practice, we are interested in preserving the output of the layer, not its weights. This is illustrated in the case of a non-linear classifier in Figure 1: preserving the weights a layer does not necessarily guarantee preserving its output. In other words, the Frobenius approximation of the weights of a layer is not guaranteed to be the best approximation of the output over some arbitrary domain (in particular for *in-domain* inputs). We thus propose an alternative to PQ that directly minimizes the *reconstruction error* on the output activations obtained by applying the layer to in-domain inputs. More precisely, given a batch of B input activations $\mathbf{x} \in \mathbf{R}^{B\times C_{\text{in}}}$, we are interested in learning a codebook $\mathcal{C}$ that minimizes the difference between the output activations and their reconstructions:

$$\|\mathbf{y} - \widehat{\mathbf{y}}\|_2^2 = \sum_j \|\mathbf{x}(\mathbf{w}_j - \mathbf{q}(\mathbf{w}_j))\|_2^2, \tag{2}$$

where $\mathbf{y} = \mathbf{x}\mathbf{W}$ is the output and $\widehat{\mathbf{y}} = \mathbf{x}\widehat{\mathbf{W}}$ its reconstruction. Our objective is a re-weighting of the objective in Equation (1). We can thus learn our codebook with a weighted k-means algorithm. First, we unroll $\mathbf{x}$ of size $B \times C_{\text{in}}$ into $\widetilde{\mathbf{x}}$ of size $(B \times m) \times d$ *i.e.* we split each row of $\mathbf{x}$ into m subvectors of size d and stack these subvectors. Next, we adapt the EM algorithm as follows.

(1) **E-step (cluster assignment).** Recall that every column $\mathbf{w}_j$ is divided into m subvectors of dimension d. Each subvector $\mathbf{v}$ is assigned to the codeword $\mathbf{c}_j$ such that

$$\mathbf{c}_j = \operatorname*{argmin}_{\mathbf{c}\in\mathcal{C}} \|\widetilde{\mathbf{x}}(\mathbf{c} - \mathbf{v})\|_2^2. \tag{3}$$

This step is performed by exhaustive exploration. Our implementation relies on broadcasting to be computationally efficient.

(2) **M-step (codeword update).** Let us consider a codeword $\mathbf{c} \in \mathcal{C}$. We denote $(\mathbf{v}_p)_{p\in I_{\mathbf{c}}}$ the subvectors that are currently assigned to $\mathbf{c}$. Then, we update $\mathbf{c} \leftarrow \mathbf{c}^\star$, where

$$\mathbf{c}^\star = \operatorname*{argmin}_{\mathbf{c}\in\mathbf{R}^d} \sum_{p\in I_{\mathbf{c}}} \|\widetilde{\mathbf{x}}(\mathbf{c} - \mathbf{v}_p)\|_2^2. \tag{4}$$

This step is performed by explicitly computing the solution of the least-squares problem[2]. In our implementation, we perform the computation of the pseudo-inverse of $\widetilde{\mathbf{x}}$ before alternating between the Expectation and Minimization steps as it does not depend on the learned codebook $\mathcal{C}$.

We initialize the codebook $\mathcal{C}$ by uniformly sampling k vectors among those we wish to quantize. After performing the E-step, some clusters may be empty. To resolve this issue, we iteratively perform the following additional steps for each empty cluster of index i. (1) Find codeword $\mathbf{c_0}$ corresponding to the most populated cluster ; (2) define new codewords $\mathbf{c}_0' = \mathbf{c}_0 + \mathbf{e}$ and $\mathbf{c}_i' = \mathbf{c}_0 - \mathbf{e}$, where $\mathbf{e} \sim \mathcal{N}(\mathbf{0}, \varepsilon\mathbf{I})$ and (3) perform again the E-step. We proceed to the M-step after all the empty clusters are resolved. We set $\varepsilon = 1\text{e}{-8}$ and we observe that its generally takes less than 1 or 2 E-M iterations to resolve all the empty clusters. Note that the quality of the resulting compression is sensitive to the choice of $\mathbf{x}$ and we discuss its influence in Section 4.1.

