Selecting the Optimal Layerwise Precisions with Iterative Post-training Quantization

Abstract—**As neural network models become larger and the demand for edge devices to run models becomes larger, methods for reducing the size of neural networks have become an important research area. Various methods have been proposed to address this challenge. Among these methods, quantization has shown to be a reliable way to both reduce model size and complexity without totally sacrificing validation accuracy. However, most existing quantization methods require retraining or fine-tuning, which demands the availability of the training data and incurs extra overhead.**

To address this challenge, in this paper we introduce Iterative Quantization, a mixed-precision post-training quantization method that iteratively improves the quantization performance by increasing the bit precision of an optimal layer. At certain bit configurations, this method can outperform fixed-precision quantization by over 50% on the CIFAR-10 dataset. This method also removes the need to retrain models, as it is a post-training quantization method that doesn't need any fine-tuning or any specific training methods.

I. Introduction

In recent years, neural network models, such as ResNet [1], MobileNetV2 [2], and VGG [3] have been getting larger and larger. The utility of having neural networks run on resource-limited edge devices has also grown. These set of circumstances have led to the growing importance of methods to reduce model size without sacrificing accuracy.

Quantization, the reduction in bit precision of the model weights from 32-bits to a lesser precision, has shown to be a practical way to both reduce model size and latency, while still maintaining an acceptable level of accuracy [4]. The most common methods of quantization are fixed-precision methods, where every weight in the model is quantized by the same bit precision (usually by either 1, 4, 8, or 16 bits).

Another method of quantization is mixed-precision quantization, in which the different parts (usually individual layers or individual channels of each layer) of the model are quantized by different amounts depending on some criteria [5], [6]. This approach is more flexible than fixed-precision quantization, leading to better performance. But many mixed precision quantization methods must either be quantized while the model is training or need extensive data to fine tune the model after. Our approach is a post-training quantization method that doesn't require much validation data to quantize the model.

Many existing mixed-precision quantization methods right now are either quantization-aware training, meaning that quantized models need to be retrained or require huge amounts of fine-tuning after training to adjust each layer's bitwitdth. Both retraining and fine-tuning are time-consuming and require the availability of the training data.

In this paper, we introduce Iterative Quantization, which is a method for iteratively searching for the optimal bit configuration for a model being quantized using mixed-precision layers. This is a post-training technique, meaning that quantizing a model doesn't require training from scratch, saving lots of time compared to many other methods which require a model to be quantized while being trained. It also doesn't require lots of fine-tuning to adjust the bit configuration.

Instead of using equations to derive the optimal bit configuration like many previous ideas [6], our method iteratively chooses which layer gets additional bit precisions. The intent of this method is similar to that of gradient descent, in that both methods will step in the direction that offers the best local gain.

This method drastically improves the performance of fixed-precision quantization, the baseline of quantization methods. On the CIFAR-10 [7] dataset, Iterative Quantization outperformed fixed-precision quantization by over 50% at an average bit precision of about 3.

II. Related Work

A. Reducing Neural Network Footprints

As new DNNs utilize more and more weights in ever-mutliplying layers, it has become imperative to find methods of reducing both the memory footprint and computational complexity of neural networks. One set of useful methods to achieve this is quantization. Normally, when a weight is stored in memory, it is in the form of a 32-bit float. However, this can be improved through quantization, which cuts down on the bit precision, sometime going lower than 8-bits per weight. The goal of quantization is to reduce the memory footprint of neural networks without sacrificing the accuracy of the model.

B. Quantization Methods

Low-Precision quantization methods are neural networks that have very low bitwidths (usually around 1-2 bits of accuracy). One example of this is Trained Ternary Quantization [8], which allows for only 3 unique states per layer, (0, a learned lower bound, and a learned upper bound). The resultant model is small, using less than 2 bits per value.

Another prominent category of Quantization methods is Mixed-Precision Quantization (MPQ). MPQ is a class of quantization methods, just like Fixed-Precision Quantization. However, unlike fixed-precision quantization, MPQ's quantize with different bit-precisions for different parts of the neural network. Some previous works have quantized by layer (where each layer has its own bit precision) and others have quantized by channel.

One example of a MPQ method is Bayesian Bits [5], a MPQ algorithm that uses the concept of gates to represent the bit precision of quantization. Some limitations of this method are that it can only work with bit precisions that are a power of two (1,2,4,8,16, and 32 in the paper) and that it requires lots of finetuning to get good results. For each new network or model, Bayesian Bits has to re-tune its bit configuration another time.

