DIFFERENTIABLE DYNAMIC QUANTIZATION WITH MIXED PRECISION AND ADAPTIVE RESOLUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Model quantization is to discretize weights and activations of a deep neural network (DNN). Unlike previous methods that manually defined the quantization hyperparameters such as precision (bitwidth), dynamic range (minimum and maximum discrete values) and stepsize (interval between discrete values), this work proposes a novel differentiable approach, named Differentiable Dynamic Quantization (DDQ), to automatically learn all of them. It possesses several appealing benefits. (1) Unlike previous works that applied the rounding operation to discretize values, DDQ provides a unified perspective by formulating discretization as a matrix-vector product, where different values of the matrix and vector represent different quantization methods such as mixed precision and soft quantization, and their values can be learned differentiably making different hidden layers in a DNN used different quantization methods. (2) DDQ is hardware-friendly and can be easily implemented using a low-precision matrix-vector multiplication, making it naturally capable in wide spectrum of hardwares. (3) The matrix variable in DDQ is carefully reparameterized to reduce the number of parameters from O(2^{2b}) to O($\log 2^b$), where b is the bit width. Extensive experiments show that DDQ outperforms prior arts on various advanced networks and benchmarks. For instance, compared to the full-precision models, MobileNetv2 trained with DDQ achieves comparable top1 accuracy on ImageNet (71.7% vs 71.9%), while ResNet18 trained with DDQ increases accuracy by 0.5%. These results relatively improve recent state-of-the-art quantization methods by 70% and 140% compared to the full-precision models.

1 INTRODUCTION

Deep Neural Networks (DNNs) have made significant progress in many applications. However, the large memory and computations impede the mass deployment of DNNs such as on portable devices. Model quantization (Courbariaux et al., 2015; 2016; Zhu et al., 2017) that discretizes the weights and activations of a DNN to reduce its resource consumption becomes an important topic, but it is challenging because of two aspects. Firstly, different DNN architectures allocate different memory and computational complexity in different layers, making quantization suboptimal when the quantization parameters such as bitwidth, dynamic range and stepsize are freezed in each layer. Secondly, gradient-based training of quantized DNNs is difficult (Bengio et al., 2013), because the gradient of previous quantization function may vanish, *i.e.* backpropagation through a quantized DNN may return zero gradients.

Previous quantization approaches typically used the round operation. They can be summarized below. Let x and x_q be values before and after quantization, we have $x_q = \text{sign}(x) \cdot d \cdot \mathcal{F}(\lfloor |x|/d + 0.5 \rceil)$, where $|\cdot|$ denotes the absolute value, $\text{sign}(\cdot)$ returns the sign of x, d is the stepsize (*i.e.* the interval between two adjacent discrete values after quantization), and $\lfloor \cdot \rceil$ denotes the round function[1]. Moreover, $\mathcal{F}(\cdot)$ is a function that maps a rounded value to a desired quantization level (*i.e.* a desired discrete value). For instance, the above equation would represent a uniform quantizer[2] when $\mathcal{F}$ is an identity mapping, or it would represent a power-of-two (nonuniform) quantizer (Miyashita et al., 2016; Zhou et al., 2017; Liss et al., 2018; Zhang et al., 2018a) when $\mathcal{F}$ is a function of power of two.

[1] A function returns the closest discrete value given a continuous value.

[2] A uniform quantizer has uniform quantization levels, which means the stepsizes between any two adjacent discrete values are the same.

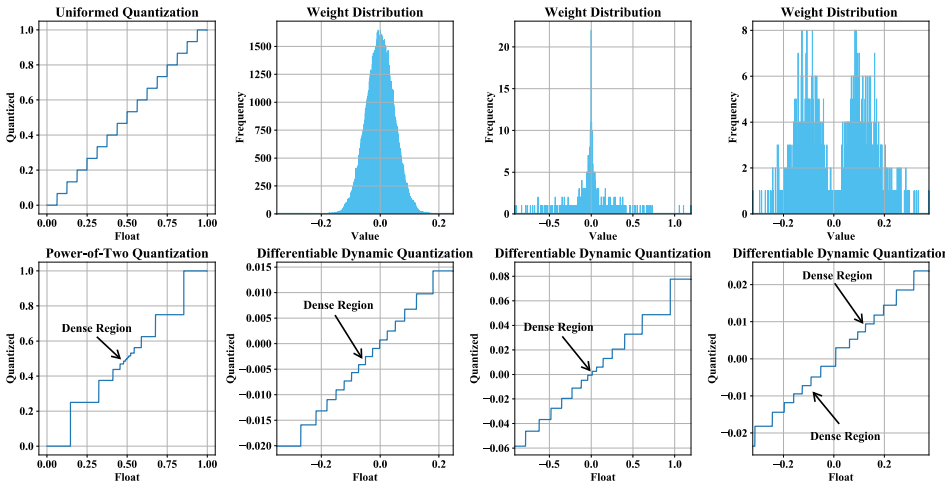


Figure 1: **Compare** methods in 4-bit low-precision quantization. **(a)** compares quantization levels between the uniform and nonuniform (power-of-two) (Miyashita et al., 2016; Zhou et al., 2017; Liss et al., 2018; Zhang et al., 2018a) quantizers, where x- and y-axis represent values before and after quantization respectively (the float values are scaled between 0 and 1 for illustration). We highlight the dense region with higher "resolution" by arrows. We see that there is no dense region in uniform quantization because the intervals between levels are the same, while a single dense region in power-of-two quantization. **(b)**, **(c)** and **(d)** show the weight distributions of different layers of a MobileNetv2 (Sandler et al., 2018) trained on ImageNet (Russakovsky et al., 2015), and the corresponding quantization levels learned by the proposed DDQ. We see that the weight distributions are Gaussian-like in (b), heavy-tailed in (c), and two-peak bell-shaped in (d). DDQ enables learning arbitrary quantization levels with different number of dense regions to model these distributions.

Although quantization using the round function is straightforward, we often see that the quantized values in a low-precision DNN after rounding have quantization error in each layer, *i.e.* large $\|x - x_q\|_2$, which significantly decreases the model's accuracy compared to its full-precision counterpart even after retraining network weights. To improve the accuracy of the quantized network, recent quantizers were proposed to reduce $||x - x_q||_2$. For example, TensorRT (Migacz, 2017), FAQ (McKinstry et al., 2018), PACT (Choi, 2018) and TQT (Jain et al., 2019) introduced and optimized an additional parameter that represents the dynamic range to calibrate the quantization levels, in order to better fit the distributions of the full-precision network (*i.e.* reduce the shift between x_q and x). Besides, prior arts (Miyashita et al., 2016; Zhou et al., 2017; Liss et al., 2018; Zhang et al., 2018a) also adopted non-uniform quantization levels to discretize values into a set of discrete numbers with different stepsizes, which could better capture the original distributions.

6 a better quantizer, most of the above approach are focused on reducing the shift of values after and before quantization, by learning or carefully designing several distribution parameters of the quantizer.