3.1.2 Convolutional layers

Despite being presented in the case of a fully-connected layer, our approach works on any set of vectors. As a consequence, our apporoach can be applied to a convolutional layer if we split the associated 4D weight matrix into a set of vectors. There are many ways to split a 4D matrix in a set of vectors and we are aiming for one that maximizes the correlation between the vectors since vector quantization based methods work the best when the vectors are highly correlated.

Given a convolutional layer, we have C_{out} filters of size $K \times K \times C_{\text{in}}$, leading to an overall 4D weight matrix $\mathbf{W} \in \mathbf{R}^{C_{\text{out}}\times C_{\text{in}}\times K\times K}$. The dimensions along the output and input coordinate have no particular reason to be correlated. On the other hand, the spatial dimensions related to the filter size are by nature very correlated: nearby patches or pixels likely share information. As depicted in Figure 3.1.2, we thus reshape the weight matrix in a way that lead to spatially coherent quantization.

[2]Denoting $\widetilde{\mathbf{x}}^+$ the Moore-Penrose pseudoinverse of $\widetilde{\mathbf{x}}$, we obtain $\mathbf{c}^* = \frac{1}{|I_{\mathbf{c}}|}\widetilde{\mathbf{x}}^+\widetilde{\mathbf{x}}\left(\sum_{p\in I_{\mathbf{c}}} \mathbf{v}_p\right)$

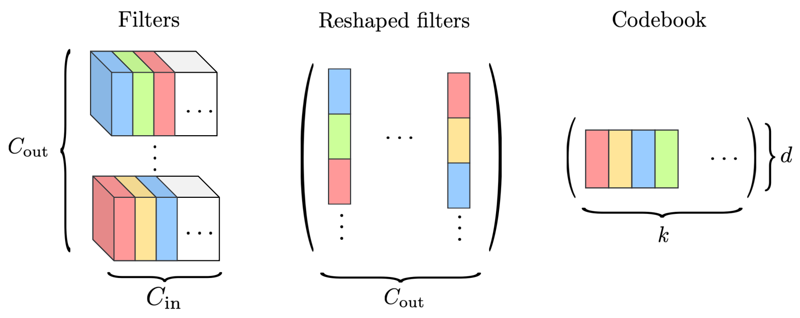

Figure 2: We quantize C_{out} filters of size $C_{\text{in}} \times K \times K$ using a subvector size of $d = K \times K$. In other words, we spatially quantize the convolutional filters to take advantage of the redundancy of information in the network. Similar colors denote subvectors assigned to the same codewords.

More precisely, we quantize $\mathbf{W}$ spatially into subvectors of size $d = K \times K$ using the following procedure. We first reshape $\mathbf{W}$ into a 2D matrix of size $(C_{\text{in}} \times K \times K) \times C_{\text{out}}$. Column j of the reshaped matrix $\mathbf{W}_{\text{r}}$ corresponds to the j^{th} filter of $\mathbf{W}$ and is divided into C_{in} subvectors of size $K \times K$. Similarly, we reshape the input activations $\mathbf{x}$ accordingly to $\mathbf{x}_{\text{r}}$ so that reshaping back the matrix $\mathbf{x}_{\text{r}}\mathbf{W}_{\text{r}}$ yields the same result as $\mathbf{x} * \mathbf{W}$. In other words, we adopt a dual approach to the one using bi-level Toeplitz matrices to represent the weights. Then, we apply our method exposed in Section 3.1 to quantize each column of $\mathbf{W}_{\text{r}}$ into $m = C_{\text{in}}$ subvectors of size $d = K \times K$ with k codewords, using $\mathbf{x}_{\text{r}}$ as input activations in (2). As a natural extension, we also quantize with larger subvectors, for example subvectors of size $d = 2 \times K \times K$, see Section 4 for details.

In our implementation, we adapt the reshaping of $\mathbf{W}$ and $\mathbf{x}$ to various types of convolutions. We account for the padding, the stride, the number of groups (for depthwise convolutions and in particular for pointwise convolutions) and the kernel size. We refer the reader to the code for more details.

3.2 Network quantization

In this section, we describe our approach for quantizing a neural network. First, we quantize the network sequentially starting from the lowest layer to the highest layer. Then, in a final step, we globally finetune the codebooks of all the layers to reduce any residual drifts and we update the running statistics of the BatchNorm layers.

3.2.1 Bottom-up quantization

We explain below the quantization procedure in which we guide the compression of the student network by the non-compressed teacher network.

