Hyperspherical Quantization: Toward Smaller and More Accurate Models

Dan Liu\(^1\), Xi Chen\(^1\), Chen Ma\(^2\), Xue Liu\(^1\)
\(^1\)McGill University, \(^2\)City University of Hong Kong
dan.liu4@mail.mcgill.ca, 
\{xi.chen11,xue.liu\}@mcgill.ca, 
chenma@cityu.edu.hk

Abstract

Model quantization enables the deployment of deep neural networks under resource-constrained devices. Vector quantization aims at reducing the model size by indexing model weights with full-precision embeddings, i.e., code-words, while the index needs to be restored to 32-bit during computation. Binary and other low-precision quantization methods can reduce the model size up to \(32 \times\), however, at the cost of a considerable accuracy drop. In this paper, we propose an efficient framework for ternary quantization to produce smaller and more accurate compressed models. By integrating hyperspherical learning, pruning and reinitialization, our proposed Hyperspherical Quantization (HQ) method reduces the cosine distance between the full-precision and ternary weights, thus reducing the bias of the straight-through gradient estimator during ternary quantization. Compared with existing work at similar compression levels (\(\sim 30 \times, \sim 40 \times\)), our method significantly improves the test accuracy and reduces the model size.

1. Introduction

Despite promising results in real-world applications, deep neural network (DNN) models usually contain a large number of parameters, making them impossible to deploy on edge devices. A significant amount of research has been made to reduce the size and computational overhead of DNN models through quantization and pruning. Pruning brings high sparsity, but cannot take advantage of compression and acceleration without customized hardware [25]. Cluster-based quantization, such as vector quantization and product quantization, remarkably reduces the model disk footprint [67, 53, 7], but its memory footprint is larger than that of the low-precision quantization method [12, 60, 10, 83, 37], as the actual weight values involved in computation remain full-precision [12]. Ultra-low-precision quantization, e.g., binary [33, 11, 10], ternary [37, 83], and 2-bit quantization [82, 8, 18], has fast inference and low memory footprint [60], but it usually leads to a significant accuracy drop, due to the inaccurate weights [22] and gradients [78].

Gradually discretizing the weights can overcome such non-differentiability [52, 34, 9], i.e., reducing the discrepancy between the quantized weights in the forward pass and the full-precision weights in the backward pass. However, it only performs well with 4-bit (or higher) precision as the ultra-low bit quantizer may seriously damage the weight magnitude leading to unstable weights [22]. Intuitively, ternary quantization barely affects the sign of the weights, making the direction of weight vectors [62] changes relatively more stable than their magnitude. Recently, many studies [48, 49, 47, 15, 13, 59, 5] show that the angular information [45] preserves the key semantics in feature maps.

We propose hyperspherical quantization (HQ), a method combining pruning and reinitialization [20, 81] to produce accurate ternary DNN models with a smaller memory/disk footprint. We first pre-train a DNN model with a hyperspherical learning method [49] to preserve the direction information of the model weights, then apply our proposed approach to push the full-precision weights close to their ternary counterparts, and lastly, we combine the straight-through estimator (STE) [2] with a gradually increased threshold to fulfill the ternary quantization process. Our main contributions are summarized as follows:

- We demonstrate that simply integrating pruning and reinitialization can significantly reduce the impact of weight discrepancy caused by the ternary quantizer. We unify the pruning and quantization thresholds to one to further optimize the quantization process.

- Our method significantly outperforms existing works in terms of the size-accuracy trade-off of DNN models. For example, on ImageNet, our method can compress a ResNet-18 model from 45 MB to 939 KB (48\(\times\) compressed) while the accuracy is only 4% lower than the original accuracy. It is the best result among the existing results (43\(\times\), 6.4% accuracy drop).
2. Related Work

2.1. Hyperspherical Learning

Hyperspherical learning aims to study the impact of the direction [62] of weight vectors on DNN models. [62] discovers that detaching the model weight direction information from its magnitude can accelerate training. [49] shows that the direction information of weight vectors, in contrast to weight magnitude, preserves useful semantic meanings in feature maps. [47, 15, 72] propose to apply regularization to angular representations on a hypersphere to enhance the model generalization ability in face recognition tasks. [46, 47, 15, 39, 5, 44] further study the empirical generalization ability of hyperspherical learning.

