Optimization-Based Post-Training Quantization With Bit-Split and Stitching

Peisong Wang, Weihan Chen, Xiangyu He, Qiang Chen,
Qingshan Liu, *Senior Member, IEEE*, and Jian Cheng, *Member, IEEE*

Abstract—Deep neural networks have shown great promise in various domains. Meanwhile, problems including the storage and computing overheads arise along with these breakthroughs. To solve these problems, network quantization has received increasing attention due to its high efficiency and hardware-friendly property. Nonetheless, most existing quantization approaches rely on the full training dataset and the time-consuming fine-tuning process to retain accuracy. Post-training quantization does not have these problems, however, it has mainly been shown effective for 8-bit quantization. In this paper, we theoretically analyze the effect of network quantization and show that the quantization loss in the final output layer is bounded by the layer-wise activation reconstruction error. Based on this analysis, we propose an Optimization-based Post-training Quantization framework and a novel Bit-split optimization approach to achieve minimal accuracy degradation. The proposed framework is validated on a variety of computer vision tasks, including image classification, object detection, instance segmentation, with various network architectures. Specifically, we achieve near-original model performance even when quantizing FP32 models to 3-bit without fine-tuning.

Index Terms—Deep neural networks, compression, quantization, post-training quantization

1 Introduction

Deep neural networks (DNNs) have been demonstrated to be effective in a wide range of computer vision tasks, including image recognition [1], [2], [3], [4], object detection [5], [6], [7], segmentation [8], [9], and so on. At the same time, the high complexities, including the huge storage and computational requirements, as well as the power consumption, have hindered the deployment of deep networks to real-world applications. Under this circumstance, network compression and acceleration [10], [11], [12], [13], [14] have drawn much attention of researchers. Among these compression techniques, network quantization is widely used no matter on special hardware like TPU or general hardware like CPU and GPU. By turning the floating-point values within the networks to low-bit integers, the complex floating-point operations can be replaced by more efficient integer operations.

Despite its efficiency, the training of low-bit neural networks is nontrivial. In [15], the authors proposed two basic network quantization paradigms, i.e., quantization-aware training and post-training quantization. Most of the current quantization approaches belong to the former, where low-bit networks are trained with quantization operations inserted. Quantization-aware training can achieve higher accuracy because the learned weights could be adjusted during training to fit the quantization operations. On the other hand, quantization-aware training has several drawbacks. It relies on the full training data and large computing resources like GPUs. Moreover, the tedious and time-consuming retraining procedure has posed high requirements in domain knowledge and experience for the users, which is hindering the broader application of quantization techniques.

By contrast, post-training quantization has many desirable properties. It does not need the training dataset, except for a very small amount of data for calibration, thus no privacy or data transmission problems will be caused. Moreover, post-training quantization is network architecture free, back-propagation free, and does not require domain knowledge or any optimization tricks. It is for these reasons that post-training quantization is supported and preferable to accelerate DNN inference by many libraries and devices, such as the TensorRT [16] on GPU, TF-Lite [15] on TPU, and the SNPE on Qualcomm devices.

Despite the advantages of post-training quantization, it has the problem of significant accuracy degradation. Thus for current post-training quantization in TensorRT [16] and TF-Lite [15] as well as the SNPE, only 8-bit quantization is supported. However, for hardware accelerators, every bit saving could result in significant resource reduction, including storage, computing units, as well as energy consumption [17]. To this end, ZeroQ [18] utilizes knowledge distillation and mixed-precision quantization, i.e., allowing different channels and filters to be

- *Peisong Wang, Weihan Chen, Xiangyu He, Qiang Chen, and Jian Cheng are with the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China. E-mail: {peisong.wang, xiangyu.he, qiang.chen, jcheng}@nlpr.ia.ac.cn, chenweihan2018@ia.ac.cn.*
- *Qingshan Liu is with the B-DAT, Nanjing University of Information Science and Technology, Nanjing 210044, China. E-mail: qsliu@nuist.edu.cn.*

Manuscript received 20 April 2021; revised 11 December 2021; accepted 3 March 2022. Date of publication 15 March 2022; date of current version 6 January 2023.
This work was supported in part by the National Key Research and Development Program of China under Grant 2020AAA0103402, in part by the National Natural Science Foundation of China under Grant 61906193, and in part by the Strategic Priority Research Program of Chinese Academy of Sciences under Grant XDA27040300.
(Corresponding author: Jian Cheng.)
Recommended for acceptance by V. Morariu.
Digital Object Identifier no. 10.1109/TPAMI.2022.3159369

quantized into different bit-widths using separate quantization scales. Besides the per-channel activation quantization and mixed-precision quantization, ACIQ [19] also exploits run-time dynamic quantization, which means the quantization parameters (such as the clipping value or quantization scales) are determined at inference time. These techniques substantially improve the quantization performance, however, at the cost of remarkable complexity for inference architecture design. Thus naive post-training quantization with high accuracy still remains an open challenge.

In this paper, we argue that current post-training quantization approaches suffer from three main drawbacks. First, current post-training quantization methods commonly minimize the local quantization error for each weight or activation tensor, without considering the global quantization loss in the final output layer. Second, for the minimization of the local quantization error mentioned above, only the step sizes (i.e., the quantization scales) or equivalently, the clip values, are optimized. While the quantized weights are only adjusted passively with rounding operations caused by the change of the step sizes. Third, the weight quantization and activation quantization are conducted independently.

To solve the above drawbacks, we first theoretically analyze the effect of network quantization and show that the quantization loss in the final output layer is bounded by the layer-wise activation reconstruction error. Then based on this analysis, we propose an Optimization-based Post-training Quantization framework and a novel Bit-split optimization approach to simultaneously optimize the quantization scales as well as the low-bit weights, to achieve minimal accuracy degradation. Besides, by utilizing a re-scaling trick between the activations and weights, we propose the Error Compensated Activation Quantization method, which can benefit from per-channel activation quantization, but still has the same efficiency as per-layer quantization without any extra computations or the need of complicated convolution. The overall framework is effective, hyper-parameter free, and can be readily implemented and integrated into current network quantization libraries.

We comprehensively evaluate the proposed framework on various computer vision tasks, including image classification, object detection, instance segmentation, with various network architectures. Specifically, we achieve near-original model performance even when quantizing FP32 models to 3-bit without fine-tuning, setting new state-of-the-art post-training quantization results. We also implement a low-bit network accelerator on Field-Programmable Gate Array (FPGA) to show the advantages of low-bit post-training quantization. This paper is an extension of our preliminary conference paper [20]. The main contributions are summarized as follows:

- Optimization-based Post-training Quantization framework and Bit-split optimization approach are proposed to simultaneously optimize the quantization scales and the low-bit weights.
- A novel Error Compensated Activation Quantization (ECAQ) method is proposed, which could lower the quantization error for activations.
- We comprehensively investigate the effect of the calibration set on the performance of post-training quantization, which is widely ignored by previous post-training quantization works.

2 Preliminaries

A typical convolutional neural network consists of multiple convolutional and fully connected layers, both of which can be implemented as matrix multiplications as follows:

$$Y = W^T X, \tag{1}$$

where W represents the learnable parameter tensor of convolutional or fully connected layers, X and Y stand for the input and output activation tensors. Note that throughout this paper, we refer to the weights (activations) corresponding to one specific layer as the weight (activation) tensor.

2.1 Network Quantization

The matrix multiplications denoted by Eq. (1) are time-consuming. The purpose of network quantization is to map the floating-point values of W and X into a finite set with discrete elements, which can be encoded by low-bit numbers. Network quantization has been widely studied, in which many design choices should be considered from different viewpoints.

Uniform Quantization versus Non-Uniform Quantization. In non-uniform quantization, there are no restrictions for the discrete values within the finite quantization set. While in uniform quantization, the floating-point real numbers are linearly quantized into low-bit integers, with the same step value between two successive quantization integers. Therefore, uniform quantization can turn the floating-point operations into integer operations, which are more efficient than lookup tables used in non-uniform quantization. A special case of non-uniform quantization, i.e., the logarithmic quantization, is also efficient by turning multiplications into bit shift operations.

Per-Layer Quantization versus Per-Channel Quantization. Quantization can be conducted with varying quantization granularity. In [15], the authors propose per-layer quantization and per-channel quantization. Per-layer quantization adopts a single quantizer (a quantization set) for an entire tensor W or X of one layer. By contrast, per-channel quantization utilizes a separate quantizer for each channel (for activation quantization) or each kernel (for weight quantization). However, per-channel quantization for activations could complicate the convolution operations.

Unified Quantization versus Mixed-Precision Quantization. In unified quantization, we usually specify the bit-width for each layer or even for the whole network. There are many works that explore mixed-precision quantization, where each channel or kernel can be quantized with different bit-widths. Mixed-precision quantization can lower the accuracy loss caused by quantization, however, at the cost of a more complicated quantization process as well as a more complicated computing architecture at inference time.

2.2 Problems of Current Post-Training Quantization

Low-bit network quantization turns all the weights and activations into integers, which can introduce much noise, resulting in dramatic accuracy loss compared with full-precision models. Especially for post-training quantization where fine-tuning is not allowed.

Because no fine-tuning is allowed, most post-training approaches aim at finding a better criterion to *independently* minimize the '*distance*' between each of the pretrained weight tensor W (or activation tensor X) and the quantized one. For example, TF-Lite [15] utilizes the maximum and minimum values of a pretrained model to determine the quantization scales. TensorRT [16] tries to minimize the KL divergence between the quantized distribution and the original distribution to determine the quantization parameters. [21] uses Mean Squared Error (MSE) minimization to find the optimal quantization scales. These straightforward criteria minimize local quantization error for each weight tensor or activation tensor, without considering the global quantization loss in the final output layer.

Besides the suboptimal distance minimization criteria, the discrete optimization problem is another obstacle. In network quantization, the quantized parameters and activations to be optimized are fixed-point integers, which are hard to solve due to the discrete nature. Instead, current post-training quantization approaches only optimize the step sizes (i.e., the quantization scales) or equivalently, the clip values, while the quantized weights are not optimized, and are only adjusted passively with rounding operations caused by the change of the step sizes.

Another problem of current post-training approaches is that they treat weight quantization and activation quantization independently. However, input activation quantization is obviously correlated with the weight quantization of the current layer. Due to these reasons, the current post-training quantization approaches work well for 8-bit quantization. However, the quantization error of these simple criteria could be catastrophic for lower-bit quantization.

In this paper, instead of seeking an approximate criterion, we treat the network quantization as an optimization problem, which can be efficiently solved by the proposed methods.

2.3 Related Work

Deep neural networks (DNNs) are believed to be overparameterized. Many compression approaches are proposed to remove the redundancy within DNN models. Among them, network quantization has become a hot topic in the DNN compression field.

2.3.1 Network Quantization

Quantization is a widely used technique for image and video compression. Researchers also use the quantization technique for deep neural network compression. HashedNets framework is proposed in [22] to group weights into hash buckets, and all connections within the same bucket share the same value. Gong *et al.* [23] first explore product quantization for the weights of the fully connected layers. After that, product quantization is used for Convolutional Neural Network (CNN) compression in [12], [24]. Moreover, instead of directly quantizing the weights, [12], [24] propose to minimize the layer-wise activation reconstruction error to learn the quantized weights. Activation reconstruction is more accurate than weight reconstruction because in network quantization we are interested in preserving the output of the layer but not its weights [24]. However, the relationship between the layer-wise reconstruction error and the final quantization loss is still not clear. In this paper, we show that activation reconstruction can also be used in fixed-point quantization. Moreover, we show that the quantization loss in the final output layer is bounded by layer-wise activation reconstruction error. Based on this, we formulate the fixed-point quantization problem as an activation reconstruction problem and propose a Bit-split framework for efficient optimization.