Learning the codebook. We recover the *current* input activations of the layer, *i.e.* the input activations obtained by forwarding a batch of images through the *quantized* lower layers, and we quantize the current layer using those activations. Experimentally, we observed a drift in both the reconstruction and classification errors when using the activations of the non-compressed network rather than the current activations.

Finetuning the codebook. We finetune the codewords by distillation [16] using the non-compressed network as the teacher network and the compressed network (up to the current layer) as the student network. Denoting y_{t} (resp. y_{s}) the output probabilities of the teacher (resp. student) network, the loss we optimize is the Kullback-Leibler divergence $\mathcal{L} = \text{KL}(\mathbf{y}_{\text{s}}, \mathbf{y}_{\text{t}})$. Finetuning on codewords is done by averaging the gradients of each subvector assigned to a given codeword. More formally, after the quantization step, we fix the assignments once for all. Then, denoting $(\mathbf{b}_p)_{p \in I_{\mathbf{c}}}$ the subvectors that are assigned to codeword $\mathbf{c}$, we perform the SGD update with a learning rate η

$$\mathbf{c} \leftarrow \mathbf{c} - \eta \frac{1}{|I_{\mathbf{c}}|} \sum_{p \in I_{\mathbf{c}}} \frac{\partial \mathcal{L}}{\partial \mathbf{b}_p}. \tag{5}$$

Experimentally, we find the approach to perform better than finetuning on the target of the images as demonstrated in Table 3. Moreover, this approach does not require any labelled data.

3.2.2 Global finetuning

We empirically find it beneficial to finetune *all* the centroids after the whole network is quantized. The finetuning procedure is exactly the same as described in Secction 3.2.1, except that we additionally switch the BatchNorms to the training mode, meaning that the learnt coefficients are still fixed but that the batch statistics (running mean and variance) are still being updated with the standard moving average procedure.

We perform the global finetuning using the standard ImageNet training set for 9 epochs with an initial learning rate of 0.01, a weight decay of 10^{-4} and a momentum of 0.9. The learning rate is decayed by a factor 10 every 3 epochs. As demonstrated in the ablation study in Table 3, finetuning on the true labels performs worse than finetuning by distillation. A possible explanation is that the supervision signal coming from the teacher network is richer than the one-hot vector used as a traditional learning signal in supervised learning [16].

4 Experiments

4.1 Experimental setup

We quantize vanilla ResNet-18 and ResNet-50 architectures pretrained on the ImageNet dataset [42]. Unless explicit mention of the contrary, the pretrained models are taken from the PyTorch model zoo[3]. We run our method on a 16 GB Volta V100 GPU. Quantizing a ResNet-50 with our method (including all finetuning steps) takes about one day on 1 GPU. We detail our experimental setup below. Our code and the compressed models are open-sourced.

Compression regimes. We explore a *large block sizes* (resp. *small block sizes*) compression regime by setting the subvector size of regular 3×3 convolutions to $d = 9$ (resp. $d = 18$) and the subvector size of pointwise convolutions to $d = 4$ (resp. $d = 8$). For ResNet-18, the block size of pointwise convolutions is always equal to 4. The number of codewords or centroids is set to $k \in \{256, 512, 1024, 2048\}$ for each compression regime. Note that we clamp the number of centroids to $\min(k, C_{\text{out}} \times m/4)$ for stability. For instance, the first layer of the first stage of the ResNet-50 has size 64× 64× 1 ×1, thus we always use $k = 128$ centroids with a block size $d = 8$. For a given number of centroids k, small blocks lead to a lower compression ratio than large blocks.

Sampling the input activations. Before quantizing each layer, we randomly sample a batch of 1024 training images to obtain the input activations of the current layer and reshape it as described in Section 3.1.2. Then, before each iteration (E+M step) of our method, we randomly sample $10,000$ rows from those reshaped input activations.

Hyperparameters. We quantize each layer while performing 100 steps of our method (sufficient for convergence in practice). We finetune the centroids of each layer on the standard ImageNet training set during 2,500 iterations with a batch size of 128 (resp 64) for the ResNet-18 (resp. ResNet-50) with a learning rate of 0.01, a weight decay of 10^{-4} and a momentum of 0.9. For accuracy and memory reasons, the classifier is always quantized with a block size $d = 4$ and $k = 2048$ (resp. $k = 1024$) centroids for the ResNet-18 (resp. ResNet-50). Moreover, the first convolutional layer of size 7×7 is not quantized, as it represents less than 0.1% (resp. 0.05%) of the weights of a ResNet-18 (resp. ResNet-50).