2.2. Quantization

Low-bit quantization methods convert float values of weights and activations to lower bit values [10, 11, 60, 54]. These methods make it possible to substantially reduce the computational cost during CPU inference [71]. For example, binary quantization [10, 11] compresses full-precision weights into a 1-bit representation, thus significantly reducing the memory footprint by 32×.

Clustering-based weight quantization, such as product quantization [21] and vector quantization [23, 4, 67] focus on optimizing the size-accuracy trade-off and can significantly compress the disk footprint by grouping weight values to a codebook. Common approaches cluster weights through k-means [67, 75, 66, 23] and further finetune the clustering center by minimizing the reconstruction error in the network [19, 67, 75]. The compression ratio and accuracy trade-off can be adjusted by changing the number of groups. [26] applies k-means-based vector quantization with pruning and Huffman coding to further reduce the model size. However, the codebook usually consists of 32-bit float numbers [67, 53], the memory footprint during computation is uncompresseed.

Some mixed-precision quantization methods overcome the shortcomings of low-bit and clustering based methods by means of reinforcement learning [73], integer programming [3], and differentiable neural architecture search [74], so as to apply different bit-widths in model weights to optimize inference time and model size. However, mixed-precision still cannot effectively compress the model size due to the use of 8 to 32-bit weights. Other mixed-precision quantization methods assign different bit widths to layer weights according to various measures, including hardware [73, 77] and second-order information [17, 65].

2.3. Pruning

Pruning consists of structured and unstructured methods. It can greatly compress redundancy and maintain high accuracy. Unstructured pruning brings high sparsity, but cannot take advantage of acceleration without customized hardware [25]. Only structured pruning methods can reduce the inference latency and are easier to accelerate [30, 38] because the original weight structures of the model are preserved. Unstructured pruning uses criteria, such as gradient [56, 35], and magnitude [26, 55] information, to remove individual weights; structured pruning [38, 32, 1] aims to remove unimportant channels of the convolutional layer based on similar criteria. The lottery ticket hypothesis [20] shows that there exists sparse subnetworks that can be trained from scratch and achieve the same performance as the full network. [81] studies the lottery ticket hypothesis from the perspective of weight initialization and points out that the key premise is the sign of weight values.

Re-training after pruning [20, 81] reveals the link between the network structure and performance. Furthermore, our findings show that training after pruning and reinitialization can be used to produce more accurate and highly compressed ternary weights, which surpasses the current model compression methods and has a wide range of application scenarios.

3. Preliminary and Notations

3.1. Hyperspherical Model

A general representation of a hyperspherical neural network layer [49] is:

\[
y = \phi(W^T x),
\]

where \( W \in \mathbb{R}^{n \times m} \) is the weight matrix, \( x \in \mathbb{R}^n \) is the input vector to the layer, \( \phi \) represents a nonlinear activation function, and \( y \in \mathbb{R}^m \) is the output feature vector. The input vector \( x \) and each column vector \( w_j \in \mathbb{R}^n \) of \( W \) satisfy \( \|w_j\|_2 = 1, \|x\|_2 = 1 \) for all \( j = 1, \ldots, m \).

3.2. Ternary Quantizer

In this work, the ternary quantizer is:

\[
W = \text{Ternary}(W, \Delta) = \begin{cases} 
\frac{1}{\sqrt{|I_\Delta|}} & : w_{ij} > \Delta, \\
0 & : |w_{ij}| \leq \Delta, \\
-\frac{1}{\sqrt{|I_\Delta|}} & : w_{ij} < -\Delta,
\end{cases}
\]

where \( W \) is the full-precision weights, \( \Delta \) is a threshold, \( I_\Delta = \{i|w_{ij}| > \Delta\} \) [37], and \( |I_\Delta| \) denotes total non-zero values in the \( j \)-th column vector \( w_j \). With \( \Delta = 0 \), \text{Ternary}(\cdot) \) becomes a variant of \text{Binary}(\cdot) \) operation. And \( \phi(\tilde{w}_j^T x) = \phi\left(\frac{1}{\sqrt{|I_\Delta|}}\tilde{w}_j^T x\right) \) s.t. \( \tilde{w} \in \{-1, 0, 1\} \).