Despite the above quantizers reduce certain shift between values before and after quantization, they are achieved by using constraints (or assumptions) that are not sufficient to quantize recent DNNs, limiting their generalization performance. For example, one main assumption is that the network weights follow bell-shaped distributions. However, we find that this is not always plausible in many common architectures such as ResNet (He et al., 2016), Inception (Szegedy et al., 2015), MobileNet (Sandler et al., 2018; Howard et al., 2017) and ShuffleNet (Zhang et al., 2018b; Ma et al., 2018). For instance, Fig.1(b-d) plot different weight distributions of a MobileNetv2 (Sandler et al., 2018) trained on ImageNet, which is a representative lightweight architecture deploying on embedded devices. We find that these distributions have irregular forms in different feature channels, especially when depthwise or group convolution (Xie et al., 2017; Huang et al., 2018; Zhang et al., 2017) are used to improve computational efficiency. Although this problem has been identified in (Krishnamoorthi, 2018; Jain et al., 2019; Goncharenko et al., 2019), showing that per-channel/per-tensor scaling or bias correction could compensate some of the problems, however none of them could bridge the gap and maintain accuracy of the low-precision (*e.g.* 4-bit) DNNs compared to their full-precision counterparts. A full summary and comparisons with previous work are provided in Appendix A.3.

To address the above issue, this work proposes Differentiable Dynamic Quantization (DDQ), which automatically learns all the quantization parameters including arbitrary bitwidths, quantization levels, and dynamic ranges for different layers in a DNN in a differentiable manner. DDQ has appealing benefits that prior arts may not have and makes the below **contributions**. (1) Instead of using round function, DDQ presents a novel perspective by formulating quantization as matrix-vector product in a unified framework, where different values of the matrix and vector represent different quantization approaches, such as mixed-precision and soft quantization[3]. The quantization parameters in DDQ are fully trainable in different layers of a DNN and updated together with the network weights, making it generalizable to different network architectures and datasets. (2) DDQ is hardware-friendly and can be easily implemented using a low-precision matrix-vector multiplication (GEMM), making it capable in wide spectrum of hardwares. Moreover, a matrix reparameterization method is devised to reduce the matrix variable in DDQ from $O(2^{2b})$ to $O(\log 2^b)$, where b is the number of bits. (3) Extensive experiments show that DDQ outperforms prior arts on various advanced networks such as ResNet and MobileNet, as well as benchmarks such as ImageNet and CIFAR10. For instance, compared to the full-precision models, MobileNetv2 trained with 4-bit DDQ achieves comparable top1 accuracy on ImageNet (71.7% versus 71.9%), while ResNet18 trained with DDQ improves accuracy by 0.5%. These results relatively improve the recent state-of-the-art quantizers by 70% and 140% compared to the full-precision counterparts.

2 Our Approach

2.1 Preliminary and Notations

In network quantization, each continuous value $x \in \mathbb{R}$ is discretized to x_q, which is an element from a set of discrete values. This set is denoted as a vector $\boldsymbol{q} = [q_1, q_2, \cdots, q_n]^\mathsf{T}$ (termed quantization levels). We have $x_q \in \boldsymbol{q}$ and $n = 2^b$ where b is the bitwidth. Existing methods often represent $\boldsymbol{q}$ using uniform or powers-of-two distribution introduced below.

Uniform Quantization. The quantization levels $\boldsymbol{q}_u$ of a symmetric b-bit uniform quantizer (Uhlich et al., 2020) is

$$\boldsymbol{q}_u(\boldsymbol{\theta}) = \left[-1, \cdots, \frac{-2}{2^{b-1}-1}, \frac{-1}{2^{b-1}-1}, -0, +0, \frac{1}{2^{b-1}-1}, \frac{2}{2^{b-1}-1}, \cdots, 1\right]^\mathsf{T} \times c + \bar{x}, \quad (1)$$

where $\boldsymbol{\theta} = \{b, c\}^\mathsf{T}$ denotes a set of quantization parameters, b is the bitwidth, c is the clipping threshold, which represents a symmetric dynamic range[4], and $\bar{x}$ is a constant scalar (a bias) used to shift $\boldsymbol{q}_u$[5]. For example, $\boldsymbol{q}_u(\boldsymbol{\theta})$ is shown in the upper plot of Fig.1(a) when $c = 0.5$ and $\bar{x} = 0.5$. Although uniform quantization could be simple and effective, it assumes the weight distribution is uniform that is implausible in many recent DNNs.

Powers-of-Two Quantization. The quantization levels $\boldsymbol{q}_p$ of a symmetric b-bit powers-of-two quantizer (Miyashita et al., 2016; Liss et al., 2018) is

$$\boldsymbol{q}_p(\boldsymbol{\theta}) = \left[-2^{-1}, \cdots, -2^{-2^{b-1}+1}, -0, +0, 2^{-2^{b-1}+1}, \cdots, 2^{-1}\right]^\mathsf{T} \times c + \bar{x}. \quad (2)$$

As shown in the bottom plot of Fig.1(a) when $c = 1$ and $\bar{x} = 0.5$, $\boldsymbol{q}_p(\boldsymbol{\theta})$ has a single dense region that may capture a single-peak bell-shaped weight distribution.

In Eqn.(1-2), both uniform and power-of-two quantizers would fix $\boldsymbol{q}$ and optimize $\boldsymbol{\theta}$, which contains the clipping threshold c and the stepsize denoted as $d = 1/(2^b - 1)$ (Uhlich et al., 2020). Although they learn the dynamic range and stepsize, they have an obvious drawback, that is, the predefined quantization levels cannot fit varied distributions of weights or activations during training.

2.2 Dynamic Differentiable Quantization (DDQ)

Formulation. Instead of freezing the quantization levels, DDQ learns all quantization hyperparameters. Let $Q(x; \boldsymbol{\theta})$ be a function with a set of parameters $\boldsymbol{\theta}$ and $x_q = Q(x; \boldsymbol{\theta})$ turns a continuous value

[3]Each continuous value could be discretized to different discrete numbers (*i.e.* quantization levels) in order to ease optimization (Louizos et al., 2018).

[4]In Eqn.(1), the dynamic range is $[-c, c]$.

[5]Note that '0' appears twice in order to assure that $\boldsymbol{q}_u$ is of size 2^b.

x into an element of $\boldsymbol{q}$ denoted as $x_q \in \boldsymbol{q}$, where $\boldsymbol{q}$ is initialized as a uniform quantizer and can be implemented in low-precision values according to hardware's requirement. DDQ is formulated by low-precision matrix-vector product,

$$x_q = Q(x;\boldsymbol{\theta}) = \boldsymbol{q}^\intercal \frac{\boldsymbol{U}}{Z_U}\boldsymbol{x}_o, \text{ where } x_o^i = \begin{cases} 1 & \text{if } i = \text{argmin}_j |\frac{1}{Z_U}(\boldsymbol{U}^\intercal \boldsymbol{q})_j - x| \\ 0 & \text{otherwise} \end{cases}, \tag{3}$$

where $x_o^i \in \boldsymbol{x}_o$, $\boldsymbol{x}_o \in \{0,1\}^{n\times 1}$ denotes a binary vector that has only one entry of '1' while others are '0', in order to select one quantization level from $\boldsymbol{q}$ for the continuous value x. Eqn.(3) has parameters $\boldsymbol{\theta} = \{\boldsymbol{q}, \boldsymbol{U}\}$, which are trainable by stochastic gradient descent (SGD), making DDQ automatically capture weight distributions of the full-precision models as shown in Fig.1(b-d). Here $\boldsymbol{U} \in \{0,1\}^{n\times n}$ is a binary block-diagonal matrix and Z_U is a constant normalizing factor used to average the discrete values in $\boldsymbol{q}$ in order to learn bitwidth. Intuitively, different values of $\boldsymbol{x}_o$, $\boldsymbol{U}$ and $\boldsymbol{q}$ make DDQ represent different quantization approaches as discussed below. To ease understanding, Fig.2(a) compares the computational graph of DDQ with the rounding-based methods. We see that DDQ learns the entire quantization levels instead of just the stepsize d as prior arts did.