Metrics. We focus on the tradeoff between two metrics, namely accuracy and memory. The accuracy is the top-1 error on the standard validation set of ImageNet. The memory footprint is calculated as the indexing cost (number of bits per weight) plus the overhead of storing the centroids in float16. As an example, quantizing a layer of size $128 \times 128 \times 3 \times 3$ with $k = 256$ centroids (1 byte per subvector) and a block size of $d = 9$ leads to an indexing cost of 16 KB for $m = 16,384$ blocks plus the cost of storing the centroids of 4.5 KB.

[3]https://pytorch.org/docs/stable/torchvision/models

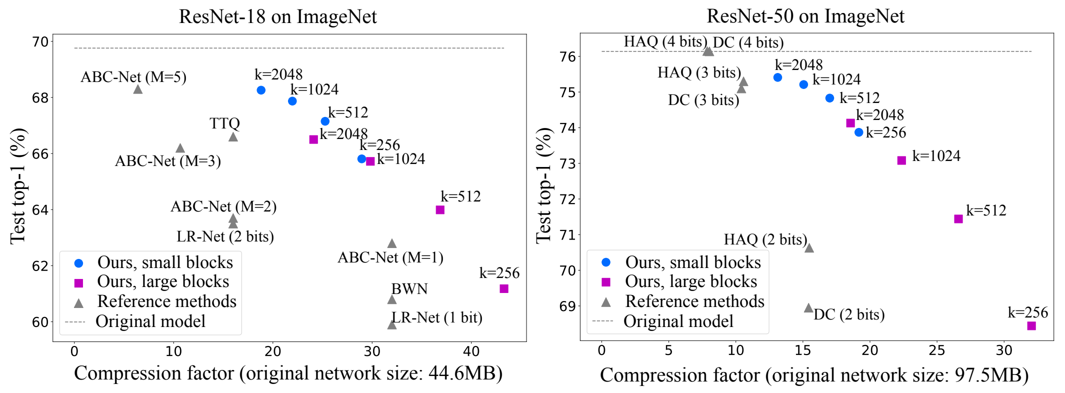

Figure 3: Compression results for ResNet-18 and ResNet-50 architectures. We explore two compression regimes as defined in Section 4.1: small block sizes (block sizes of $d = 4$ and 9) and large block sizes (block sizes $d = 8$ and 18). The results of our method for $k = 256$ centroids are of practical interest as they correspond to a byte-compatible compression scheme.

Table 1: Results for vanilla ResNet-18 and ResNet-50 architectures for $k = 256$ centroids.

Model (original top-1)	Compression	Size ratio	Model size	Top-1 (%)
ResNet-18 (69.76%)	Small blocks	29x	1.54 MB	**65.81** $\pm$0.04
	Large blocks	43x	1.03 MB	**61.10** $\pm$0.03
ResNet-50 (76.15%)	Small blocks	19x	5.09 MB	**73.79** $\pm$0.05
	Large blocks	31x	3.19 MB	**68.21** $\pm$0.04

4.2 Image classification results

We report below the results of our method applied to various ResNet models. First, we compare our method with the state of the art on the standard ResNet-18 and ResNet-50 architecture. Next, we show the potential of our approach on a competitive ResNet-50. Finally, an ablation study validates the pertinence of our method.

Vanilla ResNet-18 and ResNet-50. We evaluate our method on the ImageNet benchmark for ResNet-18 and ResNet-50 architectures and compare our results to the following methods: Trained Ternary Quantization (TTQ) [23], LR-Net [22], ABC-Net [26], Binary Weight Network (XNOR-Net or BWN) [25], Deep Compression (DC) [15] and Hardware-Aware Automated Quantization (HAQ) [41]. We report the accuracies and compression factors in the original papers and/or in the two surveys [18, 19] for a given architecture when the result is available. We do not compare our method to DoReFa-Net [27] and WRPN [29] as those approaches also use low-precision activations and hence get lower accuracies, e.g., 51.2% top-1 accuracy for a XNOR-Net with ResNet-18. The results are presented in Figure 1. For better readability, some results for our method are also displayed in Table 1. We report the average accuracy and standard deviation over 3 runs. Our method significantly outperforms state of the art papers for various operating points. For instance, for a ResNet-18, our method with large blocks and $k = 512$ centroids reaches a larger accuracy than ABC-Net ($M = 2$) with a compression ratio that is 2x larger. Similarly, on the ResNet-50, our compressed model with $k = 256$ centroids in the large blocks setup yields a comparable accuracy to DC (2 bits) with a compression ratio that is 2x larger.