2.3.2 Fixed-Point Quantization

A widely used network quantization technique is low-precision fixed-point quantization, which can be roughly categorized into two classes: the value approximation and the structure approximation [25]. The value approximation assumes the network architecture is fixed, and the weights and/or activations are quantized into low-bit format. Gupta *et al.* [26] show that deep networks can be trained using only 16-bit number representation. Lin *et al.* [27] extend fixed-point quantization to 16, 8, and 4-bit. In [28], BinaryConnect is proposed to turn weights into -1 or +1, resulting in $32\times$ compression. Further, Binarized Neural Network (BNN) [29] is proposed to turn all weights and activations into -1 or +1. In this way, the multiplication can be replaced by bit-wise operations. After that, many binary approaches are proposed [30], [31], [32], [33], however, there is still large accuracy degradation on large-scale datasets. Multi-step quantization is proposed in [34], [35] to gradually quantize the whole network. The above quantization methods commonly utilize hand-selected and fixed quantization steps. In [36], [37], [38], [39], the quantization step is automatically learned, resulting in much higher accuracy than the fixed quantization step. Besides integer fixed-point quantization, low-precision logarithmic quantization is also studied in [40], [41], [42].

The structure approximation [25] allows the network architecture after quantization to be changed. In [25], the authors propose layer-wise binary decomposition and group-wise binary decomposition. Fixed-point factorization [43], [44] approaches are also proposed to decompose one layer into multiple binary or ternary layers. The Multi-Bit Network (MBN) [45], [46], [47], [48] can be viewed as a special case of structure approximation. In MBN, each floating-point value of the floating-point model is represented by multiple binary bits. The proposed Bit-split method belongs to the value approximation. The final quantized model has the same network architecture as the floating-point counterpart, only the weights and/or activations are changed from floating-point into low-bit format. Thus Bit-split belongs to value approximation. On the other hand, Bit-split shares some properties of the structure approximation (the Multi-Bit Networks). In Bit-split, the original floating-point weights are approximated by multiple ternary bits. After optimization, these ternary bits are combined into low-bit values.

2.3.3 Data-Free Quantization

Low-bit quantization of a pre-trained model generally leads to significant accuracy degradation. To solve this problem, fine-tuning on the original training data is usually needed. However, the training data may not be available. Thus network quantization without training data has been studied.

The authors of [39] propose quantization-interval-learning (QIL) method, and find that QIL can be used on a heterogeneous dataset. Similarly, deep transferring quantization (DTQ) is proposed in [49], which explores knowledge distillation on source dataset to construct low-bit networks for target dataset. ZeroQ proposed in [18] tries to optimize a distilled dataset, which is used later for network quantization. Xu *et al.* [50] explores the generative adversarial networks (GANs) to generator fake images that are used for quantization. Zhang *et al.* [51] aim at diversifying sample generation to improve the quantization using these generated images. These approaches avoid the access of the original training dataset during the quantization stage, however, the time- and resource-consuming fine-tuning using back-propagation on large-scale dataset (heterogeneous or generated dataset) is still of necessity.

2.3.4 Post-Training Quantization

Post-training quantization (PTQ), which is first proposed in [15], has recently drawn much attention. PTQ can be viewed as a special case of data-free quantization. It only needs a very small amount of data for calibration. In addition, PTQ is also back-propagation free, avoiding the time- and resource-consuming fine-tuning process. [19] proposes a run-time analytical clipping approach for integer quantization, which is further improved by per-channel bit allocation. The authors of [52] study post-training quantization from the viewpoint of loss landscape. Mixed-precision post-training quantization is also studied in [18]. Data-free quantization for MobileNet is studied in [53]. In [54], the authors propose a post-training piecewise linear quantization (PWLQ) method to divide the floating-point values into non-overlapping regions, each of which is quantized using a linear quantizer. The PWLQ [54] can also be combined with other post-training quantization methods for higher accuracy.

2.3.5 Second-Order Based Compression

Network compression techniques by making use of the second-order Tayler-series approximation are also explored. LeCun *et al.* [55] and Hassibi *et al.* [56] propose network pruning using second-order derivatives. Recently, Peng *et al.* [57] extend similar ideas to channel pruning. Second-order information is also used for network quantization. The HAWQ method proposed in [58] allows mixed-precision quantization based on the Hessian information. HAWQ uses the top eigenvalue of the Hessian matrix as a sensitivity metric to determine the mixed-precision bit setting. Later, in [59], [60], the authors consider the full Hessian spectrum, specifically, the trace of the Hessian matrix, to determine the bit setting for mixed-precision quantization. However, in the HAWQ series, the Hessian information is only used for relative sensitivity analysis of different layers. After obtaining the bit setting, criteria-based quantization methods are used to conduct the quantization, without using any Hessian information. By contrast, in our approach, we directly optimize the quantized weights based on the second-order information.

In this paper, we aim at the simplest post-training quantization scheme, i.e., we explore uniform quantization with unified bit-width for all layers. Moreover, we use per-channel weight quantization and per-layer activation quantization, which allows for efficient convolution implementation.

3 Method

In this section, we introduce the proposed framework in detail. First, Section 3.1 gives the notations and formulation of the network quantization problem. Then in Section 3.2, we theoretically analyze the effect of network quantization and show that the quantization error in the final output layer is bounded by the layer-wise activation reconstruction error. Next, based on the theoretical analysis, we propose a Bit-split method for uniform weight quantization in Section 3.3.1 and a general iterative optimization for logarithmic weight quantization in Section 3.3.2. Last, an efficient Error Compensated Activation Quantization method is introduced in Section 3.4.

3.1 Problem Formulation

Given a deep neural network f with L layers and a pre-trained model with parameters $\{W_l\}_{l=1}^{L}$, where $W_l \in \mathbb{R}^{d_{l-1}\times d_l}$ is the parameter tensor of the lth layer. Here, d_l represents the number of input neurons of the lth layer. Given the training set $X^0 \equiv \{x_1; \ldots; x_N\} \in \mathbb{R}^{d_0\times N}$, where x_n is the nth input sample, and N is the number of all training samples. We denote the input and the linear output of the lth layer of all training samples as $X^{l-1} \equiv \{x_1^{l-1}; \ldots; x_N^{l-1}\} \in \mathbb{R}^{d_{l-1}\times N}$ and $Y^l \equiv \{y_1^l; \ldots; y_N^l\} \in \mathbb{R}^{d_l\times N}$, respectively. More specifically, we have $Y^l = W_l^T X^{l-1}$, and $X^l = \sigma(Y^l)$ with σ being the non-linear activation function, which is usually the Rectified Linear Unit (ReLU).

For network quantization, we aim to find the low-bit weight tensors $\{\hat{W}_l\}_{l=1}^{L}$ that can minimize the quantization loss, which is defined as the Frobenius norm of the discrepancy between the full-precision output tensor Y^L and quantized output tensor $\hat{Y}^L$ ($\hat{Y}^L = f(\{\hat{W}_l\}; X^0)$) over the whole training data, as follows:

$$\begin{aligned} &\min \frac{1}{N}\|f(\{\hat{W}_l\}; X^0) - Y^L\|_F^2 \\ &\text{s.t. } \hat{W}_{l_{ij}} \in Q_l, \text{ for } l = 1, \ldots, L, \end{aligned} \tag{2}$$

where Q_l represents the quantization values for the lth layer. Take b-bit uniform quantization as an example, $Q_l = \{-2^{(b-1)}, \ldots, 0, \ldots, 2^{(b-1)} - 1\} * \alpha$, where α is the step size between two consecutive quantization values.

3.2 Network Quantization Analysis

Per-channel weight quantization, which utilizes a separate quantizer for each output channel (a neuron for a fully connected layer), is more accurate and is widely used in network quantization. We also use per-channel weight quantization throughout this paper.

Let us consider the weight quantization of a single channel while keeping all the other parameters fixed. The quantized parameters associated with the ith output channel of the lth layer are denoted as $\hat{w}_i^l$. In this way, the quantization error is defined as follows:

$$E = E(\hat{w}_i^l) = \frac{1}{N}\sum_{n=1}^{N}\|f(\hat{w}_i^l; x_n) - y_n^L\|_2^2, \tag{3}$$

where $\hat{w}_i^l = \alpha_i^l * q_i^l$ is the quantized weight vector, which is a product of the integer weight vector q_i^l and the floating-point scale factor α_i^l. Like previous low-bit quantization methods [61], [62], [63], we introduce a separate scale factor α_i^l for each weight vector q_i^l.

To minimize the quantization loss, we utilize the Taylor series approximation to the loss function Eq. (3) following [57], [64], [65] as below:

$$E(\hat{w}_i^l) = E(w_i^l) + \left(\frac{\partial E}{\partial w_i^l}\right)^T \delta w_i^l + \frac{1}{2}\delta {w_i^l}^T H_i^l \delta w_i^l + O(\|\delta w_i^l\|^3), \quad (4)$$

where δw_i^l represents a perturbation (i.e., the quantization noise) of the weight, $H_i^l = \partial^2 E/\partial (w_i^l)^2$ is the Hessian matrix with respect to the parameters w_i^l, and $O(\|\delta w_i^l\|^3)$ is the third and all higher order terms.

According to the definition of the loss function in Eq. (3), it is easy to find that the first term of Eq. (4) equals to 0, i.e., $E(w_i^l) = 0$. As a result, w_i^l is a minimum of E, thus the derivative also equals to 0, i.e., $\partial E/\partial w_i^l = 0$. Moreover, following [55], we assume that the loss function Eq. (3) is nearly quadratic. Thus the third and all higher order terms can also be ignored. Our goal is then to solve

$$\begin{aligned} &\arg\min_{\hat{w}_i^l} \delta {w_i^l}^T H_i^l \delta w_i^l \\ &\text{s.t. } \delta w_i^l = \hat{w}_i^l - w_i^l. \end{aligned} \quad (5)$$

With the loss function defined in Eq. (3), the first order derivatives of the loss function with respect to w_i^l are

$$\frac{\partial E}{\partial w_i^l} = \frac{1}{N}\sum_{n=1}^{N} \frac{\partial E}{\partial y_{n_i}^l} \cdot \frac{\partial y_{n_i}^l}{\partial w_i^l}, \quad (6)$$

where $y_{n_i}^l$ is the output corresponding to the ith channel (or neuron) of the lth layer for input sample x_n. Additionally, the second-order derivatives, i.e., the Hessian matrix of the loss function with respect to w_i^l is defined as

$$\begin{aligned} H_i^l = \frac{\partial^2 E}{\partial (w_i^l)^2} &= \frac{1}{N}\sum_{n=1}^{N} \frac{\partial}{\partial w_i^l}\left[\frac{\partial E}{\partial y_{n_i}^l}\right] \cdot \frac{\partial y_{n_i}^l}{\partial w_i^l} \\ &= \frac{1}{N}\sum_{n=1}^{N} \frac{\partial^2 E}{\partial (y_{n_i}^l)^2} \cdot \frac{\partial y_{n_i}^l}{\partial w_i^l} \cdot \frac{\partial y_{n_i}^l}{\partial w_i^l}^T \\ &= \frac{1}{N}\sum_{n=1}^{N} \frac{\partial^2 E}{\partial (y_{n_i}^l)^2} \cdot x_n^{l-1} \cdot {x_n^{l-1}}^T. \end{aligned} \quad (7)$$