Another example of a MPQ method is Hessian-Aware Quantization (HAWQ) [6], which uses the Hessian (2nd derivative) of the loss function of each layer to determine its sensitivity to noise. It bases the bit precision of each layer based on its top Hessian eigenvalue, with layers with high Hessian eigenvalues getting higher bit precisions. The intuition behind the association between 2nd-order behavior and sensitivity to noise comes from the fact that loss functions with higher 2nd-derivatives will change much faster given small weight perturbations.

C. *HERO Training Algorithm*

In this paper, we used the HERO Training Algorithm to train our models [9]. HERO, which stands for Hessian-Enhanced Robust Optimization, is a training algorithm that enhances the loss function of the optimizer to include the Hessian eigenvalues, which allow the loss function to be more resistant to quantization noise and have better performance. We chose HERO over Stochastic Gradient Descent because HERO was designed for quantization in mind and has better optimization characteristics compared to SGD, so it would work better with quantization, and give a better baseline performance to compare against the quantization performance.

III. Methodology

In this section, we will first explain the proposed iterative quantization method, show bit configuration evolution, and provide our explanation on the effectiveness of this method.

A. *Proposed Iterative Quantization Method*

Now we will explore the workings of our method, Iterative Quantization. First, we will define some basic variables and functions.

W_i is the weight tensor for the i^{th} layer. m is the total number of layers in the model.

b is a vector of bit precisions and satisfies the condition $b \in \mathbb{N}^m$.

$\mathrm{acc}(b)$ is the accuracy of the model given the bit configuration b, and the weights W.

$\mathrm{avg}(b)$ is the average bit precision of the model. It is a weighted average of b with W as weights.

$$\mathrm{avg}(b) = \frac{\Sigma_{i=1}^{m} b_i \cdot |W_i|}{\Sigma_{i=1}^{m} |W_i|} \tag{1}$$

From this, we can define an improvement function $f(b_0, i)$, which takes in the original bit configuration, b_0, and the index of the layer we want to increment, i. It then calculates the improvement of a new bit configuration, b', the result of incrementing the i^{th} element in b_0. We define f as the gain in accuracy normalized by the increase in average bit precision.

$$f(b_0, i) = \frac{\mathrm{acc}(b_0 + \hat{u_i}) - \mathrm{acc}(b_0)}{\mathrm{avg}(b_0 + \hat{u_i}) - \mathrm{avg}(b_0)} \tag{2}$$

Here, the unit vector in the i^{th} dimension, $\hat{u_i}$, is the 0-vector with a 1 at the i^{th} entry. For example, $\hat{u_1}$ is:

$$\hat{u_1} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3}$$

To choose the best bit precision to increment, we need to maximize the improvement function f. Our new bit configuration is then:

$$j = \arg\max_i f(b_{k-1}, i), 1 \leq i \leq n \tag{4}$$

$$b_k = b_{k-1} + \hat{u_j} \tag{5}$$

The initial condition b_0 is the bit configuration for fixed-precision quantization (i.e. $[3\ \ 3\ \ 3 \ldots 3]$). We use a greedy strategy for each quantization step. For each iteration, we select one layer to increase the bit precision. The selection is determined by maximizing the benefit brought by the increment, i.e., we select the layer that brings the most accuracy gain for the increment. Compared with retraining or fine-tuning, evaluating the accuracy is relatively fast.

The iterative quantization process for finding the best bit configuration is described in Algorithm 1.

Algorithm 1 Pseudocode for finding the best bit configuration

```
1: Set initial bit precision $b_0$, total iteration $k$.
2: b ← $[b_0\ b_0\ b_0 \ldots b_0]$
3: for a in k do
4:     j ← $\arg\max_i f(b, i), 1 \leq i \leq n$ according to Equation (2)
5:     b ← b + $\hat{u_j}$ according to Equation (5)
   return $b$
```

B. *Bit Configuration Evolution*

To visualize how the bit configuration evolves over time, "Fig. 1" shows the general outline over the entire course of the tuning. We observe that the exploration made by the iterative quantization are steadily explore the average bit range of 3 to 4 within the 75 iterations. "Fig. 2" shows the specific changes in the bit configuration in the first 3 iterations. In this case, our exploration starts with the average bitwidth of 3 for the MobileNetV2 network.