Discussions of Representation Capacity. DDQ represents a wide range of quantization methods. For example, when $\boldsymbol{q} = \boldsymbol{q}_u$ (Eqn.(1)), $Z_U = 1$, and $\boldsymbol{U} = \boldsymbol{I}$ where $\boldsymbol{I}$ is an identity matrix, Eqn.(3) represents an ordinary uniform quantizer. When $\boldsymbol{q} = \boldsymbol{q}_p$ (Eqn.(2)), $Z_U = 1$, and $\boldsymbol{U} = \boldsymbol{I}$, Eqn.(3) becomes a power-of-two quantizer. When $\boldsymbol{q}$ is learned, it represents arbitrary quantization levels with different dense regions.

Moreover, DDQ enables mixed precision training when $\boldsymbol{U}$ is block-diagonal. For example, as shown in Fig.2(b), when $\boldsymbol{q}$ has length of 8 entries (*i.e.* 3-bit), $Z_U = \frac{1}{2}$, and $\boldsymbol{U} = \text{Diag}\left(\mathbf{1}_{2\times 2}, \cdots, \mathbf{1}_{2\times 2}\right)$, where $\text{Diag}(\cdot)$ returns a matrix with the desired diagonal blocks and its off-diagonal blocks are zeros and $\mathbf{1}_{2\times 2}$ denotes a 2-by-2 matrix of ones, $\boldsymbol{U}$ enables Eqn.(3) to represent a 2-bit quantizer by averaging neighboring discrete values in $\boldsymbol{q}$. For another example, when $\boldsymbol{U} = \text{Diag}\left(\mathbf{1}_{4\times 4}, \mathbf{1}_{4\times 4}\right)$ and $Z_U = \frac{1}{4}$, Eqn.(3) turns into a 1-bit quantizer. Besides, when $\boldsymbol{x}_o$ is a soft one-hot vector with multiple non-zero entries, Eqn.(3) represents soft quantization that one continuous value can be mapped to multiple discrete values.

Efficient Inference on Hardware. DDQ is a unified quantizer that supports adaptive $\boldsymbol{q}$ as well as predefined ones *e.g.* uniform and power-of-two. It is friendly to hardware with limited resources. As shown in Eqn.(3), DDQ reduces to a uniform quantizer when $\boldsymbol{q}$ is uniform. In this case, DDQ can be efficiently computed by a rounding function as the step size is determined by $\boldsymbol{U}$ after training (*i.e.* don't have $\boldsymbol{U}$ and matrix-vector product when deploying in hardware like other uniform quantizers). In addition, DDQ with adaptive $\boldsymbol{q}$ can be implemented using low-precision general matrix multiply (GEMM). For example, let y be a neuron's activation, $y = Q(w;\boldsymbol{\theta})x_q = \boldsymbol{q}^\intercal \frac{\boldsymbol{U}}{Z_U}\boldsymbol{w}_o x_q$, where x_q is a discretized feature value, w is a continuous weight parameter to be quantized, and $\boldsymbol{U}$ and $\boldsymbol{w}_o$ are binary matrix and one-hot vector of w respectively. To accelerate, we can calculate the major part $\frac{\boldsymbol{U}}{Z_U}\boldsymbol{w}_o x_q$ using low-precision GEMM first and then multiplying a short 1-d vector $\boldsymbol{q}$, which is shared for all convolutional weight parameters and can be float32, float16 or INT8 given specific hardware requirement. The latency in hardware is compared in Appendix A.4.2.

2.3 Matrix Reparameterization of U

In Eqn.(3), $\boldsymbol{U}$ is a learnable matrix variable, which is challenging to optimize in two aspects. First, to make DDQ a valid quantizer, $\boldsymbol{U}$ should have binary block-diagonal structure, which is difficult to learn by using SGD. Second, the size of $\boldsymbol{U}$ (number of parameters) increases when the bitwidth increases *i.e.* 2^{2b}. Therefore, rather than directly optimize $\boldsymbol{U}$ in the backward propagation using SGD, we explicitly construct $\boldsymbol{U}$ by composing a sequence of small matrices in the forward propagation (Luo et al., 2019).

A Kronecker Composition for Quantization. The matrix $\boldsymbol{U}$ can be reparameterized to reduce number of parameters from 2^{2b} to $\log 2^b$ to ease optimization. Let $\{\boldsymbol{U}_1, \boldsymbol{U}_2, \cdots, \boldsymbol{U}_b\}$ denote a set of b small matrices of size 2-by-2, $\boldsymbol{U}$ can be constructed by $\boldsymbol{U} = \boldsymbol{U}_1 \otimes \boldsymbol{U}_2 \otimes \cdots \otimes \boldsymbol{U}_b$, where $\otimes$ denotes Kronecker product and each $\boldsymbol{U}_i$ ($i = 1...b$) is either a 2-by-2 identity matrix (denoted as $\boldsymbol{I}$) or an all-one matrix (denoted as $\mathbf{1}_{2\times 2}$), making $\boldsymbol{U}$ block-diagonal after composition. For instance, when $b = 3$, $\boldsymbol{U}_1 = \boldsymbol{U}_2 = \boldsymbol{I}$ and $\boldsymbol{U}_3 = \mathbf{1}_{2\times 2}$, we have $\boldsymbol{U} = \text{Diag}\left(\mathbf{1}_{2\times 2}, \cdots, \mathbf{1}_{2\times 2}\right)$ and Eqn.(3) represents a 2-bit quantizer as shown in Fig.2(b).