By substituting H_i^l into Eq. (5), we obtain the approximate error caused by the weight quantization as follows:

$$\begin{aligned} E(\hat{w}_i^l) &\approx \delta {w_i^l}^T H_i^l \delta w_i^l \\ &= \delta {w_i^l}^T \left(\frac{1}{N}\sum_{n=1}^{N} \frac{\partial^2 E}{\partial (y_{n_i}^l)^2} \cdot x_n^{l-1} \cdot {x_n^{l-1}}^T\right) \delta w_i^l \\ &= \frac{1}{N}\sum_{n=1}^{N} \frac{\partial^2 E}{\partial (y_{n_i}^l)^2} \cdot (\delta {w_i^l}^T x_n^{l-1})^2 \\ &= \frac{1}{N}\sum_{n=1}^{N} \underbrace{\frac{\partial^2 E}{\partial (y_{n_i}^l)^2}}_{weighting} \cdot (y_{n_i}^l - {x_n^{l-1}}^T \hat{w}_i^l)^2, \end{aligned} \quad (8)$$

which is a weighted average of the activation quantization error with weight coefficient $\frac{\partial^2 E}{\partial (y_{n_i}^l)^2}$. The weighted average minimization problem is still hard to optimize because of the second-order derivatives w.r.t. the output neuron ($\frac{\partial^2 E}{\partial (y_{n_i}^l)^2}$), which can only be obtained by back-propagation. To avoid this, we relax the weighted average minimization problem into an unweighted average minimization, as follows:

$$\begin{aligned} E(\hat{w}_i^l) &\approx \frac{1}{N}\sum_{n=1}^{N} \underbrace{\frac{\partial^2 E}{\partial (y_{n_i}^l)^2}}_{weighting} \cdot (y_{n_i}^l - {x_n^{l-1}}^T \hat{w}_i^l)^2 \\ &\approx \frac{1}{N}\sum_{n=1}^{N} (y_{n_i}^l - {x_n^{l-1}}^T \hat{w}_i^l)^2 \\ &= \frac{1}{N}\|y_{\cdot i}^l - X^T \hat{w}_i^l\|_2^2, \end{aligned} \quad (9)$$

where $y_{\cdot i}^l$ is the output vector of the ith neuron across all training samples, i.e., the ith row of Y^l. In Section 4.6.1, we will discuss the effect of this relaxation.

The last row of Eq. (9), i.e., $\frac{1}{N}\|y_{\cdot i}^l - X^T \hat{w}_i^l\|_F^2$, is the layer-wise activation reconstruction error of layer l. More specifically, Eq. (8) indicates that the quantization error in the final output layer is bounded by the layer-wise activation reconstruction error with a scale factor. Thus, instead of minimizing the final quantization loss defined by Eq. (3), we could alternatively minimize the layer-wise activation reconstruction error as follows:

$$\underset{\hat{w}}{\text{minimize}} \|y - X^T \hat{w}\|_2^2. \quad (10)$$

Note that $\hat{w} = \alpha * q$, thus instead of optimizing $\hat{w}$, we can optimize the quantization scale factor α and the integer weight vector q

$$\begin{aligned} &\underset{\alpha, q}{\text{minimize}} \|y - \alpha X^T q\|_2^2, \\ &\text{s.t. } q^{(i)} \in Q, \end{aligned} \quad (11)$$

where Q is the set of all allowed quantization values. Note that in this representation as well as in the following sections, we discard all the indexes for simplicity.

Eq. (11) indicates that we simultaneously optimize the quantization scale α as well as the low-bit weights q. We want to emphasize that we pose no constraints on q other than the low-bit constraint. More specifically, different from previous post-training quantization approaches, where large full-precision weights are quantized to large low-bit values (i.e., if $w^{(i)} \geq w^{(j)}$, then $q^{(i)} \geq q^{(j)}$). In our objective, each $q^{(i)}$ can be any values in the quantization set Q. Thus we call the proposed framework Optimization-based Post-training Quantization. We will give more analysis about this property in Section 4.6. In Section 3.3, we will discuss how to solve this optimization problem efficiently.

3.3 Optimization-Based Weight Quantization

In Section 3.2, we have demonstrated that the quantization error minimization problem can be transformed into a layer-wise activation reconstruction problem of Eq. (11), which is non-trivial to optimize due to the low-bit constraints of q.

In this section, we propose a novel Bit-split and Stitching (Bit-split) method for Eq. (11) with uniform quantization constraints, which is efficient to optimize. We also give a simple iterative optimization method to deal with logarithmic quantization, which is also hardware-friendly and has been widely used [41], [42], [66], [67], [68], [69].

3.3.1 Bit-Split and Stitching

In this section, we propose a novel Bit-split and Stitching (Bit-split) method for efficient uniform quantization, which utilizes a divide-and-conquer strategy to solve the M-bit integer programming of Eq. (11). The motivation is to split integers into multiple bits (divide), then optimize each bit (conquer), and finally stitch all bits back to integers.

Initialization. Before optimization, we need to initialize α and q in Eq. (11). We use the same strategy of the TF-Lite method, i.e., the quantization scale is selected so that it can map the maximum integer into the maximum floating-point value of the weights. More specifically, for $M-$bit weight quantization, we initialize α and q as follows:

$$\begin{cases} q\_max = 2^{M-1} - 1, \\ \alpha = max(w)/q\_max, \\ q = round(w/\alpha) \end{cases} \tag{12}$$

After the initialization of α and q, we can use the proposed bit-split procedure to update α and q.

Bit-Split. In the Bit-split stage, we split the M-bit constraint of q into $(M-1)$ ternary optimization problems as follows:

$$\begin{aligned} &\underset{\alpha,\{q_1,\ldots,q_{M-1}\}}{\text{minimize}} \parallel y - \alpha X^T(2^0 q_1 + \cdots + 2^{M-2} q_{M-1}) \parallel_F^2, \\ &s.t. \;\; q_m \in \{-1, 0, +1\}^D \text{ for } m = 1, \ldots, M-1, \end{aligned} \tag{13}$$

where D is the vector length of q, q_m is the mth bit of q (from right to left), with the Mth bit (i.e., the left most bit) stands for the sign. Note that all $(M-1)$ bits share the same scale factor α. Moreover, each bit of the M-bit integer has its own implicit base, i.e., the mth bit has an implicit base of 2^{m-1}.

The first step of Fig. 1 illustrates the bit-split operation, which takes the 4-bit quantization for example. For 4-bit integer representation, there are one bit for sign and 3 bits that consist of 0 or 1. Thus, the 4-bit integer can be split into 3 ternary values (i.e., -1, 0, and 1, which takes the sign into consideration), which can be optimized separately. We will show how to optimize these split bits in the following Bit-Optimization section.

*Bit-Optimization.*There are M elements that should be optimized in Eq. (13), i.e., the scale factor α and $M-1$ bits q_m for $m = 1, \ldots, M-1$. We utilize an iterative optimization procedure.

The Optimization of α. From Eq. (11), we can see that α is a floating-point scalar, which can be easily solved given the quantized value q as follows:

$$\alpha = \frac{y^T X^T q}{q^T X X^T q} \tag{14}$$

The Optimization of q_m. To solve Eq. (13) given the scale factor α and all other $M-2$ bits fixed, we can get the following optimization problem:

$$\begin{aligned} &\underset{q_m}{\text{minimize}} \parallel y_m - \alpha_m X^T q_m \parallel_2^2, \\ &s.t. \;\; q_m \in \{-1, 0, +1\}^D, \end{aligned} \tag{15}$$

where y_m and α_m are independent of q_m

$$\begin{cases} y_m = y - \alpha \sum_{i \neq m} 2^{m-1} X^T q_i, \\ \alpha_m = \alpha 2^{m-2} \end{cases}. \tag{16}$$

By expanding Eq. (15) and denoting $A = \alpha_m^2 X X^T$ and $s = -2\alpha_m X y_m$, we could minimize the following equation:

$$\begin{aligned} J(q_m) &= y_m^T y_m - 2\alpha_m y_m^T X^T q_m + \alpha_m^2 q_m^T X X^T q_m \\ &= q_m^T A q_m + s^T q_m + const. \end{aligned} \tag{17}$$

The quadratic optimization problem of Eq. (17) is hard to solve. Here we utilize an iterative optimization procedure, i.e., to optimize each element of q_m with the rest elements fixed. In this way, the kth element of q_m is as follows:

$$q_m^{(k)} = \begin{cases} -sign(r_{\bar{k}}) & \text{if } |r_{\bar{k}}| > A_{kk} \\ 0 & \text{otherwise} \end{cases}, \tag{18}$$

where $r_{\bar{k}} = s_k + 2\sum_{i \neq k} A_{ki} q_m^{(i)}$.

Through iterative optimization from Eqs. (14) to (18), we can get an approximate solution to Eq. (11).

Bit-Stitching. Note that during the bit-split stage, all bits corresponding to the same integer share the same sign, i.e., each integer after bit-split consists of 0/1's or 0/ -1's (Fig. 1b). However, after bit-optimization, this property does not hold any longer. In other words, an integer after bit-optimization may consist of 0/1/-1's (Fig. 1c), where no unified sign bit can be extracted.

To stitch all bits after bit-optimization back into integers, we find that we can simply add all bits together. More specifically, given the optimized bits $q'_1, \ldots, q'_{M-1}$, the new integer values can be obtained through the following equation:

$$q' = 2^0 q'_1 + \cdots + 2^{M-2} q'_{M-1}. \tag{19}$$

Note that during bit-stitching, each of the $M-1$ bits also has its own implicit base, i.e., the mth bit has an implicit base of 2^{m-1}. Figs. 1c to 1d illustrates the bit-stitching procedure.

3.3.2 Logarithmic Optimization

In Section 3.3.1, we have shown how to conduct efficient uniform quantization with Bit-split optimization framework. Besides uniform quantization, logarithmic quantization is another hardware-friendly and widely used quantization method [41], [42], [66], [67], [68], [69]. In this section, we propose a general iterative optimization procedure to deal with the optimization of Eq. (11) with logarithmic quantization constraints. Specifically, by expanding Eq. (11), we have

$$\begin{aligned} J(\alpha, q) &= \|y - \alpha X^T q\|_2^2 \\ &= y^T y - 2\alpha y^T X^T q + \alpha^2 q^T X X^T q \end{aligned} \tag{20}$$

By setting $\frac{\partial J}{\partial \alpha} = 0$, the optimal value of α is given by

$$\alpha^* = \frac{y^T X^T q}{q^T X X^T q} \tag{21}$$

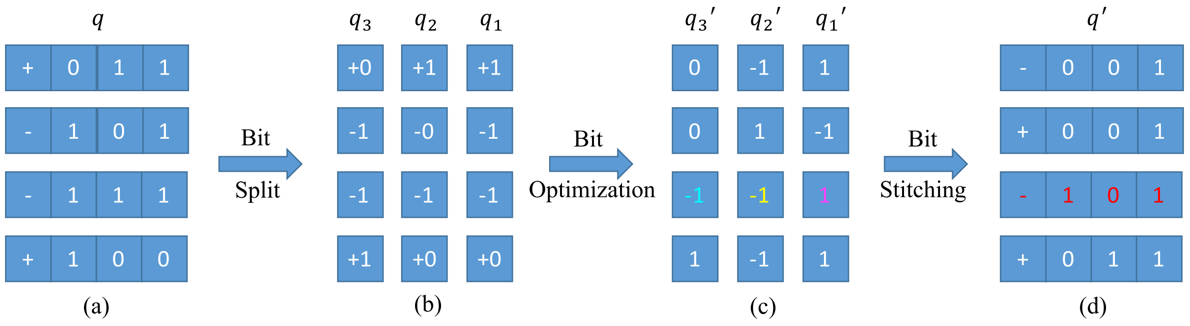


Fig. 1. An illustration of Bit-split and Stitching (Bit-split) optimization procedure for 4-bit weight quantization. (a) Initial low-bit weights before Bit-split optimization. (b) In the bit-split stage, each 4-bit value is split into 3 ternary values. (c) The bit-optimization for the decomposed ternary vectors. (d) The last stage to stitch optimized bits back into integers. Take the third value for example, $2^0 \cdot \mathbf{1} + 2^1 \cdot (\mathbf{-1}) + 2^2 \cdot (\mathbf{-1}) = -5 = \mathbf{-101}$b.