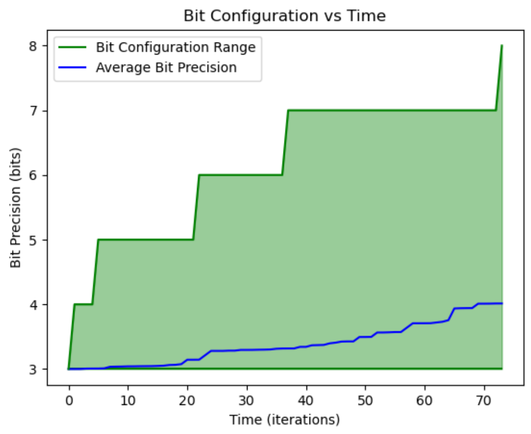

Fig. 1. A visualization of the evolution of the bit configuration, showing the average bit and bit range over time.

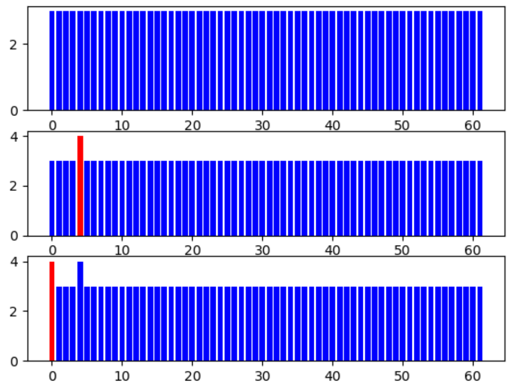

Fig. 2. The quantization bit selection in the first 3 iterations, where the red column is the most recently changed bit precision. This exploration is under MobileNetV2 on CIFAR-10 dataset.

C. Why This Method Works

The thinking behind this method is similar to gradient descent, in which the weight adjusts in the direction of the local optimal direction. In Iterative Quantization, the bit configuration undergoes a similar behavior, incrementing the local optimal layer.

The accuracy normalized by average bit precision was chosen as the function modeling the utility of a certain local step because maximizing it optimizes the accuracy to grow as fast as possible with respect to the bit precision.

IV. Evaluation

A. Experiment Setup

We tested this quantization method by using 3 models: Resnet-20 [1], MobileNetV2 [2], and VGG19 [3], pre-trained and tested on the CIFAR-10 dataset [7]. Resnet-20 has 0.27M parameters, MobileNetV2 has 3.4M parameters, and VGG19 has 143.7M parameters, so our testing models represent a variety model sizes.

TABLE I
Baseline Model Performance

Model	Testing Accuracy
Resnet-20	93.44%
MobileNetV2	95.03%
VGG	94.79%

All models use an cosine learning rate scheduler, with an initial learning rate of 0.1. The optimization method we used is HERO [9], with a momentum of 0.9, an a weight decay of 10^{-4}. On the training dataset, we used three data augmentation methods, that being random crop, padding, and random horizontal flip. We trained the models for 200 epochs and a batch size of 128. These hyperparameters were just to get a good baseline performance, and our method works for all models, no matter what hyperparameters were used to train them.

For the iterative quantization, we ran the iteration for different lengths for the 3 different models. For Resnet-20, we ran the methods for 200 iterations. For MobileNetV2, we ran the methods for 500 iterations. For VGG19, we ran the method for 100 iterations. Each iteration represents testing the model on a subset of data a couple times to determine the local optimal step, no a complete iteration of model training. These discrepancies were due to the iterative speed of each model. MobileNetV2 has over 50 layers, while VGG19 only 19, so it makes sense that MobileNetV2 would iterate slower, as each incremented bit would only increase the average bit precision by a small fraction.

B. Quantization Results

In our experiments, we compared fixed-precision quantization, our iterative quantization method, and a mixed-precision quantization method which bases its bits on the Hessian eigenvalues of each layer, similar to HAWQ [6] but based on the HERO [9] method for calculating curvature loss as a replacement for Hessian eigenvalues. Table. II shows that Iterative Quantization performs much better than the fixed precision method for all models. It also shows that Iterative Quantization performed much better than the HERO-based quantization method on VGG-19.

If we look at a variety of bit precisions for each method, we see that Iterative Quantization is better than the other two methods for low bit precisions. Here we have shown the graph for VGG-19 in “Fig. 3”, which clearly shows a huge performance gap between Iterative and the other methods from 4-5 bits.

C. Quantization Behavior

Visualizing the behavior of our model as shown in “Fig. 4” and “Fig. 5”, we can see that the quantization method has found a handful of layer to prioritize, and that these high-priority layers are relatively consistent throughout the process.