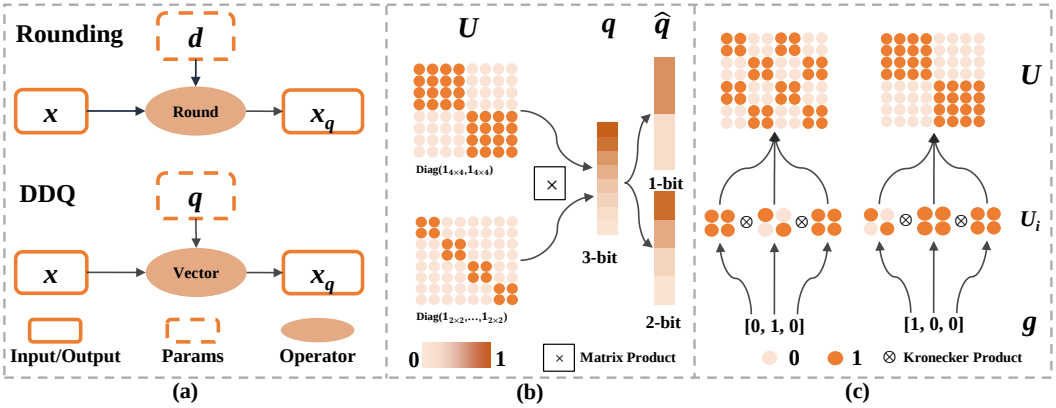


Figure 2: **Illustrations of DDQ. (a)** compares computations of DDQ with the round operator. Unlike rounding methods that only learn the stepsize d, DDQ treats $\boldsymbol{q}$ as trainable variable, learning arbitrary quantization levels. **(b)** illustrates that DDQ enables mixed-precision training by using different binary block-diagonal matrix $\boldsymbol{U}$. The circles in light and dark indicate '0' and '1' respectively. For example, when $\boldsymbol{q}$ is of length 8 entries (*i.e.* 3-bit) and $\boldsymbol{U} = \text{Diag}\left(\mathbf{1}_{2\times2}, \cdots, \mathbf{1}_{2\times2}\right)$, where $\text{Diag}(\cdot)$ returns a matrix with the desired diagonal blocks while the off-diagonal blocks are zeros and $\mathbf{1}_{2\times2}$ denotes a 2-by-2 matrix of ones, we have $\hat{\boldsymbol{q}} = \boldsymbol{U}^\top \boldsymbol{q}$ that enables DDQ to represent a 2-bit quantizer by averaging neighboring discrete values in $\boldsymbol{q}$. Another example is a 1-bit quantizer when $\boldsymbol{U} = \text{Diag}\left(\mathbf{1}_{4\times4}, \mathbf{1}_{4\times4}\right)$. **(c)** shows relationship between gating variables $\boldsymbol{g} = \{g_i\}_{i=1}^b$ and $\boldsymbol{U}$. For example, when the entries of $\boldsymbol{g} = [1, 0, 0]$ are arranged in an descending order and let $s = \sum_{i=1}^b g_i = 1 + 0 + 0 = 1$, $\boldsymbol{U}$ has $2^s = 2^1 = 2$ number of all-one diagonal blocks. In such case, DDQ is a $s = 1$ bit quantizer.

To pursue a more parameter-saving composition, we further parameterize each $\boldsymbol{U}_i$ by using a single trainable variable. As shown in Fig.2(c), we have $\boldsymbol{U}_i = g_i \boldsymbol{I} + (1 - g_i)\mathbf{1}_{2\times2}$, where $g_i = H(\hat{g}_i)$ and $H(\cdot)$ is a Heaviside step function[6], *i.e.* $g_i = 1$ when $\hat{g}_i \geq 0$; otherwise $g_i = 0$. Here $\{g_i\}_{i=1}^b$ is a set of gating variables with binary values. Intuitively, each $\boldsymbol{U}_i$ switches between a 2-by-2 identity matrix and a 2-by-2 all-one matrix.

In other words, $\boldsymbol{U}$ can be constructed by applying a series of Kronecker products involving only $\mathbf{1}_{2\times2}$ and $\boldsymbol{I}$. Instead of updating the entire matrix $\boldsymbol{U}$, it can be learned by only a few variables $\{\hat{g}_i\}_{i=1}^b$, significantly reducing the number of parameters from $2^b \times 2^b = 2^{2b}$ to b. In summary, the parameter size to learn $\boldsymbol{U}$ is merely the number of bits. With Kronecker composition, the quantization parameters of DDQ is $\boldsymbol{\theta} = \{\boldsymbol{q}, \{\hat{g}_i\}_{i=1}^b\}$, which could be different for different layers or kernels (*i.e.* layer-wise or channel-wise quantization) and the parameter size is negligible compared to the network weights, making different layers or kernels have different quantization levels and bithwidth.

Discussions of Relationship between $\boldsymbol{U}$ and $\boldsymbol{g}$. Let $\boldsymbol{g}$ denote a vector of gates $[g_1, \cdots, g_b]^\top$. In general, different values of $\boldsymbol{g}$ represent different block-diagonal structures of $\boldsymbol{U}$ in two aspects. (1) **Permutation.** As shown in Fig.2(c), $\{g_i\}_{i=1}^b$ should be permuted in an descending order by using a permutation matrix. Otherwise, $\boldsymbol{U}$ is not block-diagonal when $\boldsymbol{g}$ is not ordered, making DDQ an invalid quantizer. For example, $\boldsymbol{g} = [0, 1, 0]$ is not ordered compared to $\boldsymbol{g} = [1, 0, 0]$. (2) **Sum of Gates.** Let $s = \sum_{i=1}^b g_i$ be the sum of gates and $0 \leq s \leq b$. We see that $\boldsymbol{U}$ is a block-diagonal matrix with 2^s diagonal blocks, implying that $\boldsymbol{U}^\top \boldsymbol{q}$ has 2^s different discrete values and represents a s-bit quantizer. For instance, as shown in Fig.2(b,c) when $b = 3$, $\boldsymbol{g} = [1, 0, 0]^\top$ and $\boldsymbol{U} = \text{Diag}\left(\mathbf{1}_{4\times4}, \mathbf{1}_{4\times4}\right)$, we have a $s = 1 + 0 + 0 = 1$ bit quantizer. DDQ enables to regularize the value of s in each layer given memory constraint, such that optimal bitwidth can be assigned to different layers of a DNN.

3 Training with DDQ

DDQ is used to train a DNN with mixed precision to satisfy memory constraints, which reduce the memory to store the network weights and activations, making a DNN appealing for deployment in embedded devices.

[6]The Heaviside step function returns '0' for negative arguments and '1' for positive arguments.

3.1 DNN WITH MEMORY CONSTRAINT

Consider a DNN with L layers trained using DDQ, the forward propagation of each layer can be written as

$$\boldsymbol{y}^l = F\big(Q(\boldsymbol{W}^l;\boldsymbol{\theta}^l) * Q(\boldsymbol{y}^{l-1}) + Q(\boldsymbol{b}^l;\boldsymbol{\theta}^l)\big),\ l = 1, 2, \cdots, L \tag{4}$$

where $*$ denotes convolution, $\boldsymbol{y}^l$ and $\boldsymbol{y}^{l-1}$ are the output and input of the l-th layer respectively, F is a non-linear activation function such as ReLU, and Q is the quantization function of DDQ. Let $\boldsymbol{W}^l \in \mathbb{R}^{C^l_{out}\times C^l_{in}\times K^l\times K^l}$ and $\boldsymbol{b}^l$ denote the convolutional kernel and bias vector (network weights), where C_{out}, C_{in}, and K are the output and input channel size, and the kernel size respectively. Remind that in DDQ, $\{g^l_i\}^b_{i=1}$ is a set of gates at the l-th layer and the bitwidth can be computed by $s^l = \sum_{i=1}^b g^l_i$. For example, the total memory footprint (denoted as ζ) can be computed by

$$\zeta(s^1, \cdots, s^L) = \sum_{l=1}^{L} C^l_{out} C^l_{in} (K^l)^2 2^{s^l}, \tag{5}$$

which represents the memory to store all network weights at the l-th layer when the bitwidth is s^l.

If the desired memory is $\zeta(b^1, \cdots, b^L)$, we could use a weighted product to approximate the Pareto optimal solutions to train a b-bit DNN. The loss function is

$$\min_{\boldsymbol{W}^l, \boldsymbol{\theta}^l} \quad \mathcal{L}(\{\boldsymbol{W}^l\}^L_{l=1}, \{\boldsymbol{\theta}^l\}^L_{l=1}) \cdot \left(\frac{\zeta(b^1, \cdots, b^L)}{\zeta(s^1, \cdots, s^L)}\right)^{\alpha} \quad \text{s.t. } \zeta(s^1, \cdots, s^L) \le \zeta(b^1, \cdots, b^L), \tag{6}$$

where the loss $\mathcal{L}(\cdot)$ is reweighted by the ratio between the desired memory and the practical memory similar to (Tan et al., 2019; Deb, 2014). α is a hyper-parameter. We have $\alpha = 0$ when the memory constraint is satisfied. Otherwise, $\alpha < 0$ is used to penalize the memory consumption of the network.