Substituting α^* in Eq. (20), we can get

$$q^* = \arg\max_{q} \frac{(y^T X^T q)^2}{q^T X X^T q}. \tag{22}$$

Suppose q is D-dimensional, for M-bit logarithmic quantization, the programming of Eq. (22) has 2^{DM} feasible points, thus it is impractical to get the optimal solution using exhaustive search.

Instead of pursuing the optimal solution using exhaustive search, we could obtain the approximate solution by utilizing the Discrete Cyclic Coordinate Descent (DCC) method [70]. More specifically, during each coordinate descent step, we only update one element of q while keeping all the other elements unchanged. In this way, we only need to check $D2^M$ possibilities for each optimization cycle.

However, the $D2^M$ iterative optimization is still time-consuming for large bit-width. Thus, we need to control the bit-width to be small. On the other hand, logarithmic quantization has higher coding efficiency than uniform quantization when bit-width is small. As will be shown in Section 4.2.2, logarithmic quantization can notably outperform uniform quantization for 3-bit or 4-bit weight quantization. However, as the bit-width increases, the logarithmic quantization can not make use of the increased representation capacity and achieves much lower accuracy than uniform quantization. Thus, we only utilize the general iterative optimization for logarithmic quantization with $M \leq 4$. We will give more results in Section 4.2.2.

3.4 Error Compensated Activation Quantization

Once the weights are quantized, the quantized weights stay fixed during inference time. However, it is not the case for activations that are dynamically produced at inference time. Thus we could not directly optimize the quantized activations. Therefore, we adopt Mean Squared Error (MSE) as the criterion to minimize the '*distance*' between the activations of the pretrained model and those after quantization.

Different from previous approaches, which utilize one quantizer for a whole layer for activation quantization, the proposed Error Compensated Activation Quantization (ECAQ) method could benefit from per-channel quantization, but still has the same efficiency as per-layer quantization without any extra computations or the need for complicated convolution.

3.4.1 *Per-Channel Activation Quantization*

In this section, we first show that the activation ranges vary severely for different channels. For demonstration, we extract the output of the first and the second convolutional layers (after ReLU) of the VGG-16-BN model for 2400 images which are randomly selected from the training set. We plot the range for each channel as shown in Fig. 3.

From Fig. 3, we can see that large differences are observed in output channel ranges for pretrained models. Due to the large differences between channel ranges, per-layer activation quantization may suffer from large quantization error. Thus in this paper, we explore channel-wise quantizer for activations. Specifically, each input channel is approximated by the quantized channel, which is scaled by a separate scale factor as follows:

$$X[c,:,:] \approx \beta[c]\hat{X}[c,:,:]. \tag{23}$$

Here, β can be optimized by the following formulation:

$$\underset{\beta,\hat{X}}{\text{minimize}} \parallel X - \beta\hat{X} \parallel_F^2 . \tag{24}$$

Fig. 2a shows the per-channel quantization scheme, where different channels with different quantization scales are denoted by different colors.

Per-channel quantization can significantly lower the quantization error, however, at the cost of more complicated convolution operations. We will show how to solve this problem in the next section.

3.4.2 *From Per-Channel to Per-Layer Quantization*

As shown in the previous section, per-channel activation quantization could complicate convolution operations. This is because that the convolutions (CONV) are implemented as matrix multiplications (MatMul), as shown in the bottom row of Fig. 2. When each channel has a separate scale factor, the large matrix multiplication is separated into multiple small matrix multiplications, which could dramatically reduce the computing efficiency.

In this section, we exploit the re-scaling of activations and weights to facilitate simple convolution implementation when per-channel activation quantization is utilized. Specifically, we can move the scale factor of each input channel to its corresponding 2D kernels of all filters

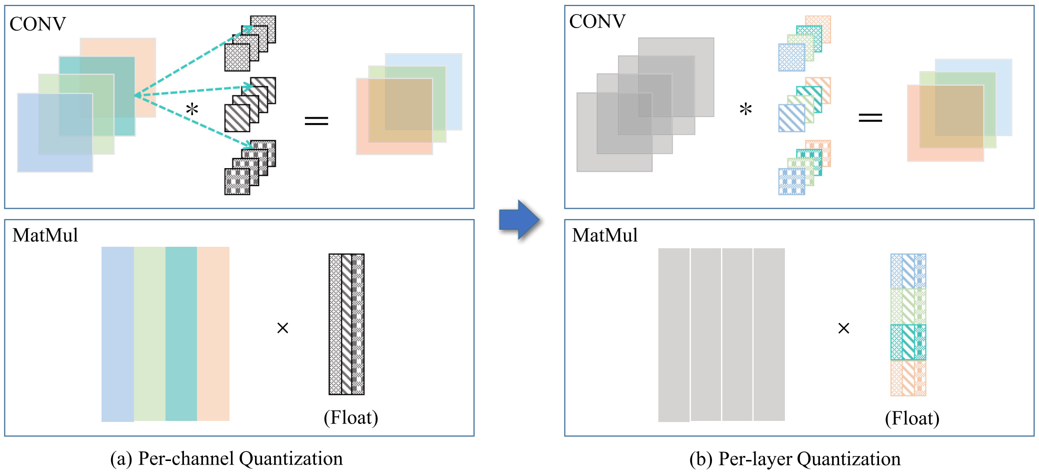


Fig. 2. (a) Illustration of per-channel activation quantization. Each input channel is quantized using a separate scale factor denoted by a specific color. (b) Illustration of transforming per-channel quantization into per-layer quantization by re-scaling. The scale factors of input activations are merged into the corresponding 2D kernels of the following floating-point filters. The bottom row shows how to conduct convolution (CONV) using matrix multiplication (MatMul) operations. *Note that at this stage, the weights are not quantized yet.* After merge, the floating-point filters can be further quantized into integers using the Optimization-based Weight Quantization proposed in Section 3.3, so that the convolutions can be conducted by integer-only accumulations. Best viewed in color.

$$W * X \approx W * \beta \cdot \hat{X} \\ = W_\beta * \hat{X}, \tag{25}$$

where W_β satisfies

$$W_\beta[:, c, :, :] = \beta[c]W[:, c, :, :]. \tag{26}$$

The re-scaling operation does not affect the convolution output. Note that at this stage, the weights are not quantized yet, i.e., the weights are floating-point values. After re-scaling, we can further quantize the weights W_β using the Bit-split method according to Section 3.3.1 as follows:

$$\underset{\alpha_\beta, q_\beta}{\text{minimize}} \parallel y - \alpha_\beta \hat{X}^T q_\beta \parallel_2^2, \tag{27}$$

where α_β and q_β represent the scale factor and quantized filter for the re-scaled weights W_β. Fig. 2b illustrates the convolution after re-scaling, which can also be efficiently implemented by matrix multiplication.

From Eq. (27), we can see that we are actually learning a mapping from the *per-layer quantized activations* $\hat{X}$ to the target output Y by taking the activation quantization and scale factors β into consideration. Thus we call the proposed method Error Compensated Activation Quantization. We summarize the overall optimization procedure of the proposed approach in Algorithm 1.

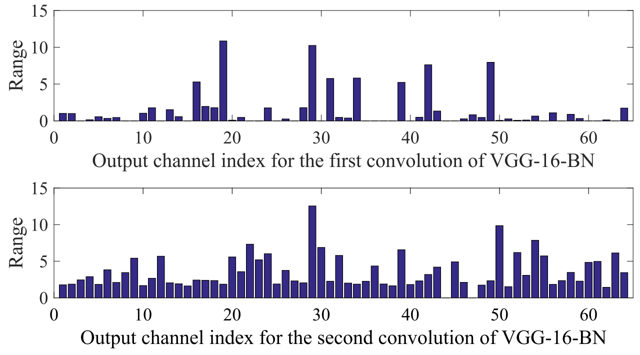


Fig. 3. The ranges of activations for each channel (after ReLU) of the first and second convolutions of VGG-16-BN. It can be seen that the ranges for different channels differ severely.

Algorithm 1. Post-Training Quantization Using Error Compensated Activation Quantization and Bit-Split and Stitching Weight Quantization

Input: Pretrained model denoted by weights $\{W^l\}_{l=1}^L$, weight bit-width M_w and activation bit-width M_a.
Output: Quantized weights $\{q^l\}_{l=1}^L$, quantization scale $\{\alpha^l\}_{l=1}^L$ for weights and $\{\beta^l\}_{l=2}^L$ for activations.

```
1:  for $l = 1; l \leq L$ do
2:      Sampling a mini-batch images
3:      Forward propagation to get $X^l$ and $Y^l$
4:      if $l > 1$ then
5:          Optimize activation quantization scale $\beta^l$ according to Eq. (24)
6:          Move $\beta^l$ from activations to weights to obtain $W_\beta^l$ according to Eq. (26)
7:      else
8:          $W_\beta^l \leftarrow W^l$, $\hat{X}^l \leftarrow X^l$
9:      end if
10:     $q\_max = 2^{M_w - 1} - 1$
11:     $\alpha^l \leftarrow max(W_\beta^l)/q\_max$
12:     $q^l \leftarrow round(W_\beta^l/\alpha^l)$
13:     Calculate $q_1^l, \ldots, q_{M_w-1}^l$ using Bit-split
14:     while $\alpha^l$ not converge do
15:         Optimize $\alpha^l$ according to Eq. (14)
16:         for $m = 1; m < M_w$ do
17:             Optimize $q_m^l$ according to Eq. (18)
18:         end for
19:         Calculate $q^l$ using Bit-Stitching
20:     end while
21: end for
```

4 Experiments

In this section, we evaluate the efficiency of the proposed method. We first evaluate the Bit-split and Stitching method

TABLE 1
Comparison Results of the Top-1 and Top-5 Accuracy (%) for Uniform *Weight Quantization With Various Bit-Widths*