3.3. Pruning

The unstructured pruning [26] is defined by:

\[
W' = \text{Prune}(W, r) = W \odot M,
\]
where \( \odot \) denotes the element-wise multiplication. Mask \( M \) selects the top \( r \) percent of the smaller weights in \( W \):

\[
M = \text{Mask}(W, r) = \text{Sign}([\text{Ternary}(W, \Delta)]),
\]

where \( \Delta = \text{threshold}(W, r) \), and \( 0 \leq r < 1 \). The \( \text{threshold}(W, r) \) returns the corresponding minimum value of matrix \( W \) based on the percentage \( r \). We use pruning to represent unstructured pruning for simplicity.

### 3.4. Cosine Similarity \( S \) on HyperSphere

Based on Eq. (2) and hyperspherical learning, we have \( \|\hat{w}_j\|_2 = 1 \) and \( \|w_j\|_2 = 1 \). The vector-wise cosine similarity between \( w_j \) and \( \hat{w}_j \) is:

\[
S(\hat{w}_j, w_j) = \frac{\hat{w}_j \cdot w_j}{\|\hat{w}_j\|_2 \|w_j\|_2} = \sum_{i=1}^{n} \frac{1}{\sqrt{|I_\Delta|}}|w_{ij}|. \tag{5}
\]

If without pruning, \( W \) and \( \hat{W} \) will not contain 0. With \( \Delta = 0 \), we have \( |I_\Delta| = n \) and the cosine similarity between \( W \) and \( \hat{W} \) becomes:

\[
S(\hat{W}, W) = \frac{1}{m} \sum_{j=1}^{m} \sum_{i=1}^{n} \frac{1}{\sqrt{|I_\Delta|}}|w_{ij}|. \tag{6}
\]

After applying pruning (Eq. (3)) to \( W \):

\[
S(\hat{W}', W') = \frac{1}{m} \sum_{j=1}^{m} \sum_{i=1}^{n} \frac{1}{\sqrt{|I_\Delta|}}|w_{ij}'|. \tag{7}
\]

The cosine distance between full-precision and ternary weights is:

\[
D = 1 - \frac{1}{l} \sum_{k=1}^{l} S(\hat{W}_k, W_k), \tag{8}
\]

where \( l \) denotes the number of quantized layers.

### 4. Hyperspherical Quantization

In this section, we propose using pruning to increase the cosine similarity \( S \) between the full-precision weights \( W \) and the ternary weights \( \hat{W} \). We show how we can effectively quantize such discrepancy reduced model weights.

Our proposed method includes the preprocessing and quantization steps (Fig. 1). In the preprocessing step, we use iterative pruning with gradually increasing sparsity and reinitialization [81] to push \( W \) close to its ternary counterpart \( \hat{W} \). In the quantization step, as \( W \) is close to \( \hat{W} \) after the first step, it is easy to obtain a more accurate \( \hat{W} \) by using regular STE-based ternary quantization methods. We unify the thresholds of pruning and quantization as one single threshold during ternary quantization process.

#### 4.1. Increasing \( S \) by Preprocessing

We show that pruning on hypersphere can increase the cosine similarity \( S(\hat{W}, W) \), thus pushing the full-precision weight close to its ternary version. But the decayed weights during training cause unstable \( S \), which makes ternary quantization infeasible. Then the reinitialization is proposed to stabilize \( S \).

#### 4.1.1 Cosine Similarity and Hyperspherical Pruning

Given a full-precision \( W \) and its ternary form \( \hat{W} \), we seek to optimize the following problem under hyperspherical learning settings:

\[
\max_r S(\hat{W}', W') \tag{9}
\]

\[
s.t. \ 0 < S \leq 1,
\]

where \( W' = \text{Prune}(W, r) \) and \( \hat{W}' = \text{Ternary}(\hat{W}', 0) \). Obviously, if \( \|W'\|_0 = 1 \) then \( S(\hat{W}', W') = 1 \), but it is meaningless. Although there is no explicit solution for \( r \), we can increase \( S \) by gradually increasing \( r \). Since the model is trained with hyperspherical learning, based on Eq. (6)-(7), with pruning ratio \( r \), we have \( \frac{1}{\sqrt{|I_\Delta|}} \geq \frac{1}{\sqrt{n}} \) and:

\[
|w_{ij}'| = \frac{|w_{ij}|}{\|w'\|_2} \geq \frac{|w_{ij}|}{\|w\|_2} = |w_{ij}|, \tag{10}
\]

where \( \|w'\|_2 \leq \|w\|_2 = 1 \). Therefore:

\[
S(\hat{W}', W') \geq S(\hat{W}, W) \tag{11}
\]
4.2 Quantization with a Unified Threshold