3.2 UPDATING QUANTIZATION PARAMETERS

All parameters of DDQ can be optimized by using SGD. This section derives their update rules. We omit the superscript 'l' for simplicity.

Gradients *w.r.t.* $\boldsymbol{q}$. To update $\boldsymbol{q}$, we reparameterize $\boldsymbol{q}$ by a trainable vector $\tilde{\boldsymbol{q}}$, such that $\boldsymbol{q} = R(\tilde{\boldsymbol{q}})(x_{max} - x_{min})/(2^b - 1) + x_{min}$, in order to make each quantization level lies in $[x_{min}, x_{max}]$, where x_{max} and x_{min} are the maximum and minimum continuous values of a layer, and $R()$ denotes a uniform quantization function transforming $\tilde{q}$ to given b_q bits ($b_q < b$). Let $\boldsymbol{x}$ and $\boldsymbol{x}_q$ be two vectors stacking values before and after quantization respectively (*i.e.* x and x_q), the gradient of the loss function with respect to each entry q_k of $\boldsymbol{q}$ is given by

$$\frac{\partial \mathcal{L}}{\partial q_k} = \sum_{i=1}^{N} \frac{\partial \mathcal{L}}{\partial x^i_q} \frac{\partial x^i_q}{\partial q_k} = \frac{1}{Z_U} \sum_{i \in \mathcal{S}_k} \frac{\partial \mathcal{L}}{\partial x^i_q}, \tag{7}$$

where $x^i_q = Q(x^i;\boldsymbol{\theta})$ is the output of DDQ quantizer and $\mathcal{S}_k$ represents a set of indexes of the values discretized to the corresponding quantization level q_k. In Eqn.(7), we see that the gradient with respect to the quantization level q_k is the summation of the gradients $\frac{\partial \mathcal{L}}{\partial x^i_q}$. In other words, the quantization level in denser regions would have larger gradients. The gradient *w.r.t.* gate variables $\{g_i\}^b_{i=1}$ are discussed in Appendix A.1.

Gradient Correction for $\boldsymbol{x}_q$. In order to reduce the quantization error $\|\boldsymbol{x}_q - \boldsymbol{x}\|^2_2$, a gradient correction term is proposed to regularize the gradient with respect to the quantized values,

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} \leftarrow \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_q}, \quad \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_q} \leftarrow \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_q} + \lambda(\boldsymbol{x}_q - \boldsymbol{x}), \tag{8}$$

where the first equation holds by applying STE. In Eqn.(8), we first assign the gradient of $\boldsymbol{x}_q$ to that of $\boldsymbol{x}$ and then add a correction term $\lambda(\boldsymbol{x}_q - \boldsymbol{x})$. In this way, the corrected gradient can be back-propagated to the quantization parameters $\boldsymbol{q}$ and $\{g_i\}^b_{i=1}$ in Eqn.(7) and (14), while not affecting the gradient of $\boldsymbol{x}$. Intuitively, this gradient correction term is effective and can be deemed as the ℓ_2 penalty on $\|\boldsymbol{x} - \boldsymbol{x}_q\|^2_2$. Please note that this is not equivalent to simply impose a ℓ_2 regularization directly on the loss function, which would have no effect when STE is presented.

		MobileNetV2			ResNet18		
	Training Epochs	Model Size	Bitwidth (W/A)	Top-1 Accuracy	Model Size	Bitwidth (W/A)	Top-1 Accuracy
Full Precision	120	13.2 MB	32	71.9	44.6 MB	32	70.5
DDQ (ours)	30	1.8 MB	4 / 8 (mixed)	**71.8**	5.8 MB	4 / 8 (mixed)	**71.0**
DDQ (ours)	90	1.8 MB	4 / 8 (mixed)	**71.9**	5.8 MB	4 / 8 (mixed)	**71.2**
Deep Compression (Han et al., 2016)	-	1.8 MB	4 / 32	71.2	-	-	-
HMQ (Habi et al., 2020)	50	1.7 MB	4 / 32 (mixed)	71.4	-	-	-
HAQ (Wang et al., 2019)	100	1.8 MB	4 / 32 (mixed)	71.5	-	-	-
WPRN (Mishra et al., 2017)	100	-	-	-	5.8 MB	4 / 8	66.4
BCGD (Baskin et al., 2018)	80	-	-	-	5.8 MB	4 / 8	68.9
LQ-Net (Zhang et al., 2018a)	120	-	-	-	5.8 MB	4 / 32	70.0
DDQ (ours)	90	1.8 MB	4 / 4 (mixed)	**71.8**	5.8 MB	4 / 4 (mixed)	**71.1**
DDQ (ours)	30	1.8 MB	4 / 4 (mixed)	**71.5**	5.8 MB	4 / 4 (mixed)	**70.9**
DDQ (ours)	90	1.8 MB	4 / 4 (fixed)	**71.3**	5.8 MB	4 / 4 (fixed)	**70.9**
DDQ (ours)	30	1.8 MB	4 / 4 (fixed)	**70.7**	5.8 MB	4 / 4 (fixed)	**70.6**
HMQ (Habi et al., 2020)	50	1.7 MB	4 / 4 (mixed)	70.9	-	-	-
APOT (Li et al., 2020)	30	1.8 MB	4 / 4	69.7*	-	-	-
APOT (Li et al., 2020)	100	1.8 MB	4 / 4	71.0*	5.8 MB	4 / 4	70.7
LSQ (Esser et al., 2020)	90	1.8 MB	4 / 4	70.6*	5.8 MB	4 / 4	71.1
PROFIT (Park & Yoo, 2020)	140	1.8 MB	4 / 4	71.5	-	-	-
SAT (Jin et al., 2019)	150	1.8 MB	4 / 4	71.1	-	-	-
PACT (Choi, 2018)	110	1.8 MB	4 / 4	61.4	5.6 MB	4 / 4	69.2
DSQ (Gong et al., 2019)	90	1.8 MB	4 / 4	64.8	5.8 MB	4 / 4	69.6
TQT (Jain et al., 2019)	50	1.8 MB	4 / 4	67.8	5.8 MB	4 / 4	69.5
Uhlich *et al.* (Uhlich et al., 2020)	50	1.6 MB	4 / 4 (mixed)	69.7	5.4 MB	4 / 4	70.1
QIL (Jung et al., 2019)	90		-	-	5.8 MB	4 / 4	70.1
LQ-Net (Zhang et al., 2018a)	120		-	-	5.8 MB	4 / 4	69.3
NICE (Baskin et al., 2018)	120		-	-	5.8 MB	4 / 4	69.8
BCGD (Baskin et al., 2018)	80		-	-	5.8 MB	4 / 4	67.4
Dorefa-Net (Zhou et al., 2016)	110		-	-	5.8 MB	4 / 4	68.1

Table 1: Comparisons between DDQ and state-of-the-art quantizers on ImageNet. "W/A" means bitwidth of weight and activation respectively. Mixed precision approaches are annotated as "mixed". "-" denotes the absence of data in previous papers. We see that DDQ outperforms prior arts with much less training epochs. * denotes our re-implemented results using public codes.