The work by Tung & Mori [40] is likely the only one that remains competitive with ours with a 6.8 MB network after compression, with a technique that prunes the network and therefore implicitly changes the architecture. The authors report the delta accuracy for which we have no direct comparable top-1 accuracy, but their method is arguably complementary to ours.

Semi-supervised ResNet-50. Recent works [6, 2] have demonstrated the possibility to leverage a large collection of unlabelled images to improve the performance of a given architecture. In particular, Yalniz *et al.* [2] use the publicly available YFCC-100M dataset [43] to train a ResNet-50 that reaches 79.1% top-1 accuracy on the standard validation set of ImageNet. In the following,

Table 2: Best test top-1 accuracy on ImageNet for a given size budget (no architecture constraint).

Size budget	Best previous published method	Ours
$\sim$1 MB	**70.90%** (HAQ [41], MobileNet v2)	64.01% (vanilla ResNet-18)
$\sim$5 MB	71.74% (HAQ [41], MobileNet v1)	**76.12%** (semi-sup. ResNet-50)
$\sim$10 MB	75.30% (HAQ [41], ResNet-50)	**77.85%** (semi-sup. ResNet-50)

Table 3: Ablation study on ResNet-18 (test top-1 accuracy on ImageNet).

Compression	Centroids k	No act + Distill	Act + Labels	**Act + Distill (ours)**
Small blocks	256	64.76	65.55	**65.81**
	512	66.31	66.82	**67.15**
	1024	67.28	67.53	**67.87**
	2048	67.88	67.99	**68.26**
Large blocks	256	60.46	61.01	**61.18**
	512	63.21	63.67	**63.99**
	1024	64.74	65.48	**65.72**
	2048	65.94	66.21	**66.50**

we use this particular model and refer to it as semi-supervised ResNet-50. In the low compression regime (block sizes of 4 and 9), with $k = 256$ centroids (practical for implementation), our compressed semi-supervised ResNet-50 reaches **76.12% top-1 accuracy**. In other words, the compressed model attains the performance of a vanilla, non-compressed ResNet50 while having a size of 5 MB (vs. 97.5MB for the non-compressed ResNet-50).

Comparison for a given size budget. To ensure a fair comparison, we compare our method for a given model size budget against the reference methods in Table 2. It should be noted that our method can further benefit from advances in semi-supervised learning to boosts the performance of the non-compressed and hence of the compressed network.

Ablation study. We perform an ablation study on the vanilla ResNet-18 to study the respective effects of quantizing using the activations and finetuning by distillation (here, finetuning refers both to the per-layer finetuning described in Section 3 and to the global finetuning after the quantization described in Section 4.1). We refer to our method as Act + Distill. First, we still finetune by distillation but change the quantization: instead of quantizing using our method (see Equation (2)), we quantizing using the standard PQ algorithm and do not take the activations into account, see Equation (1). We refer to this method as No act + Distill. Second, we quantize using our method but perform a standard finetuning using the image labels (Act + Labels). The results are displayed in Table 3. Our approach consistently yields significantly better results. As a side note, quantizing all the layers of a ResNet-18 with the standard PQ algorithm and without any finetuning leads to top-1 accuracies below 25% for all operating points, which illustrates the drift in accuracy occurring when compressing deep networks with standard methods (as opposed to our method).

4.3 Image detection results

To demonstrate the generality of our method, we compress the Mask R-CNN architecture used for image detection in many real-life applications [8]. We compress the backbone (ResNet-50 FPN) in the *small blocks* compression regime and refer the reader to the open-sourced compressed model for the block sizes used in the various heads of the network. We use $k = 256$ centroids for every layer. We perform the fine-tuning (layer-wise and global) using distributed training on 8 V100 GPUs. Results are displayed in Table 4. We argue that this provides an interesting point of comparison for future work aiming at compressing such architectures for various applications.