With $\hat{W}$ close to $\hat{W}$ in the preprocessing step, we still need to perform weight quantization and pruning to further increase $S(\hat{W}, \hat{W})$. We introduce a gradually increasing quantization threshold $\Delta$ (Eq. 2) to unify pruning and quantization, as $\Delta$ can be seen as a pruning threshold. The non-differentiability of Ternary($) is bypassed with STE [2]:

Forward:
$$W = \text{Ternary}(W, \Delta)$$

(12)

Backward:
$$\frac{\partial E}{\partial \hat{W}} = \frac{\partial E}{\partial \hat{W}} \frac{\partial \hat{W}}{\partial W} \approx \frac{\partial E}{\partial W}.$$ (13)

STE essentially ignores the quantization operation and approximates it with an identity function.

The threshold $\Delta$ should gradually increase along with the training error, and such increase should slow down after the model converges (Fig. 3). Therefore, we directly use the averaged gradients to update the pruning threshold:

$$\Delta = \sum_{i=1}^{n} \frac{\partial L}{\partial w_i}, w_i \in w \text{ and } w_i \neq 0.$$ (14)

The gradient of $\frac{\partial L}{\partial w_i}$, where $w_i = 0$ is ignored. The training error is very large at the beginning, leading to a rapid increase of the threshold. As the model gradually converges, the training error will decrease and the threshold growth slows down (Fig. 3). The model sparsity is gradually increasing along with the learned threshold. The accuracy starts to decrease if the quantization continues after the model converges.

4.3 Implementation Details

4.3.1 Training Algorithm

Our proposed method is shown in Algorithm 1. We prune the model with a ratio from $r_l = 0.3$ to $r_h = 0.7$ based on [50, 83]. The overall process can be summarized as: i) Pre-training with hyperspherical learning architecture [49]; ii) Iterative preprocessing the model weights, i.e., prune the model to target sparsity $r_h$, and reset the weights via $W = \text{Ternary}(W, 0)$ after each pruning (Fig. 1); iii) Ternary quantization, updating the weights and $\Delta$ through STE.
Algorithm 1 HQ training approach

1. Input: Input x, a hyperspherical neural network
   \( \phi(W, \cdot), r_l = 0.3, r_h = 0.7 \), and step size \( \delta = 0.01 \).
2. Result: Quantized ternary network for inference
3. 1. Preprocessing:
   4. \( r = r_l \)
   5. while \( r < r_h \) do \( \triangleright \) Iterative pruning and reinitializing
   6. \( M = \text{Mask}(W, r) \) \( \triangleright \) Obtain the pruning mask
   7. \( W = W \odot M \) \( \triangleright \) Pruning
   8. \( W = \text{Ternary}(W, 0) \) \( \triangleright \) Reinitialization
   9. while not converged do
   10. \( y = \phi((M \odot \hat{W}), x) \) \( \triangleright \) Eq. (1)
   11. Perform SGD, calculate \( \frac{\partial L}{\partial \hat{W}} \), and update \( \hat{W} \)
   12. end while
   13. \( r = r + \delta \) \( \triangleright \) Increase the pruning ratio \( r \)
   14. end while
5. 2. Ternary Quantization:
   16. while not converged do
   17. \( \hat{W} = \text{Ternary}(W, \Delta) \)
   18. \( y = \phi((M \odot \hat{W}), x) \)
   19. Get \( \frac{\partial L}{\partial \hat{W}} \) via SGD; update \( W, \Delta \) \( \triangleright \) Eq. (13,14)
   20. end while

4.3.2 Training Settings and Training Time

For image classification, the batch size is 128. The weight decay is 0.0001, and the momentum of stochastic gradient descent (SGD) is 0.9. We use the cosine annealing schedule with restarts every 10 epochs [51] to adjust the learning rates. The initial learning rate is 0.001. All of the experiments use 16-bit half-precision from PyTorch to accelerate the training process. Thus, all parameters (except codebook with 2-bit) are stored as 16-bit precision values.