Implementation Details. Training DDQ can be simply implemented in existing platforms such as PyTorch and Tensorflow. The forward propagation only involves differentiable functions except the Heaviside step function. In practice, STE can be utilized to compute its gradients, *i.e.* $\frac{\partial x_q}{\partial x} = 1_{\hat{q}_{min} \leq x \leq \hat{q}_{max}}$ and $\frac{\partial g_k}{\partial \hat{g}_k} = 1_{|\hat{g}_k| \leq 1}$, where $\hat{q}_{min}$ and $\hat{q}_{max}$ are minimum and maximum value in $\hat{\boldsymbol{q}} = \boldsymbol{U}^\top \boldsymbol{q}$. Our Appendix A.2 provides detailed procedure. The codes will be released.

4 Experiments

We extensively compare DDQ with existing state-of-the-art methods and conduct multiple ablation studies on ImageNet (Russakovsky et al., 2015). The results on CIFAR dataset (Krizhevsky et al., 2009) are in Appendix A.4.3. The reported validation accuracy are simulated on Qualcomm AIMET with INT8 ($b_q = 8$), if no other states.

Comparisons with Existing Methods. Table 1 compares DDQ with existing methods in terms of model size, bitwidth, and top1 accuracy on ImageNet using MobileNetv2 and ResNet18, which are two representative networks for portable devices. We see that DDQ outperforms recent state-of-the-art approaches by significant margins in different settings. For example, MobileNetV2+DDQ yields 71.7% accuracy when quantizing weights and activations using 4 and 8 bit respectively, while achieving 71.5% when training with 4/4 bit. These results only drop 0.2% and 0.4% compared to the 32-bit full-precision model, outperforming all other quantizers, which may decrease performance a lot ($2.4\%\sim10.5\%$). For ResNet18, DDQ outperforms all methods even the full-precision model (*i.e.* 71.0% vs 70.5%). More importantly, DDQ is trained for 30 epochs, reducing the training time compared to most of the reported approaches that trained much longer (*i.e.* 90 or 120 epochs). Note that PROFIT (Park & Yoo, 2020) achieves 71.5% on MobileNetv2 using a progressive training scheme (reducing bitwidth gradually from 8-bit to 5, 4-bit, 15 epochs each stage and 140 epochs in total.) This is quite similar to mixed-precision learning process of DDQ, in which bitwidth of each weight is initialized to maximum bits and learn to assign proper precision to each layer by decreasing the layerwise bitwidth. More details can see in Fig.5 in Appendix A.4.

Fig.3 shows the converged bitwidth for each layer of MobileNetv2 and ResNet18 trained with DDQ. We have two interesting findings. (1) Both networks tend to apply more bitwidth in lower layers, which have fewer parameters and thus being less regularized by the memory constraint. This allows to learn better feature representation, alleviating the performance drop. (2) As shown in the right hand side of Fig.3, we observe that depthwise convolution has larger bitwidth than the regular convolution.

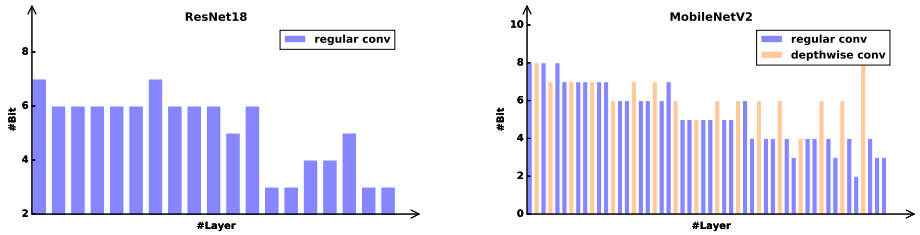


Figure 3: Learned quantization policy of each layer for ResNet18 and MobileNetV2 trained by DDQ on ImageNet. DDQ learns to allocate more bits to lower layers and depthwise layers of the networks.

	UQ	PoT	DDQ (fixed)	DDQ (mixed)	
Maximum bitwidth	4	4	4	6	8
Target bitwidth	4	4	4	4	4
Weight memory footprint	1×	1×	1×	0.98×	1.03×
Top-1 Accuracy (MobileNetV2)	65.2	68.6	70.7	71.2	71.5
Weight memory footprint	1×	1×	1×	1.01×	0.96×
Top-1 Accuracy (ResNet18)	70.0	67.8	70.6	70.8	70.9

Table 2: Comparisons between PACT(Choi et al., 2018)+UQ, PACT(Choi et al., 2018)+PoT and DDQ on ImageNet. "DDQ (fixed)" and "DDQ (mixed)" indicate DDQ trained with fixed / mixed bitwidth. We see that DDQ+mixed surpasses all counterparts.

As found in (Jain et al., 2019), the depthwise convolution with irregular weight distributions is the main reason that makes quantizing MobileNet difficult. With mixed-precision training, DDQ allocates more bitwidth to depthwise convolution to alleviate this difficulty.

Ablation Study I: mixed versus fixed precision. Table 2 compares DDQ trained using mixed precision to different fixed-precision quantization setups, including DDQ with fixed precision, uniform (UQ) and power-of-two (PoT) quantization by PACT (Choi et al., 2018) with gradient calibration (Jain et al., 2019; Esser et al., 2020; Jin et al., 2019). When the target bitwidth is 4, we see that DDQ trained with mixed precision significantly reduces accuracy drop of MobileNetv2 from 6.7% (*e.g.* PACT+UQ) to 0.4%.

Ablation Study II: adaptive resolution. We evaluate the proposed adaptive resolution by training DDQ with homogeneous bitwidth (*i.e.* fixed $\boldsymbol{U}$) and only updating $\boldsymbol{q}$. Table 3 shows performance of DNNs quantized with various quantization levels. We see that UQ and PoT incur a higher loss than DDQ, especially for MobileNetV2. We ascribe this drop to the irregular weight distribution as shown in Fig 1. Specially, when applying 2-bit quantization, DDQ still recovers certain accuracy compared to the full-precision model, while UQ and PoT may not converge. To our knowledge, DDQ is the first method to successfully quantize 2-bit MobileNet without using full precision in the activations.

Ablation Study III: gradient correction. We demonstrate how gradient correction improves DDQ. Fig 4(a) plots the training dynamics of layerwise quantization error *i.e.* $\|W_q - W\|_2^2$. We see that "DDQ+gradient correction" achieves low quantization error at each layer (average error is 0.35), indicating that the quantized values well approximate their continuous values. Fig. 4(b) visualizes the

	Full Precision	UQ	PoT	DDQ(fix)	DDQ(fix) + GradCorrect
Bitwidth (W/A)	32 / 32	4 / 8	4 / 8	4 / 8	4 / 8
Top-1 Accuracy (MobileNetV2)	71.9	67.1	69.2	71.2	**71.6**
Top-1 Accuracy (ResNet18)	70.5	70.6	70.8	70.8	**70.9**
Bitwidth (W/A)	32 / 32	4 / 4	4 / 4	4 / 4	4 / 4
Top-1 Accuracy (MobileNetV2)	71.9	65.2	68.4	70.4	**70.7**
Top-1 Accuracy (ResNet18)	70.5	70.0	68.8	70.6	**70.8**
Bitwidth (W/A)	32 / 32	2 / 4	2 / 4	2 / 4	2 / 4
Top-1 Accuracy (MobileNetV2)	71.9	-	-	60.1	**63.7**
Top-1 Accuracy (ResNet18)	70.5	66.5	63.8	67.4	**68.5**
Bitwidth (W/A)	32 / 32	2 / 2	2 / 2	2 / 2	2 / 2
Top-1 Accuracy (MobileNetV2)	71.9	-	-	51.1	**55.4**
Top-1 Accuracy (ResNet18)	70.5	62.8	62.4	65.7	**66.6**

Table 3: Ablation studies of adaptive resolution and gradient correction. "UQ" and "PoT" denote uniform and power-of-two quantization respectively. "DDQ(fix)+GradCorrect" refers to DDQ with gradient correction but fixed bitwidth. "-" denotes training diverged. "4/8" denotes training with 4-bit weights and 8-bit activations. Here we find that DDQ with gradient correction shows stable performances gains against DDQ (w/o gradient correction) and UQ/PoT baselines.

trained quantization levels. DDQ trained with gradient correction would capture the distribution to the original weights, thus reducing the quantization error. See Appendix A.4 for more details.