Table 4: Compression results for Mask R-CNN (backbone ResNet-50 FPN) for $k = 256$ centroids (compression factor $26\times$).

Model	Size	Box AP	Mask AP
Non-compressed	170 MB	37.9	34.6
Compressed	6.51 MB	33.9	30.8

5 Conclusion

We presented a quantization method based on Product Quantization that gives state of the art results on ResNet architectures and that generalizes to other architectures such as Mask R-CNN. Our compression scheme does not require labeled data and the resulting models are byte-aligned, allowing for efficient inference on CPU.

Further research directions include testing our method on a wider variety of architectures. In particular, our method can be readily adapted to simultaneously compress and transfer ResNets trained on ImageNet to other domains. Finally, we plan to take the non-linearity into account to improve our reconstruction error.

Acknowledgements

We thank Fransisco Massa and Alexandre Défossez for their advice regarding code optimization.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, 2015.

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

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 2012.

[4] Saining Xie, Ross Girshick, Piotr Dollar, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Conference on Computer Vision and Pattern Recognition*, 2017.

[5] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. *Conference on Computer Vision and Pattern Recognition*, 2017.

[6] Dhruv Mahajan, Ross B. Girshick, Vignesh Ramanathan, Kaiming He, Manohar Paluri, Yixuan Li, Ashwin Bharambe, and Laurens van der Maaten. Exploring the limits of weakly supervised pretraining. *CoRR*, 2018.

[7] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. Mixmatch: A holistic approach to semi-supervised learning. *arXiv preprint arXiv:1905.02249*, 2019.

[8] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. *International Conference on Computer Vision (ICCV)*, 2017.

[9] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: hardware-aware automated quantization. *arXiv preprint arXiv:1811.08886*, 2018.

[10] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, 2017.

[11] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, 2017.

[12] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[13] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Conference on Computer Vision and Pattern Recognition*, 2018.

[14] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product Quantization for Nearest Neighbor Search. IEEE *Transactions on Pattern Analysis and Machine Intelligence*, 2011.

[15] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations*, 2016.

[16] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *NIPS Deep Learning Workshop*, 2014.

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

[18] Yunhui Guo. A survey on methods and theories of quantized neural networks. *CoRR*, 2018.

[19] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *CoRR*, 2017.

[20] Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, 2016.

[21] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *CoRR*, 2015.

[22] Oran Shayer, Dan Levi, and Ethan Fetaya. Learning discrete weights using the local reparameterization trick. *CoRR*, 2017.

[23] Chenzhuo Zhu, Song Han, Huizi Mao, and William J. Dally. Trained ternary quantization. *CoRR*, 2016.

[24] Fengfu Li and Bin Liu. Ternary weight networks. *CoRR*, 2016.

[25] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, 2016.

[26] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. *CoRR*, 2017.

[27] 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. *CoRR*, 2016.

[28] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *CoRR*, 2017.

[29] Asit K. Mishra, Eriko Nurvitadhi, Jeffrey J. Cook, and Debbie Marr. WRPN: wide reduced-precision networks. *CoRR*, 2017.

[30] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. IEEE *Transactions on Pattern Analysis and Machine Intelligence*, 2011.

[31] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2014.

[32] Mohammad Norouzi and David J Fleet. Cartesian k-means. In *Conference on Computer Vision and Pattern Recognition*, 2013.

[33] Yann Le Cun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, 1990.

[34] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. *International Conference on Computer Vision*, 2017.

[35] Forrest Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size. *CoRR*, 2016.

[36] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, 2017.

[37] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet V2: practical guidelines for efficient CNN architecture design. *CoRR*, 2018.

[38] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, 2018.

[39] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *arXiv e-prints*, 2019.

[40] Frederick Tung and Greg Mori. Deep neural network compression by in-parallel pruning-quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

[41] Kuan Wang, Zhijian Liu, Yujun Lin andx Ji Lin, and Song Han. HAQ: hardware-aware automated quantization. *CoRR*, 2018.

[42] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Conference on Computer Vision and Pattern Recognition*, 2009.

[43] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. The new data and new challenges in multimedia research. *CoRR*, 2015.