TABLE II

QUANTIZATION PERFORMANCE ACROSS MODELS AND METHODS

Model	Method	Average Bit Precision	Accuracy
Resnet-20	Fixed	3	32.62%
	HERO-based	3	84.00%
	Iterative	**3**	**85.01%**
MobileNetV2	Fixed	3	70.76%
	HERO-based	3	90.11%
	Iterative	**3**	**93.05%**
VGG-19	Fixed	4	32.79%
	HERO-based	4	37.77%
	Iterative	**4**	**69.21%**

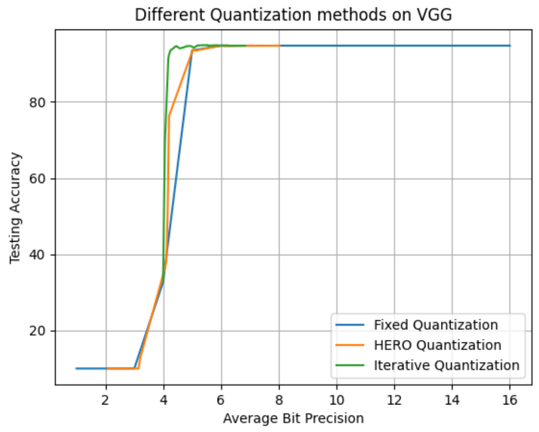

Fig. 3. Iterative, HERO-based, and fixed quantization performance visualized on VGG-19.

This shows that the method has found a few layers that really boost the performance, and has cranked the bit precision of those layers higher than the less important ones. Fig. 5 shows that our method can explore in a relatively large range of bit precision, and thus can fulfill the requirement of flexible and adaptive bit choices.

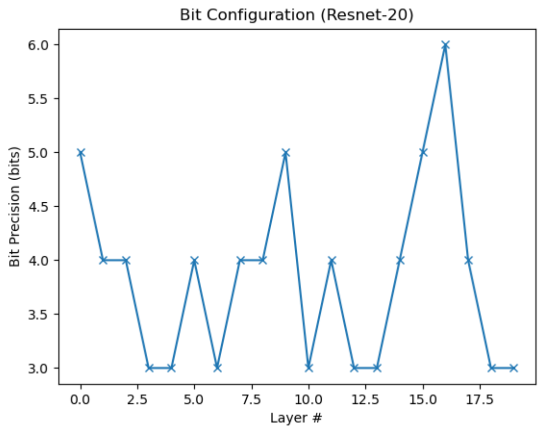

Fig. 4. The bit configuration of Resnet-20 in a representative case with high test accuracy.

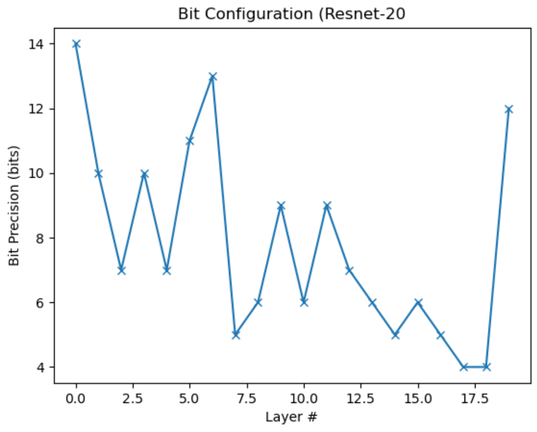

Fig. 5. The Bit Configuration of Resnet-20 for the 100^{th} search iteration.

V. CONCLUSION

This paper introduced a novel post-training quantization method, Iterative Quantization. It performs better than many previous methods, and it is very flexible in the bit configurations offers. This may allow neural networks to run on smaller devices with less processing power by reducing the power draw of operations and storage while still maintaining a good performance.

REFERENCES

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

[2] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, “Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation,” *CoRR*, vol. abs/1801.04381, 2018.

[3] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *CoRR*, vol. abs/1409.1556, 2014.

[4] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding,” in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2016.

[5] M. van Baalen, C. Louizos, M. Nagel, R. A. Amjad, Y. Wang, T. Blankevoort, and M. Welling, “Bayesian bits: Unifying quantization and pruning,” *CoRR*, vol. abs/2005.07093, 2020.

[6] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, “HAWQ: hessian aware quantization of neural networks with mixed-precision,” *CoRR*, vol. abs/1905.03696, 2019.

[7] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” Tech. Rep. 0, University of Toronto, Toronto, Ontario, 2009.

[8] C. Zhu, S. Han, H. Mao, and W. J. Dally, “Trained ternary quantization,” *CoRR*, vol. abs/1612.01064, 2016.

[9] H. Yang, X. Yang, N. Z. Gong, and Y. Chen, “Hero: Hessian-enhanced robust optimization for unifying and improving generalization and quantization performance,” in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 25–30, 2022.