5 CONCLUSION

This paper introduced a differentiable dynamic quantization (DDQ), a versatile and powerful algorithm for training low-bit neural network, by automatically learning arbitrary quantization policies such as quantization levels and bitwidth. DDQ represents a wide range of quantizers. DDQ did well in the challenging MobileNet by significantly reducing quantization errors compared to prior work. We also show DDQ can learn to assign bitwidth for each layer of a DNN under desired memory constraints. Unlike recent methods that may use reinforcement learning(Wang et al., 2019; Yazdanbakhsh et al., 2018), DDQ doesn't require multiple epochs of retraining, but still yield better performance compared to existing approaches.

REFERENCES

Chaim Baskin, Natan Liss, Yoav Chai, Evgenii Zheltonozhskii, Eli Schwartz, Raja Giryes, Avi Mendelson, and Alexander M Bronstein. Nice: Noise injection and clamping estimation for neural network quantization. *arXiv preprint arXiv:1810.00162*, 2018.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Zhaowei Cai and Nuno Vasconcelos. Rethinking differentiable search for mixed-precision neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2349–2358, 2020.

Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5918–5926, 2017.

Jungwook Choi. Pact: Parameterized clipping activation for quantized neural networks. *arXiv: Computer Vision and Pattern Recognition*, 2018.

Jungwook Choi, Pierce I-Jen Chuang, Zhuo Wang, Swagath Venkataramani, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Bridging the accuracy gap for 2-bit quantized neural networks (qnn). *arXiv preprint arXiv:1807.06964*, 2018.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pp. 3123–3131, 2015.

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

Marc Davio. Kronecker products and shuffle algebra. *IEEE Transactions on Computers*, 100(2): 116–125, 1981.

Kalyanmoy Deb. Multi-objective optimization. In *Search methodologies*, pp. 403–449. Springer, 2014.

Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. 2020.

Alexander Goncharenko, Andrey Denisov, Sergey Alyamkin, and Evgeny Terentev. Fast adjustable threshold for uniform neural network quantization. *International Journal of Computer and Information Engineering*, 13(9):499–503, 2019.

Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4852–4861, 2019.

Hai Victor Habi, Roy H Jennings, and Arnon Netzer. Hmq: Hardware friendly mixed precision quantization block for cnns. *arXiv preprint arXiv:2007.09952*, 2020.

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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Gao Huang, Shichen Liu, Laurens Van der Maaten, and Kilian Q Weinberger. Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2752–2761, 2018.

Sambhav R Jain, Albert Gural, Michael Wu, and Chris H Dick. Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks. *arXiv: Computer Vision and Pattern Recognition*, 2019.

Qing Jin, Linjie Yang, and Zhenyu Liao. Towards efficient training for neural network quantization. *arXiv preprint arXiv:1912.10207*, 2019.

Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4350–4359, 2019.

Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Yuhang Li, Xin Dong, and Wei Wang. Additive powers-of-two quantization: A non-uniform discretization for neural networks. 2020.

Natan Liss, Chaim Baskin, Avi Mendelson, Alex M Bronstein, and Raja Giryes. Efficient non-uniform quantizer for quantized neural network targeting reconfigurable hardware. *arXiv preprint arXiv:1811.10869*, 2018.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Christos Louizos, Matthias Reisser, Tijmen Blankevoort, Efstratios Gavves, and Max Welling. Relaxed quantization for discretized neural networks. *arXiv preprint arXiv:1810.01875*, 2018.

Ping Luo, Peng Zhanglin, Shao Wenqi, Zhang Ruimao, Ren Jiamin, and Wu Lingyun. Differentiable dynamic normalization for learning deep representation. In *International Conference on Machine Learning*, pp. 4203–4211, 2019.

Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 116–131, 2018.

Jeffrey L McKinstry, Steven K Esser, Rathinakumar Appuswamy, Deepika Bablani, John V Arthur, Izzet B Yildiz, and Dharmendra S Modha. Discovering low-precision networks close to full-precision networks for efficient embedded inference. *arXiv preprint arXiv:1809.04191*, 2018.

Szymon Migacz. 8-bit inference with tensorrt. In *GPU technology conference*, volume 2, pp. 5, 2017.

Asit Mishra, Eriko Nurvitadhi, Jeffrey J Cook, and Debbie Marr. Wrpn: wide reduced-precision networks. *arXiv preprint arXiv:1709.01134*, 2017.

Daisuke Miyashita, Edward H Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*, 2016.

Eunhyeok Park and Sungjoo Yoo. Profit: A novel training method for sub-4-bit mobilenet models. *arXiv preprint arXiv:2008.04693*, 2020.

Qualcomm. Snapdragon neural processing engine sdk. In *https://developer.qualcomm.com/docs/snpe/setup.html*, 2019.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, pp. 525–542. Springer, 2016.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828, 2019.

Stefan Uhlich, Lukas Mauch, Fabien Cardinaux, Kazuki Yoshiyama, Javier Alonso Garcia, Stephen Tiedemann, Thomas Kemp, and Akira Nakamura. Mixed precision dnns: All you need is a good parametrization. In *International Conference on Learning Representations*, 2020.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8612–8620, 2019.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

Amir Yazdanbakhsh, Ahmed T Elthakeb, Prannoy Pilligundla, FatemehSadat Mireshghallah, and Hadi Esmaeilzadeh. Releq: An automatic reinforcement learning approach for deep quantization of neural networks. 2018.

Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. Understanding straight-through estimator in training activation quantized neural nets. *International Conference on Learning Representations*, 2019a.

Penghang Yin, Shuai Zhang, Jiancheng Lyu, Stanley Osher, Yingyong Qi, and Jack Xin. Blended coarse gradient descent for full quantization of deep neural networks. *Research in the Mathematical Sciences*, 6(1):14, 2019b.

Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 365–382, 2018a.

Ting Zhang, Guo-Jun Qi, Bin Xiao, and Jingdong Wang. Interleaved group convolutions. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4373–4382, 2017.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6848–6856, 2018b.

Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *International Conference on Learning Representations*, 2017.

Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *International Conference on Learning Representations*, 2017.

A APPENDIX

A.1 GRADIENT DERIVATION

We derive the gradient with respect to quantization parameters $\boldsymbol{\theta} = \{\boldsymbol{q}, \{\hat{g}_i\}_{i=1}^{b}\}$ in detail.