Model		8-bit		7-bit		6-bit		5-bit		4-bit		3-bit	
		Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
ResNet-18 (69.76, 89.08)	TF-Lite [15]	69.63	88.96	69.67	89.02	69.06	88.72	66.81	87.39	55.53	79.21	0.85	2.68
	Bit-split	69.79	**89.15**	**69.84**	**89.15**	**69.83**	**89.12**	**69.70**	88.93	**69.11**	**88.69**	**66.76**	87.45
	Bit-split (A8)	**69.82**	**89.15**	69.82	89.05	69.80	**89.12**	69.64	**88.98**	69.10	**88.69**	66.75	**87.46**
ResNet-50 (76.15, 92.87)	TF-Lite [15]	76.12	92.88	76.07	92.86	75.87	92.82	75.17	92.50	70.14	89.57	4.22	11.53
	Bit-split	**76.20**	**92.97**	**76.16**	**92.91**	**76.17**	**92.90**	**76.05**	**92.82**	**75.58**	**92.57**	**73.64**	**91.61**
ResNet-101 (77.47, 93.56)	TF-Lite [15]	77.32	93.57	77.28	93.51	77.06	93.47	76.25	93.05	72.67	90.87	9.19	20.05
	Bit-split	**77.55**	**93.59**	**77.44**	**93.59**	**77.51**	**93.60**	**77.55**	**93.59**	**76.89**	**93.31**	**74.98**	**92.42**
VGG-16-BN (73.37, 91.50)	TF-Lite [15]	73.36	91.51	73.34	91.48	73.12	91.36	72.37	90.86	66.36	87.26	1.16	4.49
	Bit-split	**73.43**	**91.61**	**73.37**	**91.52**	**73.22**	**91.53**	**73.37**	**91.50**	**72.97**	**91.35**	**72.11**	**90.77**
MobileNetV2 (71.87, 90.28)	TF-Lite [15]	71.62	**90.28**	70.24	89.46	65.11	86.02	59.31	82.03	13.42	30.85	0.18	0.81
	Bit-split	**71.94**	90.27	**71.86**	**90.26**	**71.77**	**90.14**	**70.91**	**89.96**	**68.41**	**88.47**	**54.49**	**78.68**

Per-channel weight quantization and floating-point activations are utilized. For ResNet-18, the results with 8-bit per-layer activation quantization, denoted by Bit-split (A8), are also given, demonstrating that 8-bit activation quantization has no accuracy loss compared with floating-point activations. Bold values indicate the best results.

for weight quantization. Then the performance of Error Compensated Activation Quantization is evaluated. We also compare our method with current post-training methods.

4.1 Implementation Details

Like previous Post-training Quantization approaches, the proposed Bit-split framework relies on a batch of calibration images. For ImageNet experiments, if not specified, we randomly select 1024 images from the ImageNet training set as the calibration set. During the optimization procedure, for each layer, we extract the input and output feature maps, then we randomly sample 12000 data points (the feature vector of each spatial position at the output feature maps is treated as a data point) to form the X and Y for the optimization of Eq. (11). For all experiments, we quantize the first and the last layer into 8-bit, which is the common setting in network quantization [19], [29]. We report the Top-1 and Top-5 accuracy for ImageNet classification. Pytorch deep learning framework is utilized for all our experiments, and all ImageNet pretrained models are from torchvision. We use GPU for feature maps extraction, while use CPU for the quantization optimization procedure. The Bit-split optimization for ResNet-18 can be accomplished within 15 minutes on CPU, which is negligible compared with several GPU days needed by Quantization-aware Training. All bit-width representation throughout this paper includes the sign bit. Codes are available on GitHub at https://github.com/wps712/BitSplit.

4.2 Weight Quantization Results

In this section, we evaluate the post-training weights quantization using the proposed approach. Both uniform quantization results using Bit-split optimization as well as the logarithmic quantization results are reported.

4.2.1 Bit-Split and Stitching for Uniform Quantization

We evaluate uniform quantization with Bit-split and Stitching method. The Top-1 and Top-5 accuracy results of post-training quantization are reported using four popular convolutional models pre-trained on the ImageNet dataset. We use the PyTorch pretrained models for all experiments. The results are shown in Table 1. We quantize the weights into 3$\sim$8 bit while keeping activations un-quantized. We also report the TF-Lite [15] results as a baseline for comparison.

From Table 1, it is clear that the proposed Bit-split and Stitching method has very small accuracy degradation for post-training weight quantization without fine-tuning. Our method consistently outperforms the TF-Lite method. The accuracy gap becomes larger as the bit-width goes down. We first compare the proposed method with full-precision models as well as TF-Lite models for ResNets and VGG-16. Using the proposed Bit-split and Stitching method, no obvious accuracy loss is observed for 4-bit quantization or higher. We also want to highlight that for 3-bit quantization, previous approaches like the TF-Lite fail to work, while in our Bit-split and Stitching method, only 0.9%$\sim$1.7% Top-5 accuracy drop is observed for various networks. Then, for the light-weight MobileNet-V2 architecture, both the proposed method and TF-Lite have higher accuracy drop than that on ResNets and VGG-16. However, the proposed method still achieves promising results for various bit-widths, outperforming the TF-Lite method by large margins. Specifically, for 5-bit and 4-bit weight quantization, the proposed Bit-split method has 0.3% and 1.8% Top-5 accuracy drop, respectively. Even for 3-bit quantization, the Bit-split method still achieves reasonable accuracy.

In Table 1, we also report the results when activations are quantized into 8-bit for ResNet-18, denoted by Bit-split (A8). We can see that 8-bit activation quantization has no loss compared with floating-point activations. The differences in results (less than 0.1%) may be caused by the randomness.

4.2.2 Logarithmic Quantization Results

In this section, we evaluate the proposed general optimization for logarithmic weight quantization. We quantize the weights into 3 $\sim$ 8 bit to show how the accuracy will change as the bit-width increases for logarithmic quantization. In all experiments, ResNet-18 and ResNet-50 are utilized for evaluation. The results compared with uniform quantization are summarized in Fig. 4.

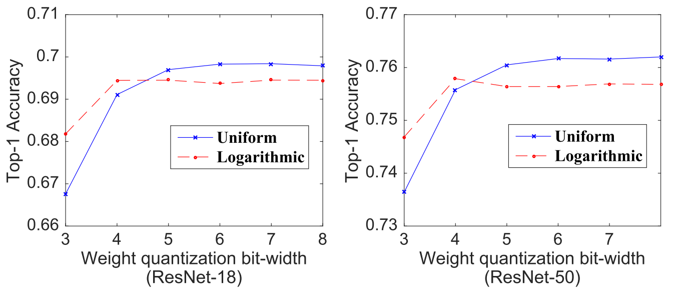


Fig. 4. The accuracy comparison between uniform weight quantization and logarithmic weight quantization with various bit-widths.

Fig. 4 shows that logarithmic quantization can notably outperform uniform quantization for 3-bit and 4-bit weight quantization. Specifically, 3-bit logarithmic quantization can achieve 68.18% and 74.69% Top-1 accuracy for ResNet-18 and ResNet-50, outperforming uniform quantization by 1.42% and 1.04%, respectively. These results are promising considering that only 8 values are allowed for all the weights. To our knowledge, *this is the first work that achieves close-to-full-precision accuracy for 3-bit weight quantization without fine-tuning*. With 4-bit quantization, the accuracy gaps between logarithmic and uniform quantization become 0.34% and 0.21% for ResNet-18 and ResNet-50. However, as have been discussed in Section 3.3.2, logarithmic quantization has much lower coding efficiency than uniform quantization when bit-width increases further. It can be verified from Fig. 4 that the accuracy of logarithmic quantization stops growing as the bit-width becomes higher than 4. Actually, the accuracy of 5- $\sim$ 8-bit logarithmic quantization is slightly lower than that of the 4-bit logarithmic quantization. Besides the coding efficiency, the optimization of logarithmic quantization is more time-consuming than uniform quantization using Bit-split method. Take the 8-bit quantization as an example, the optimization time is more than 2 hours for logarithmic quantization compared with 14 minutes for uniform quantization. Thus, we only prefer to use the general iterative optimization for logarithmic quantization with bit-width $M \leq 4$.

4.3 Error Compensated Activation Quantization

In this section, we evaluate the proposed Error Compensated Activation Quantization (ECAQ) approach thoroughly. We compare per-layer activation quantization, per-channel activation quantization, and the proposed ECAQ. To be clear, we summarize the main difference between these three activation quantization schemes bellow:

- *Per-layer:* Per-layer quantization adopts a single quantization scale for the entire input tensor X. The convolution can be transformed into one large matrix multiplication by unrolling (a.k.a. *im2col*) [71].
- *Per-channel:* Per-channel quantization utilizes a separate quantization scale for each input channel X_i. The convolution is transformed into the sum of C_{in} (the number of input channels) small matrix multiplications, which is less efficient than per-layer quantization.
- *ECAQ:* Each input channel has a separate quantization scale, however, these channel-wise quantization scales are merged into weights of the following convolution. Thus the convolution can also be transformed into one large matrix multiplication.

ResNet-18 network is adopted for demonstration. We quantize activations into 3$\sim$8 bit and weights to 8-bit using Bit-split. The results are shown in Table 3.

Table 3 shows that per-channel activation quantization indeed generally outperforms per-layer quantization under various bit-width settings. However, as shown in Section 3.4, per-channel activation quantization can complicate convolution implementation. By contrast, both the proposed ECAQ method and direct per-layer quantization utilize a single quantizer for a whole layer, which is efficient at inference time. For higher bit quantization, the ECAQ method achieves similar results (less than 0.1% differences) as direct per-layer quantization. However, for bit-width less than 5 bit, the proposed ECAQ consistently outperforms per-layer quantization. Especially, for 4-bit and 3-bit quantization, ECAQ outperforms direct per-layer quantization by 0.45% and 1.44%, respectively, with a single scale for a whole layer.

To further demonstrate the effectiveness of the proposed framework thoroughly, we give the results of Error Compensated Activation Quantization coupled with Bit-split and Stitching weight quantization for various bit-widths on different networks. The results are shown in Table 2. Like

TABLE 2
Comparison Results of Top-1 and Top-5 Accuracy (%) for Uniform Quantization of *Both Weights and Activations With Various Bit-Widths*

Model		8-bit		7-bit		6-bit		5-bit		4-bit		3-bit	
		Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
ResNet-18 (69.76, 89.08)	TF-Lite [15]	69.57	89.02	69.46	88.87	67.95	88.02	61.47	83.43	18.84	36.33	0.13	0.61
	Bit-split	**69.74**	**89.09**	**69.68**	**89.07**	**69.58**	**88.96**	**69.28**	**88.77**	**67.56**	**87.76**	**61.30**	**83.47**
ResNet-50 (76.15, 92.87)	TF-Lite [15]	**76.05**	**92.93**	75.75	92.70	73.83	91.66	65.46	86.34	10.40	22.36	0.11	0.54
	Bit-split	75.96	92.83	**76.09**	**92.84**	**75.90**	**92.75**	**75.38**	**92.59**	**73.71**	**91.62**	**66.22**	**87.18**
ResNet-101 (77.47, 93.56)	TF-Lite [15]	76.78	93.31	74.07	91.79	31.78	55.96	0.82	2.65	0.25	0.98	0.09	0.54
	Bit-split	**77.23**	**93.55**	**77.20**	**93.47**	**76.93**	**93.42**	**76.07**	**92.95**	**74.68**	**92.18**	**63.96**	**85.65**
VGG-16-BN (73.37, 91.50)	TF-Lite [15]	73.31	91.53	72.94	91.25	70.65	89.77	54.45	78.18	3.41	10.17	0.18	0.78
	Bit-split	**73.43**	**91.54**	**73.43**	**91.55**	**73.34**	**91.45**	**72.89**	**91.22**	**71.14**	**90.29**	**66.11**	**86.92**
MobileNetV2 (71.87, 90.28)	TF-Lite [15]	71.15	90.07	68.54	88.51	51.72	75.40	10.95	25.52	0.19	0.93	0.10	0.52
	Bit-split	**71.73**	**90.22**	**71.36**	**90.03**	**70.16**	**89.31**	**65.94**	**86.93**	**49.23**	**74.21**	**9.59**	**23.76**

We use per-channel scale factors for the weights and per-layer scale factors for the activations. Bold values indicate the best results.