It takes about 50 epochs to convert a Pytorch model to a hyperspherical one. The SGD loop (Line 5-14 in Algorithm 1) takes at least 200 epochs. The ternary quantization loop (Line 16-20 in Algorithm 1) takes about 200 epochs.

4.3.3 Compression Strategy

Inspired by vector quantization, we use codebook as the compression method [67, 53]. Unlike the vector quantization methods that use a learned codebook, we use the Huffman table [70] as a fixed codebook (Table 2 in the Appendix). We use gzip to finalize the model file as in other work [69, 67]. It is shown that the Huffman table can maximize the compression effect when each codeword consists of three ternary values, e.g., \( \{0, 0, 0\} \). Note that the Huffman table can only boost the compression when the high-frequency patterns exist, such as \( \{0, 0, 0\} \). For other works with low-sparsity or non-sparse quantization [67, 7, 53, 83], applying the Huffman table may not help compress the model. We find inconsistent compression results in ABGD [67] and PQF [53]. The actual size of their models (obtained from github in gzip format) is shown in Table 2 is different from the results stated in their paper.

5. Experiments

Our experiments involve image classification and object detection tasks. We evaluate our method on ImageNet data set [61] with MobileNetV2 [63] and ResNet-18/50 [29]. For object detection, we use the MS COCO [42] dataset and Mask R-CNN [28, 76]. The pre-trained weights are provided by the PyTorch zoo and Detectron2 [76].

5.1. Image Classification

Following the practices of mainstream model compression work [67, 17, 53], when compressing the model size, we quantize all of the weights of the convolution and the fully-connected (FC) layers to 2-bit (except the first layer). We compare our results with leading compression results from PQF [53], ABGD [67], BRECQ [41], HAWQ [17], and TQNE [19]. We also compare our work with milestone approaches, such as ABC-Net [43], Deep Compression (DC) [26], Hardware-Aware Automated Quantization (HAQ) [73], Hessian AWare Quantization (HAWQ) [17], LR-Net [64], and BWN [60].

Our method significantly outperforms leading model compression methods in terms of bit-width, compression...
Figure 4: The compression ratio and accuracy of ResNet-18/50 on ImageNet. Our method achieves much higher accuracy and compression ratio compared to other work. The dash-line is the baseline accuracy from PyTorch zoo.

Table 1: Ternary quantization results on ImageNet. We leave the last FC layer as full-precision like other works.

<table>
<thead>
<tr>
<th>Models</th>
<th>Methods</th>
<th>Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18 Acc.: 69.76</td>
<td>HQ (Ours)</td>
<td>68.5</td>
</tr>
<tr>
<td></td>
<td>TWN (2016) [37]</td>
<td>61.8</td>
</tr>
<tr>
<td></td>
<td>TTQ (2016) [83]</td>
<td>66.6</td>
</tr>
<tr>
<td></td>
<td>INQ (2017) [80]</td>
<td>66.0</td>
</tr>
<tr>
<td></td>
<td>ADMM (2018) [36]</td>
<td>67.0</td>
</tr>
<tr>
<td></td>
<td>LQ-NET (2018) [79]</td>
<td>68.0</td>
</tr>
<tr>
<td></td>
<td>ADAROUND (2020) [57]</td>
<td>55.9</td>
</tr>
<tr>
<td></td>
<td>BRECQ (2021) [41]</td>
<td>66.3</td>
</tr>
<tr>
<td></td>
<td>RTN (2020) [40]</td>
<td>68.5</td>
</tr>
<tr>
<td>ResNet-50 Acc.: 76.15</td>
<td>HQ (Ours)</td>
<td>75.2</td>
</tr>
<tr>
<td></td>
<td>TWN [37]</td>
<td>72.5</td>
</tr>
<tr>
<td></td>
<td>LQ-NET [79]</td>
<td>75.1</td>
</tr>
<tr>
<td></td>
<td>ADAROUND [57]</td>
<td>47.5</td>
</tr>
<tr>
<td></td>
<td>BRECQ [41]</td>
<td>72.4</td>
</tr>
</tbody>
</table>

We also compare our work with conventional ternary quantization works which leave the last FC layer as full-precision (Table 1). Our work achieves comparable results with other leading methods.