TABLE 3
Comparison Results of the Top-1 Accuracy (%) for Post-Training Activation Quantization of ResNet-18

Act. Quant. Type	FP32	8-bit	7-bit	6-bit	5-bit	4-bit	3-bit
Per-layer	69.76	**69.82**	69.78	69.79	69.37	68.31	64.77
Per-channel	69.76	69.79	**69.86**	**69.79**	**69.63**	**68.83**	65.91
ECAQ	69.76	69.74	69.75	69.70	69.52	68.76	**66.21**

Weights are quantized to 8-bit using Bit-split with per-channel quantization scales. The boldfaced and underlined values stand for the best and second best results for each setting.

previous settings, the Top-1 and Top-5 accuracy results for four popular convolutional models are reported. We quantize both weights and activations simultaneously into 3$\sim$8 bit. We also use TF-Lite as a baseline for comparison.

Table 2 shows that when both activations and weights are quantized using the same bit-width, the proposed framework still dramatically outperforms TF-Lite by large margins. For ResNet-18, ResNet-50 and VGG-16-BN, TF-Lite fails at 4-bit quantization. By contrast, the proposed method has less than 1.3% Top-5 accuracy degradation for these three networks at 4-bit quantization for both weights and activations. Note that even when both weights and activations are quantized to 3-bit, only 5 points accuracy loss is observed, which is encouraging considering the very limited representation space and no fine-tuning is used. For the very deep ResNet-101, the baseline approach fails at 4-bit, while our approach still works at 3-bit quantization. For the light-weight MobileNet-V2, Bit-split achieves small (less than 1%) Top-5 accuracy drop with 6-bit weights and activations.

In summary, the results in Table 2 show that the proposed framework, i.e., Error Compensated Activation Quantization coupled with Bit-split and Stitching weight quantization, is powerful for post-training uniform quantization. We will compare the proposed framework with previous works thoroughly in the following section.

4.4 Comparison With the State-of-the-Arts

In this section, we evaluate the proposed framework with existing post-training quantization approaches. The results are shown in Table 4, which are summarized under different bit-widths: A8W4 represents 8-bit activations and 4-bit weights, and A4W4 (A6W6) indicates both weights and activations are quantized into 4-bit (6-bit). For a comprehensive comparison, we also report whether per-layer or per-channel activation quantization, and unified-precision or mixed-precision quantization schemes are utilized.

Table 4 shows that the proposed bit-split framework dramatically outperforms previous results when per-layer activation quantization and unified precision quantization schemes are utilized. With 4-bit weight quantization, our bit-split framework even outperforms the current state-of-the-art method ACIQ-Mix, which utilizes per-channel quantization, mixed-precision, and dynamic quantization. When we utilize per-channel quantization for the proposed bit-split framework, the accuracy can further be improved, setting new state-of-the-arts for post-training quantization.

4.5 Object Detection and Instance Segmentation

To show the generalization ability of the proposed framework, we conduct experiments on object detection and instance segmentation. MS COCO dataset is used for

TABLE 4
Comparison of Different Post-Training Quantization Approaches on ImageNet Classification Benchmark

Model		Per-layer	Unified-precision	ResNet-18	ResNet-50	ResNet-101	VGG-16-BN
Full-precision		-	-	69.8	76.2	77.5	73.4
A8W4	TF-Lite [15]	√	√	55.5	70.1	72.6	66.4
	ACIQ [19]	×	√	67.4	74.8	76.3	71.7
	ACIQ-Mix [19]	×	×	68.3	75.3	**76.9**	72.4
	Bit-split	√	√	**69.1**	**75.6**	**76.9**	**73.0**
A6W6	TF-lite [15]	√	√	63.8	-	-	-
	TF-lite-per-channel	×	√	67.5	-	-	-
	DFQ[52]	√	√	66.3	-	-	-
	Bit-split	√	√	**69.6**	**76.2**	**77.5**	**73.2**
A4W4	TF-Lite [15]	√	√	18.8	10.4	0.3	3.4
	DFC [71]	√	√	55.5	-	-	-
	TensorRT [16]	√	√	31.9	46.2	49.9	-
	LAPQ [51]	√	√	59.8	70.0	59.2	-
	ACIQ-Mix [19]	×	×	67.0	73.8	75.0	**71.8**
	Bit-split	√	√	67.6	73.7	74.7	71.1
	Bit-split-per-channel	×	√	**68.1**	**74.2**	**75.3**	**71.8**

Top-1 accuracy (%) is reported. We also indicate whether per-layer or per-channel activation quantization, and unified-precision or mixed-precision quantization schemes are utilized. Weights are quantized using per-channel scale factors. Bold values indicate the best results.

TABLE 5
Object Detection (Bounding Box AP) and Instance Segmentation (Mask AP) Results on COCO Minival Set

Model		$AP_{0.5:0.95}$	$AP_{0.5}$	$AP_{0.75}$
RetinaNet (Box)	Full-precision	35.6	55.5	38.3
	ACIQ (A8W4)	34.0	**54.3**	36.3
	Bit-split (A8W4)	**34.4**	54.2	**36.5**
Mask R-CNN (Box)	Full-precision	37.3	59.0	40.2
	ACIQ (A8W4)	36.0	57.4	38.8
	Bit-split (A8W4)	**36.2**	**57.5**	**39.3**
Mask R-CNN (Mask)	Full-precision	34.2	55.9	36.2
	ACIQ (A8W4)	33.0	53.8	35.1
	Bit-split (A8W4)	**33.4**	**54.4**	**35.4**

We use per-channel scale factors for the weights and per-layer scale factors for the activations.

evaluation. We use mmdetection[1] toolbox for the experiments. The pre-trained models are trained on 80k training images and 35k of validation images (trainval35k). The accuracy is evaluated on the remaining 5k validation images (minival). Input images are resized to 800 pixels in the shorter edge.

We evaluate the bit-split framework using single-stage object detection of RetinaNet [73], as well as the object detection and instance segmentation using two-stage Mask R-CNN [74]. Both networks using ResNet-50 as a backbone. We quantize all layers into 4bit except the first layer and the final output layers which are quantized to 8bit. Activations are quantized into 8bit. The results are shown in Table 5. We also report the post-training quantization results of ACIQ for comparison. We can see that there are about 0.8%$\sim$1.2% mAP degradation with 8bit activations and 4bit weights without fine-tuning, which demonstrates the generalization ability of the proposed framework. The proposed Bit-split method outperforms ACIQ by 0.2%$\sim$0.4% mAP on both networks and tasks.

4.6 Optimization-Based Quantization Analysis

In this section, we analyze the Hessian approximation and the quantization effect of the Bit-split method.

4.6.1 Hessian Approximation Analysis

In the Hessian matrix approximation of Eq. (9), we relax the weighted quantization error minimization problem into an unweighted minimization problem. However, is it good enough to relax the weighted error minimization into unweighted error minimization? The answer to this question is not apparent. Actually, our target is to minimize the quadratic form of the Hessian matrix H and the weight perturbation δw, i.e., the $\delta w^T H \delta w$. But instead, we choose to minimize the activation reconstruction error of Eq. (9). Thus, we need to check the correlation between the Hessian error term and the activation reconstruction error.

To this end, we calculate the hessian error term and the corresponding activation reconstruction error by randomly selecting 200 perturbations, for a convolutional kernel in ResNet-18. The results are shown in Fig. 5. It can be seen

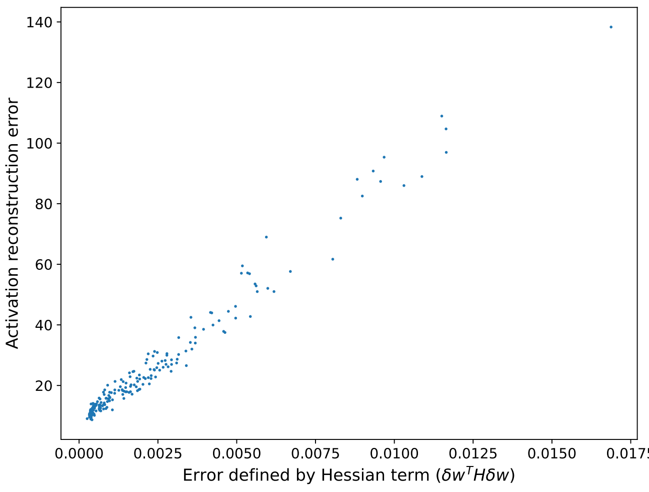


Fig. 5. The correlation between the Hessian error term and the activation reconstruction error for a convolutional kernel of ResNet-18.

that these two error terms have a high correlation. Thus we could use the activation reconstruction error minimization as a delegate task for the minimization of the Hessian error term.

4.6.2 Quantization Result Analysis

One key property of the proposed Optimization-based Post-training Quantization framework is that we simultaneously optimize the quantization scale α as well as the low-bit weights q, which can be seen from Eq. (11). More specifically, previous post-training quantization methods mainly optimize the quantization scale α, while all low-bit weights are obtained by rounding operations. This quantization strategy implies that large full-precision weights are quantized to large low-bit values (i.e., if $w^{(i)} \geq w^{(j)}$, then $q^{(i)} \geq q^{(j)}$). However, in the proposed Optimization-based Post-training Quantization framework, each $q^{(i)}$ can be optimized into any values in quantization set Q. In this section, we compare the quantization effect of the proposed framework with MSE based quantization method [21]. The results are illustrated in Fig. 6.

Fig. 6a shows how many weights have changed their values after Bit-split optimization, compared with MSE based post-training quantization method [21]. Overall, 15.45% weights have changed their values. The distributions of the absolute value change among these weights are illustrated in Fig. 6b. The majority of the weights (93.4%) have changed their values by 1. There are about 5.6% weights that have changed by 2. The rest changes occupy about 1%. As for the quantization scale change, we extract the first 50 quantization scales of the 16th convolutional layer, which are illustrated in Fig. 6c. We can see that after optimization, the quantization scales become a bit larger. Note that we find this phenomenon for all layers, not just for the 16th layer. The large quantization scale indicates that the learned quantizer after optimization pays more attention to the weights with large values.