5.2. Object Detection and Segmentation

Similar to previous work [67, 53, 41], we test our method on the Mask R-CNN [28] architecture with ResNet-50 backbone to verify its generalizability. The source code, hyperparameters and the pre-trained model are provided by Detectron2 [76]. We apply our method to the entire model except for the first layer. We compare against recent baselines, such as the ABGD [67], PQF [53], and BRECQ [41]. As shown in Table 3, compared to ABGD and PQF, our method gives a higher compression ratio and a similar or better recognition result.

5.3. Model Size and Accuracy

The accuracy, sparsity and size of the quantized ResNet-18 models are shown in Table 4 and Fig. 5. The results show that the model accuracy will increase with the sparsity until it reaches a certain level (the triangle symbol in Fig. 5). Then the accuracy starts to decrease as the pruning continues, which is the same as the Figure 5 in TTQ [83]. This phenomenon is different from pruning, where the accuracy decreases linearly as the sparsity increases. One possible reason is that the capacity [68] of the quantized model changes with the portion of $0$ and $\frac{1}{\sqrt{1+|I_\Delta|}}$. The model capacity is low when the number of $\frac{1}{\sqrt{1+|I_\Delta|}}$ is dominant (close to binary). As the sparsity increases, the weight tends to become ternary, whose capacity is higher than binary weights. Pruning does not have this issue since it has full-precision weight values. As the proportion of 0 keeps increasing, the capacity will drop, leading to accuracy drop.
Table 2: Model compression results on ImageNet. “Bits (W/A)” denotes the bit-width of weight and activation. “Ratio” denotes the storage compression level. “+” denotes 16-bit weight precision in FC layer. “Size” denotes disk size. “*” indicates gzip-compressed publicly available models. The detailed bit allocations can be found in the Appendix.

<table>
<thead>
<tr>
<th>Models</th>
<th>Comp. Level</th>
<th>Methods</th>
<th>Bits (W/A)</th>
<th>Acc.</th>
<th>Size</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>~30×</td>
<td>HQ (Ours)</td>
<td>2/16</td>
<td>67.03</td>
<td>1.23 MB</td>
<td>37×</td>
</tr>
<tr>
<td>Acc.: 69.76</td>
<td></td>
<td>DKM (2022)[7]</td>
<td>32/32</td>
<td>66.7</td>
<td>1.49 MB</td>
<td>30×</td>
</tr>
<tr>
<td>Size: 45 MB</td>
<td></td>
<td>PQF* (2021)[53]</td>
<td>32/32</td>
<td>66.74</td>
<td>1.54 MB</td>
<td>30×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ABDG* (2020)[67]</td>
<td>32/32</td>
<td>65.81</td>
<td>1.51 MB</td>
<td>30×</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>~40×</td>
<td>HQ (Ours)</td>
<td>2/16</td>
<td>65.48</td>
<td>939 KB</td>
<td>48×</td>
</tr>
<tr>
<td>Acc.: 76.15</td>
<td></td>
<td>DKM(2022)[7]</td>
<td>32/32</td>
<td>65.1</td>
<td>1 MB</td>
<td>45×</td>
</tr>
<tr>
<td>Size: 99 MB</td>
<td></td>
<td>PQF* (2021)[53]</td>
<td>32/32</td>
<td>63.33</td>
<td>1.04 MB</td>
<td>43×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ABDG* (2020)[67]</td>
<td>32/32</td>
<td>61.18</td>
<td>1.01 MB</td>
<td>45×</td>
</tr>
<tr>
<td>MobileNetV2</td>
<td>~30×</td>
<td>HQ (Ours)</td>
<td>2/16</td>
<td>75.2</td>
<td>6.89 MB</td>
<td>14×</td>
</tr>
<tr>
<td>Acc.: 71.88</td>
<td></td>
<td>HAWQ (2019)[17]</td>
<td>2<del>8/4</del>8</td>
<td>75.4</td>
<td>7.96 MB</td>
<td>12×</td>
</tr>
<tr>
<td>Size: 14 MB</td>
<td></td>
<td>PQF* (2021)[53]</td>
<td>32/32</td>
<td>75.04</td>
<td>5.10 MB</td>
<td>19×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TQNE (2020)[19]</td>
<td>32/32</td>
<td>74.3</td>
<td>-</td>
<td>19×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ABDG* (2020)[67]</td>
<td>32/32</td>
<td>73.79</td>
<td>5.01 MB</td>
<td>20×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DC (2015)[26]</td>
<td>2<del>8/2</del>8</td>
<td>70.63</td>
<td>6.30MB</td>
<td>16×</td>
</tr>
</tbody>
</table>

Table 3: The model size and Average Precision (AP) with bounding box (bb) and mask (mk) are compared.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Bits (W/A)</th>
<th>AP&lt;bb</th>
<th>AP&lt;mk&gt;</th>
<th>Size</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASELINE</td>
<td>32/32</td>
<td>37.9</td>
<td>34.6</td>
<td>170MB</td>
<td>1×</td>
</tr>
<tr>
<td>HQ (Ours)</td>
<td>2/16</td>
<td>58.74</td>
<td>56.29</td>
<td>0.71 MB</td>
<td>20×</td>
</tr>
<tr>
<td>ABDG (2020)</td>
<td>32/32</td>
<td>33.9</td>
<td>30.8</td>
<td>6.6MB</td>
<td>26×</td>
</tr>
<tr>
<td>PQF (2021)</td>
<td>32/32</td>
<td>36.3</td>
<td>33.5</td>
<td>6.6MB</td>
<td>26×</td>
</tr>
<tr>
<td>BRECQ (2021)</td>
<td>2/8</td>
<td>34.23</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4: Size-accuracy results of ResNet-18 on ImageNet.

<table>
<thead>
<tr>
<th>Epoch</th>
<th>Accuracy (%)</th>
<th>Sparsity (%)</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>65.64</td>
<td>74.06</td>
<td>1.50MB</td>
</tr>
<tr>
<td>99</td>
<td>65.97</td>
<td>75.77</td>
<td>1.40MB</td>
</tr>
<tr>
<td>179</td>
<td>67.03</td>
<td>78.96</td>
<td>1.30MB</td>
</tr>
<tr>
<td>259</td>
<td>66.66</td>
<td>81.86</td>
<td>1.20MB</td>
</tr>
<tr>
<td>299</td>
<td>66.23</td>
<td>84.67</td>
<td>1.10MB</td>
</tr>
<tr>
<td>389</td>
<td>65.37</td>
<td>87.26</td>
<td>0.94MB</td>
</tr>
</tbody>
</table>

Figure 5: The trend lines of size-accuracy and sparsity of ResNet-18. The blue dashed line denotes the model size.

6. Ablation Study

We experiment with the criteria including different pruning settings, reinitialization, and hyperspherical learning. “HYPER” means hyperspherical training. “PRUNING+REINIT” means pruning with reinitialization. “BASELINE/BL” means the pre-trained model from PyTorch or Detectron[76]. Experiments in the Section 6.2 and 6.3 apply full-precision to the last FC layer.

6.1. Hyperspherical Pruning and Reinitialization

We study the change of the cosine distance $D$ (Eq. (8)) brought by our proposed method: applying hyperspherical preprocessing method prior to ternary quantization.

Table 5 shows that pruning can reduce $D$ with or without hyperspherical learning and $D$ tends to decrease along
Table 5: The cosine distance $\mathcal{D}$ of convolutional layers of ResNet-50 on ImageNet.

<table>
<thead>
<tr>
<th>Methods</th>
<th>$\mathcal{D}$</th>
<th>Accuracy</th>
<th>Sparsity</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASELINE</td>
<td>0.262</td>
<td>76.15</td>
<td>0.0</td>
</tr>
<tr>
<td>HYPER</td>
<td>0.288</td>
<td>76.18</td>
<td>0.0</td>
</tr>
<tr>
<td>BASELINE+PRUNING</td>
<td>0.187</td>
<td>76.06</td>
<td>0.4</td>
</tr>
<tr>
<td>HYPER+PRUNING</td>
<td>0.155</td>
<td>76.99</td>
<td>0.4</td>
</tr>
<tr>
<td>HYPER+PRUNING+REINIT</td>
<td>0.068</td>
<td>77.04</td>
<td>0.4</td>
</tr>
<tr>
<td>BASELINE+PRUNING</td>
<td>0.149</td>
<td>76.09</td>
<td>0.6</td>
</tr>
<tr>
<td>HYPER+PRUNING+REINIT</td>
<td>0.056</td>
<td>77.03</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Table 6: Distance comparison of convolutional layers on the object detection task.