4.7 Calibration Set Analysis

In post-training quantization, commonly a small number of calibration images are required. In this section, we explore

1. https://github.com/open-mmlab/mmdetection

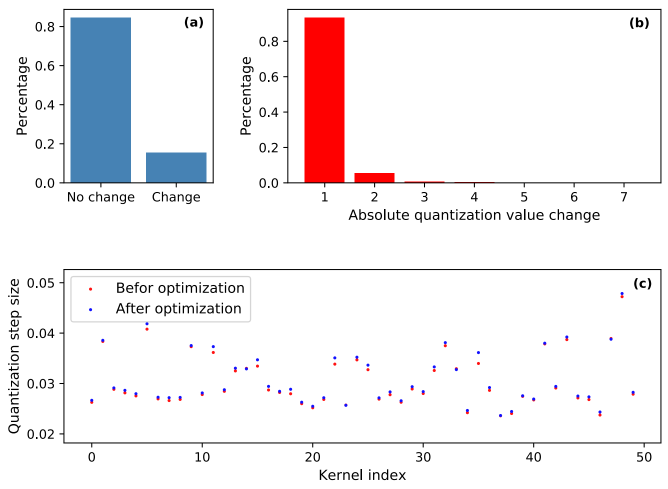


Fig. 6. The changes of quantization scales and the low-bit weights after W4 Bit-split optimization on ResNet-18. (a) The percentage of quantized weights that have changed the values after optimization. Overall, 15.45% weights have changed their values. (b) The distribution of the absolute value change among these weights. (c) The quantization scales before and after optimization for the first 50 channels of the 16th convolutional layer.

the effect of the data source and the size of different calibration set on the performance of post-training quantization, which are widely ignored by previous post-training quantization works.

4.7.1 Calibration Data Source

The proposed Bit-split method needs some calibration images for quantization parameters optimization. We want to point out that these calibration images are not necessarily from the training data. Theoretically, any images can serve as the calibration set, however, different calibration sets can affect the performance of post-training quantization.

In this section, we evaluate the performance of post-training quantization with various calibration data sources. We use ResNet-18 and the proposed Bit-split framework with 8-bit activations and 4-bit uniform weights (A8W4) for evaluation. TF-Lite and TensorRT results are also given for comparison. The calibration sets for different application scenarios are listed as follows:

- *ImageNet-Train:* A small number of training images without annotations are available. Thus we use ImageNet training dataset for post-training quantization.
- *ImageNet-Test:* The training dataset is confidential, however, a small number of images that come from similar distribution as the training set are available. For this case, we use images sampled from the test set of ImageNet for simulation.
- *DeepInversion:* Due to privacy concerns, no real images are available. However, the model provider can provide some synthesized images. We use the images synthesized by DeepInversion [75] as the calibration set.
- *COCO-2017/VOC-2012:* Suppose no training images or synthesized images are available. Images of public datasets, which come from different distributions, can be used as the calibration set. We use COCO-2017 and VOC-2012 datasets for evaluation, which have some similarities with the ImageNet dataset.
- *CIFAR-10:* No training images or synthesized images are available. CIFAR-10 dataset is used for evaluation, which is very dissimilar to the ImageNet dataset.
- *Gaussian-Noise:* As a baseline, we also report results using random images drawn from Gaussian distribution for calibration.

TABLE 6
The Effect of Calibration Data Source of Post-Training Quantization on ResNet-18

Calibration Set	Bit-split		TF-Lite		TensorRT	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
ImageNet-Train	69.10	88.69	55.49	79.10	55.50	79.15
ImageNet-Test	69.02	88.63	55.49	79.18	55.45	79.18
DeepInv. [74]	68.36	88.07	55.49	79.17	55.55	79.14
COCO-2017	68.53	88.37	55.47	79.13	55.54	79.18
VOC-2012	68.48	88.25	55.51	79.20	55.51	79.15
CIFAR-10	62.53	84.38	55.53	79.14	55.46	79.09
Gaussian-Noise	9.73	21.98	54.22	78.31	54.10	78.24

A8W4 with uniform quantization is adopted for evaluation. We use per-channel scale factors for the weights and per-layer scale factors for the activations. All results in this table are obtained with 1024 calibration images.

The results with different calibration sets are summarized in Table 6. It is easy to find that the calibration sets would affect the performance of the Bit-split method. Table 6 also gives the results of TF-Lite and TensorRT. Different from Bit-split, TF-Lite and TensorRT are more robust to different calibration sets. This is because that the weight quantization of TF-Lite and TensorRT does not rely on the calibration images. Moreover, the Bit-split method consistently outperforms TF-Lite and TensorRT by large except for *Gaussian-Noise*, which we will discuss later.

For Bit-split, using the same training images (*ImageNet-Train*) as the pretrained model obtains the highest accuracy. Moreover, images from *ImageNet-Test* can be viewed as sampled from the same distribution of the training images of the pretrained model, thus *ImageNet-Test* results are almost the same as *ImageNet-Train*.

Surprisingly, the Bit-split method can still achieve high accuracy even without any real images. By using the synthesized images from *DeepInversion*, about 0.62% Top-5 accuracy degradation is observed compared with that using real training images. These results show that though the synthesized images are visually quite different from training images, they can still be used as the calibration set for network quantization. One drawback of *DeepInversion* is that the generating of the synthesized images requires the whole training dataset as well as the time-consuming Stochastic Gradient Descent (SGD) procedure.

Another finding from Table 6 is that the accuracy of using public datasets for calibration can be very different. It depends on the similarity between the calibration dataset and the training dataset of the pretrained models. Specifically, when the public calibration dataset is similar to the training set, the accuracy could be very close to that of using

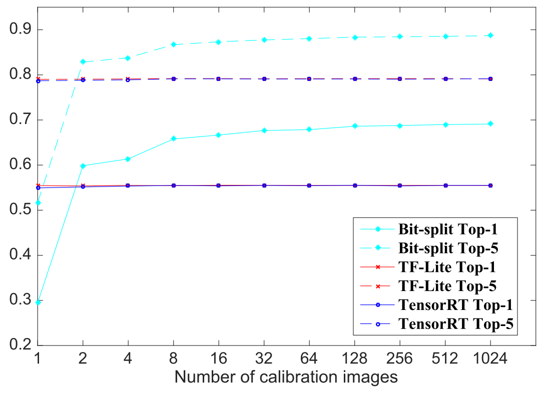


Fig. 7. The accuracy of post-training quantization on ResNet-18 with different numbers of images for calibration. A8W4 with uniform quantization is adopted for evaluation.

the original training set for calibration. For example, there is only 0.32% Top-5 accuracy degradation using *COCO-2017* for calibration. The results of *VOC-2012* are similar. However, when the public calibration dataset is very dissimilar to the training set, there will be a large accuracy drop after quantization. Take the *CIFAR-10* images for example, there is more than 4% accuracy gap with pretrained model. The reason is that the CIFAR-10 images are very different from ImageNet, especially that the image resolution of CIFAR-10 is much smaller than ImageNet. Thus it is inaccurate to use CIFAR-10 images for the quantization parameter selection of ImageNet pretrained models. In summary, if we have some prior information about the pretrained models, we can simply select a similar public dataset for calibration, and the results should be very close to that of using the original training dataset.

As a baseline, we also report results using random images drawn from Gaussian distribution for calibration (*Gaussian-Noise*). The Bit-split method with *Gaussian-Noise* has a large accuracy drop, only 9.73% Top-1 accuracy is achieved. By contrast, the TF-Lite and TensorRT are more robust even with random noise for calibration. These results are not surprising, because the weight quantization of TF-Lite and TensorRT does not rely on the calibration images. However, the Bit-split method, which learns the mapping between the inputs and the outputs, would overfit to the random noise.

4.7.2 Calibration Size

In this section, we explore another factor that can result in performance diversity of the proposed post-training quantization, i.e., the size of the calibration set. We use ResNet-18 and the proposed Bit-split framework with 8-bit activations and 4-bit uniform weights (A8W4) for evaluation. The numbers of images used for calibration are $\{2^i, i = 0, 1, \ldots, 10\}$. The results are presented in Fig. 7.

Fig. 7 shows that calibration size could affect the performance of post-training quantization. As the number of images increases, the accuracy improves. There is no noticeable accuracy drop with more than 100 images, which means that most of the classes are never seen by the model during the post-training optimization procedure. This result indicates that the proposed post-training framework learns a general low-bit feature representation, no over-fitting is observed. More surprisingly, with only 2 calibration images, Bit-split can already achieve very promising results.

In Fig. 7, we also plot the accuracy of TF-Lite and TensorRT for comparison. Not surprisingly, TF-Lite and TensorRT are robust to the calibration size, because these two approaches do not use calibration images for the optimization of the low-bit weights. With only one calibration image, Bit-split achieves much lower accuracy than TF-Lite and TensorRT. This is because that the low-bit weight optimization of Bit-split overfits to that calibration image. However, with more than one images, Bit-split can substantially outperform TF-Lite and TensorRT.

TABLE 7
Speed Comparison of ResNet-18 With Various Bit-Widths on FPGA

Bit-width	Execution time (ms)	Speedup ratio
A8W8	222.704	1×
A8W4	114.305	1.98×
A4W4	57.605	3.87×

4.8 Speed Estimation

The ultimate goal of neural network quantization is to reduce the storage and to speed up the inference. Thus in this section, we evaluate the speed efficiency of the quantized networks with various bit-widths on Field-Programmable Gate Array (FPGA). In this experiment, we implement the Bit Fusion [76], a state-of-the-art bit-scalable CNN accelerator on Xilinx VC709 FPGA. Bit Fusion employs a systolic array-based design, each Processing Element (PE) in the array is composed of 16 2-bit multipliers that can be fused to support for 2/4/8-bit multiplications. To highlight the hardware efficiency of quantization, we use double buffer scheme for all the on-chip buffers to overlap the transfer time by computation, leaving the other configurations of our implementation the same as the original ASIC design. All the experiments are conducted at 200 MHz.

At the inference stage, we merge all the quantization scales into the corresponding batch normalization layers following [67] to simplify the execution. Moreover, for consistency, we report the results with per-channel weight quantization and per-layer activation quantization. Note that in this setting, different quantization approaches including [15], [16], [52], [53] only vary in the optimization procedure, while the resulting network architectures at inference are the same. Thus the execution time is determined only by the bit-width settings. However, previous approaches only show acceptable accuracy with 8-bit quantization, while the proposed method can compress the network into 4-bit. Thus in this section, we use the A8W8 quantization as the baseline to show the speed advantages of the proposed method.

The speed results for one single image with input size 224×224 are reported in Table 7. We use ResNet-18 for demonstration. With 8-bit quantization, the execution time on the FPGA board is about 222.7 ms, which is a very strong baseline. When quantizing the weights into 4-bit (A8W4), Bit-split delivers 1.98× speedup compared to the baseline (A8W8), at the cost of about 0.7% accuracy degradation.

When the activations are further quantized into 4-bit (A4W4), the speed can be further improved, achieving about 3.87× speedup compared to the baseline. These results show that lowering the bit-width for activations and weights using the proposed Bit-split framework can significantly improve the inference efficiency.

5 Conclusion

In this work, we propose an efficient framework for lower-bit post-training quantization. We first theoretically show that the quantization loss in the final output layer is bounded by the layer-wise activation reconstruction error. Then based on the analysis, we propose an Optimization-based Post-training Quantization framework and a novel Bit-split optimization approach to simultaneously optimize the quantization scales as well as the low-bit weights, to achieve minimal accuracy degradation. Besides, we propose an Error Compensated Activation Quantization method to lower the quantization error for activations. The overall framework is effective, hyper-parameter free, and can be readily implemented and integrated into current network quantization libraries. Extensive experiments demonstrate that the proposed framework sets new state-of-the-arts for post-training quantization. Specifically, we achieve near-original model performance even when quantizing FP32 models to 3-bit without fine-tuning.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. 3rd Int. Conf. Learn. Representations*, 2015, pp. 1–14.

[3] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," in *Proc. Brit. Mach. Vis. Conf.*, 2014, pp. 1–12.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 580–587.

[6] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 91–99.

[7] W. Liu *et al.*, "SSD: Single shot MultiBox detector," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 21–37.

[8] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 3431–3440.

[9] Q. Chen, A. Cheng, X. He, P. Wang, and J. Cheng, "SpatialFlow: Bridging all tasks for panoptic segmentation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 6, pp. 2288–2300, Jun. 2021.

[10] J. Cheng, P.-S. Wang, G. Li, Q.-H. Hu, and H.-Q. Lu, "Recent advances in efficient computation of deep convolutional neural networks," *Front. Inf. Technol. Electron. Eng.*, vol. 19, no. 1, pp. 64–77, 2018.

[11] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[12] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 4820–4828.

[13] P. Wang and J. Cheng, "Accelerating convolutional neural networks for mobile applications," in *Proc. ACM Multimedia Conf.*, 2016, pp. 541–545.

[14] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "FitNets: Hints for thin deep nets," in *Proc. 3rd Int. Conf. Learn. Representations*, 2015, pp. 1–10.

[15] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018, *arXiv:1806.08342*.

[16] S. Migacz, "8-bit inference with TensorRT," in *Proc. GPU Technol. Conf.*, 2017, Art. no. 5.

[17] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, 2014, pp. 10–14.

[18] Y. Cai, Z. Yao, Z. Dong, A. Gholami, M. W. Mahoney, and K. Keutzer, "ZeroQ: A novel zero shot quantization framework," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 13166–13175.

[19] R. Banner, Y. Nahshan, and D. Soudry, "Post training 4-bit quantization of convolutional networks for rapid-deployment," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2019, pp. 7948–7956.

[20] P. Wang, Q. Chen, X. He, and J. Cheng, "Towards accurate post-training network quantization via bit-split and stitching," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 243–252.

[21] Y. Choukroun, E. Kravchik, F. Yang, and P. Kisilev, "Low-bit quantization of neural networks for efficient inference," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshop*, 2020, pp. 3009–3018.

[22] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *Proc. 32nd Int. Conf. Mach. Learn.*, 2015, pp. 2285–2294.

[23] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," 2014, *arXiv:1412.6115*.

[24] P. Stock, A. Joulin, R. Gribonval, B. Graham, and H. Jégou, "And the bit goes down: Revisiting the quantization of neural networks," in *Proc. 8th Int. Conf. Learn. Representations*, 2020, pp. 1–11.

[25] B. Zhuang, C. Shen, M. Tan, L. Liu, and I. Reid, "Structured binary neural networks for accurate image classification and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 413–422.

[26] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1737–1746.

[27] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2849–2858.

[28] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.

[29] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 4114–4122.

[30] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 525–542.

[31] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K.-T. Cheng, "Bi-real net: Enhancing the performance of 1-bit CNNs with improved representational capability and advanced training algorithm," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 722–737.

[32] C. Leng, Z. Dou, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with ADMM," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 3466–3473.

[33] P. Wang, X. He, G. Li, T. Zhao, and J. Cheng, "Sparsity-inducing binarized neural networks," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 12 192–12 199.

[34] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless CNNs with low-precision weights," in *Proc. 5th Int. Conf. Learn. Representations*, 2017, pp. 1–14.

[35] P. Wang, Q. Hu, Y. Zhang, C. Zhang, Y. Liu, and J. Cheng, "Two-step quantization for low-bit neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4376–4384.

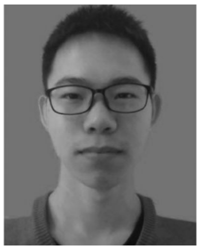
[36] J. Faraone, N. Fraser, M. Blott, and P. H. W. Leong, "SYQ: Learning symmetric quantization for efficient deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4300–4309.

[37] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," in *Proc. Int. Conf. Learn. Representations*, 2020, pp. 1–10.

[38] D. Zhang, J. Yang, D. Ye, and G. Hua, "LQ-Nets: Learned quantization for highly accurate and compact deep neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 365–382.

[39] S. Jung *et al.*, "Learning to quantize deep networks by optimizing quantization intervals with task loss," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4345–4354.
[40] M. Courbariaux, Y. Bengio, and J. David, "Low precision arithmetic for deep learning," in *Proc. 3rd Int. Conf. Learn. Representations Workshop*, 2015, pp. 1–10.
[41] H. Tann, S. Hashemi, R. I. Bahar, and S. Reda, "Hardware-software codesign of accurate, multiplier-free deep neural networks," in *Proc. 54th ACM/EDAC/IEEE Des. Autom. Conf.*, 2017, pp. 1–6.
[42] S. Vogel, M. Liang, A. Guntoro, W. Stechele, and G. Ascheid, "Efficient hardware acceleration of CNNs using logarithmic data representation with arbitrary log-base," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2018, pp. 1–8.
[43] P. Wang and J. Cheng, "Fixed-point factorized networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 3966–3974.
[44] Q. Hu, G. Li, P. Wang, Y. Zhang, and J. Cheng, "Training binary weight networks via semi-binary decomposition," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 637–653.
[45] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 344–352.
[46] S. Zhu, X. Dong, and H. Su, "Binary ensemble neural network: More bits per network or more networks per bit?" in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4918–4927.
[47] Z. Li, B. Ni, W. Zhang, X. Yang, and W. Gao, "Performance guaranteed network acceleration via high-order residual quantization," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2603–2611.
[48] W. Tang, G. Hua, and L. Wang, "How to train a compact binary neural network with high accuracy?," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 2625–2631.
[49] Z. Xie, Z. Wen, J. Liu, Z. Liu, X. Wu, and M. Tan, "Deep transferring quantization," in *Proc. 16th Eur. Conf. Comput. Vis.*, 2020, pp. 625–642.
[50] S. Xu *et al.*, "Generative low-bitwidth data free quantization," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 1–17.
[51] X. Zhang *et al.*, "Diversifying sample generation for accurate data-free quantization," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 15653–15662.
[52] Y. Nahshan *et al.*, "Loss aware post-training quantization," 2019, *arXiv:1911.07190*.
[53] M. Nagel, M. V. Baalen, T. Blankevoort, and M. Welling, "Data-free quantization through weight equalization and bias correction," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 1325–1334.
[54] J. Fang, A. Shafiee, H. Abdel-Aziz, D. Thorsley, G. Georgiadis, and J. H. Hassoun, "Post-training piecewise linear quantization for deep neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 69–86.
[55] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel, "Optimal brain damage," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 1989, pp. 598–605.
[56] B. Hassibi and D. G. Stork, *Second Order Derivatives for Network Pruning: Optimal Brain Surgeon*. San Mateo, CA, USA: Morgan Kaufmann, 1993.
[57] H. Peng, J. Wu, S. Chen, and J. Huang, "Collaborative channel pruning for deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 5113–5122.
[58] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "HAWQ: Hessian aware quantization of neural networks with mixed-precision," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 293–302.
[59] Z. Dong *et al.*, "Trace weighted hessian-aware quantization," in *Proc. Int. Conf. Neural Inf. Process. Syst. Workshops*, 2019, pp. 1–5.
[60] Z. Dong *et al.*, "HAWQ-V2: Hessian aware trace-weighted quantization of neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2020, pp. 18 518–18 529.
[61] V. Lebedev, Y. Ganin, M. Rakhuba, I. V. Oseledets, and V. S. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned CP-decomposition," in *Proc. Int. Conf. Learn. Representations*, 2015, pp. 1–10.
[62] Q. Hu, P. Wang, and J. Cheng, "From hashing to CNNs: Training binary weight networks via hashing," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 3247–3254.
[63] Z. Cai, X. He, J. Sun, and N. Vasconcelos, "Deep learning with low precision by half-wave gaussian quantization," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 5406–5414.
[64] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 4857–4867.
[65] M. Nagel, R. A. Amjad, M. Van Baalen, C. Louizos, and T. Blankevoort, "Up or down? Adaptive rounding for post-training quantization," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 7197–7206.
[66] S. Vogel, J. Springer, A. Guntoro, and G. Ascheid, "Self-supervised quantization of pre-trained neural networks for multiplierless acceleration," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, 2019, pp. 1094–1099.
[67] F. Li *et al.*, "A system-level solution for low-power object detection," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. Workshops*, 2019, pp. 2461–2468.
[68] G. Li, P. Wang, Z. Liu, C. Leng, and J. Cheng, "Hardware acceleration of CNN with one-hot quantization of weights and activations," in *Proc. Des. Autom. Test Eur. Conf. Exhib.*, 2020, pp. 971–974.
[69] J. Choi, B. Y. Kong, and I.-C. Park, "Retrain-less weight quantization for multiplier-less convolutional neural networks," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 3, pp. 972–982, Mar. 2020.
[70] F. Shen, C. Shen, W. Liu, and H. Tao Shen , "Supervised discrete hashing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 37–45.
[71] K. Chellapilla, S. Puri, and P. Simard, "High performance convolutional neural networks for document processing," in *Proc. 10th Int. Workshop Front. Handwriting Recognit.*, 2006, pp. 1–6.
[72] M. Haroush, I. Hubara, E. Hoffer, and D. Soudry, "The knowledge within: Methods for data-free model compression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 8491–8499.
[73] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2999–3007.
[74] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2980–2988.
[75] H. Yin *et al.*, "Dreaming to distill: Data-free knowledge transfer via deepinversion," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 8712–8721.
[76] H. Sharma *et al.*, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit.*, 2018, pp. 764–775.

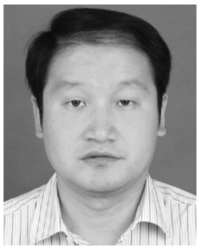
Peisong Wang received the BE degree in software engineering from Shandong University, Jinan, China, in 2013, and the PhD degree in computer science from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2018. He is currently an associate researcher with the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing, China. His current research interests include computer vision, network acceleration, and compression.



Weihan Chen received the BE degree in communication engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 2018. He is currently working toward the PhD degree in the Institute of Automation, Chinese Academy of Sciences, Beijing, China. His current research interests include deep learning, model compression, and acceleration.

Xiangyu He received the BE degree in information security from the Beijing University of Posts and Telecommunications, Beijing, China, in 2017. He is currently working toward the PhD degree in the Institute of Automation, Chinese Academy of Sciences, Beijing, China. His current research interests include deep learning, image retrieval, and high performance computing.

Qiang Chen received the BS degree from Beihang University, Beijing, China, in 2016. He is currently working toward the doctorate degree in the Institute of Automation, Chinese Academy of Sciences, Beijing, China. His current research interests include object detection and image segmentation.



Qingshan Liu (Senior Member, IEEE) received the PhD degree from the National Laboratory of Pattern Recognition (NLPR), Chinese Academy of Sciences, Beijing, China, in 2003, and then he worked with NLPR after graduation. From April, 2006 to August, 2011, he worked with the Computational Biomedicine Imaging and Modeling Center, Department of Computer Science, Rutgers University, New Brunswick, NJ, USA. He is currently a professor with B-Data Laboratory, Nanjing University of Information Science and Technology, Nanjing. His current research interests include image and vision analysis, computer vision, and pattern recognition.

Jian Cheng (Member, IEEE) received the BS and MS degrees in mathematics from Wuhan University, Wuhan, China, in 1998 and 2001, respectively, and the PhD degree in pattern recognition and intelligent systems from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2004. He is currently a professor with the Institute of Automation, Chinese Academy of Sciences. His major research interests include deep learning, computer vision, chip design, etc.