<table>
<thead>
<tr>
<th>Methods</th>
<th>$\mathcal{D}$</th>
<th>AP$^b$</th>
<th>AP$^m$</th>
<th>Sparsity</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASELINE</td>
<td>0.263</td>
<td>41.0</td>
<td>37.2</td>
<td>0.0</td>
</tr>
<tr>
<td>HYPER</td>
<td>0.246</td>
<td>41.03</td>
<td>37.54</td>
<td>0.0</td>
</tr>
<tr>
<td>HYPER+PRUNING</td>
<td>0.193</td>
<td>41.33</td>
<td>37.94</td>
<td>0.8</td>
</tr>
<tr>
<td>HYPER+PRUNING+REINIT_1</td>
<td>0.186</td>
<td>40.92</td>
<td>37.53</td>
<td>0.8</td>
</tr>
<tr>
<td>HYPER+PRUNING+REINIT_10</td>
<td>0.113</td>
<td>40.31</td>
<td>36.86</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Table 7: Quantization accuracy of ResNet-18 on ImageNet.

<table>
<thead>
<tr>
<th>Initial Models (Accuracy)</th>
<th>HYPER</th>
<th>NON-HYPER</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASELINE (69.76)</td>
<td>66.45</td>
<td>60.46</td>
</tr>
<tr>
<td>BL+PRUNING+REINIT (69.63)</td>
<td>67.17</td>
<td>66.11</td>
</tr>
<tr>
<td>HYPER+BL+PRUNING+REINIT (69.67)</td>
<td>67.50</td>
<td>65.24</td>
</tr>
</tbody>
</table>

with the growing sparsity. “PRUNING+REINIT” significantly enlarges the distance gap with the pruning-only results and can improve model’s performance [81, 20].

Table 6 shows the difference between applying reinitialization once (REINIT_1) and 10 times (REINIT_10) to the target sparsity of 0.8. “REINIT_10” has a smaller average distance than “REINIT_1”, indicating that iterative pruning and reinitialization encourage model weights close to its ternary version (Section 4.1, Fig. 2b).

6.2. Hyperspherical Ternary Quantization

We perform ternary quantization on ResNet-18 to examine the impact of hyperspherical learning and “PRUNING+REINIT”. The models are initialized by three different pre-trained weights (Table 7). The initialized models are quantized with hyperspherical learning and regular training (“NON-HYPER”) by 100 epochs. Table 7 shows significant improvements brought by hyperspherical learning and “PRUNING+REINIT”. The figure of trend lines can be found in the Appendix.

6.3. The impact of Pruning Settings

We further study the impact of different pruning ranges ($r$) and step sizes ($\delta$, line 14 of Algorithm 1) on ResNet18. The Figure 5 of TTQ [83] indicates that a proper pruning range before ternary quantization is from 0.3 to 0.7, and we take that as a reference. During preprocessing, $r$ changes from 0.3 to 0.7 or 0.4 to 0.8. The step sizes can be 0.01, 0.02, or controlled by cosine annealing. The “PRUNING+REINIT” starts at the 20-th epoch. The total training epochs are 100. Fig. 6 shows that pruned models with larger step sizes and ratios perform poorly and take longer to recover. Using cosine annealing [51] method to adjust the step size $\delta$ improves the overall performance. Fig. 7 shows the following ternary quantization results. Cosine annealing accelerates the convergence.

7. Conclusion

We propose a novel method, Hyperspherical Quantization, to construct sparse ternary weights by unifying pruning, reinitialization and ternary quantization on the hypersphere. The proposed iterative pruning and reinitialization strategy greatly outperform state-of-the-art model compression results in terms of size-accuracy trade-offs. A major contribution of our method is the use of hyperspherical learning to enhance the compression capability. Our work further reveals and demonstrates that pruning and quantization are linked through hypersphere. Our work also explores a new way to extremely compress DNN models without using clustering. Future work may combine our method with ternary activation quantization [6, 39] to further speed up the inference.

References


[61] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy,