Gradient *w.r.t.* $\boldsymbol{q}$. Let $\boldsymbol{x}, \boldsymbol{x}_q$ be two vectors stacking values before and after quantization (x and x_q) respectively, the gradient of the loss function with respect to each entry q_k is given by

$$\frac{\partial \mathcal{L}}{\partial q_k} = \sum_{i=1}^{N} \frac{\partial \mathcal{L}}{\partial x_q^i} \frac{\partial x_q^i}{\partial q_k} \tag{9}$$

where $x_q^i = Q(x^i; \boldsymbol{\theta})$. From the definition of $Q(x^i; \boldsymbol{\theta})$ in Eqn.(3), we obtain

$$\frac{\partial x_q^i}{\partial q_k} = \begin{cases} \frac{1}{Z_U} & \text{if } k = \text{argmin}_j |\frac{1}{Z_U}(\boldsymbol{U}^\mathsf{T}\boldsymbol{q})_j - x^i| \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

Hence, we have

$$\frac{\partial \mathcal{L}}{\partial q_k} = \frac{1}{Z_U} \sum_{i \in \mathcal{S}_k} \frac{\partial \mathcal{L}}{\partial x_q^i}, \tag{11}$$

where $\mathcal{S}_k = \{m \,|\, m \in [N] \text{ and } (\boldsymbol{x}_o^m)_k = 1\}$ represents a set of indexes of the values discretized to the quantization level q_k.

Remark. *For $\boldsymbol{A} \in \mathbb{R}^{m_1 \times n_1}, \boldsymbol{B} \in \mathbb{R}^{m_2 \times n_2}$, then*

$$\boldsymbol{B} \otimes \boldsymbol{A} = \boldsymbol{T}_{m_1,m_2}(\boldsymbol{A} \otimes \boldsymbol{B})\boldsymbol{T}_{n_1,n_2} \tag{12}$$

where $\boldsymbol{T}_{m,n} = \sum_{i=1}^{m}(e_i^\mathsf{T} \otimes \boldsymbol{I}_n \otimes \boldsymbol{e}_i) = \sum_{j=1}^{n}(\boldsymbol{e}_j \otimes \boldsymbol{I}_m \otimes \boldsymbol{e}_j^\mathsf{T})$ is the perfect shuffle permutation matrix. $\boldsymbol{e}_i$ denotes the i-th canonical vector that is the vector with 1 in the i-th coordinate and 0 elsewhere. $\otimes$ is the Kronecker product. $\boldsymbol{I}_n$ is a n-by-n identity matrix.

Gradients *w.r.t.* g_k. The gradients back-propagated through the Heaviside step function $g_k = H(\hat{g}_k)$ can be approximated by the Straight-Through Estimator (STE) (Bengio et al., 2013; Yin et al., 2019a). Denote $\tilde{\boldsymbol{U}}_k = \otimes_{t=k+1}^{K} \boldsymbol{U}_t \otimes_{t=1}^{k-1} \boldsymbol{U}_t$, $\boldsymbol{U}$ can be reformulated by Remark A.1 as follows

$$\boldsymbol{U} = \frac{1}{Z_U}\left(\boldsymbol{T}_{2^{b-1},2}^{k-1}\right)\boldsymbol{U}_k \otimes \tilde{\boldsymbol{U}}_k \left(\boldsymbol{T}_{2^{b-1},2}^{k-1}\right)^\mathsf{T}. \tag{13}$$

Note that Z_U is also a function of g_k and $Z_U = \prod_{i=1}^{b}(2 - g_i)$, the derivative of $\boldsymbol{U}$ w.r.t. g_k can be derived as follows:

$$\begin{aligned} \frac{\partial \boldsymbol{U}}{\partial g_k} &= \frac{1}{Z_U} T_{2^{b-1},2}^{k-1} \left[\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \otimes \tilde{\boldsymbol{U}}_k\right]\left(\boldsymbol{T}_{2^{b-1},2}^{k-1}\right)^\mathsf{T} \\ &- \frac{1}{Z_U(2-g_k)} \boldsymbol{T}_{2^{b-1},2}^{k-1} \left[\begin{bmatrix} 1 & 1-g_k \\ 1-g_k & 1 \end{bmatrix} \otimes \tilde{\boldsymbol{U}}_k\right]\left(\boldsymbol{T}_{2^{b-1},2}^{k-1}\right)^\mathsf{T} \\ &= \frac{1}{Z_U(2-g_k)} \boldsymbol{T}_{2^{b-1},2}^{k-1} \left[\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \otimes \tilde{\boldsymbol{U}}_k\right]\left(\boldsymbol{T}_{2^{b-1},2}^{k-1}\right)^\mathsf{T}. \end{aligned} \tag{14}$$

where $\boldsymbol{T}_{2^{b-1},2}^{k-1}$ is a perfect shuffle permutation matrix (Davio, 1981). From Eqn.(14), we obtain $\frac{\partial \boldsymbol{U}}{\partial g_k}|_{g_k=1} = 2^{b+1}\frac{\partial \boldsymbol{U}}{\partial g_k}|_{g_k=0}$, implying that DDQ assigns smaller gradients to those inhibited gate ($g_k = 0$). In other words, once g_k decreases to 0, it is unlikely to return to 1, making DDQ appealing to achieve mixed-precision training. With Eqn.(9) and (14), the quantization parameters of DDQ and the weights of the network can be jointly optimized by using SGD.

A.2 TRAINING ALGORITHM

More details about training algorithm of DDQ can refer to Algorithm. 1.

Algorithm 1 Training procedure of DDQ

Input: the full precision kernel $\boldsymbol{W}$ and bias kernel $\boldsymbol{b}$, quantization parameter $\boldsymbol{\theta} = \{\boldsymbol{q}, \{\hat{g}_i\}_{i=1}^{b}\}$, the target bitwidth of b_m, input activation $\boldsymbol{y}_{in}$.

Output: the output activation $\boldsymbol{y}_{out}$

1: Apply DDQ to the kernel $\boldsymbol{W}$, bias $\boldsymbol{b}$, input activation $\boldsymbol{y}_{in}$ by Eqn.(3)
2: Compute the output activation $\boldsymbol{y}_{out}$ by Eqn.(4)
3: Compute the loss $\mathcal{L}$ by Eqn.(6) and gradients $\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_{out}}$
4: Compute the gradient of ordinary kernel weights and bias $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}$, $\frac{\partial \mathcal{L}}{\partial \boldsymbol{m}}$
5: Applying gradient correction in Eqn.(9) to compute the gradient of parameters, $\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}}$, $\frac{\partial \mathcal{L}}{\partial \hat{g}_i}$ by Eqn.(7) and Eqn.(8)
6: Update $\boldsymbol{W}, \boldsymbol{m}, \hat{g}_i$ and $\boldsymbol{q}$.

Method	Differentiablity	Mixed Precision	Quantization Level	Step Size	Quantizer Calibration	Gradient Calibration
X-Nor-Net (Rastegari et al., 2016)	✓		UQ			
DoReFa-Net (Zhou et al., 2016)	✓		UQ+Tanh			
WPRN (Mishra et al., 2017)	✓		UQ+Tanh			
PACT (Choi, 2018)	✓		UQ+Tanh	✓	✓	
FAQ (McKinstry et al., 2018)	✓		UQ		✓	
NICE (Baskin et al., 2018)	✓		UQ		✓	
LSQ (Esser et al., 2020)	✓		UQ	✓	✓	✓
BCGD (Yin et al., 2019b)	✓		UQ			✓
TQT (Jain et al., 2019)	✓		UQ	✓	✓	✓
HAQ (Wang et al., 2019)		✓	UQ	✓	✓	
Releq (Yazdanbakhsh et al., 2018)		✓	UQ			
Uhlich *et al.* (Uhlich et al., 2020)	✓	✓	UQ/Non-UQ	✓	✓	✓
INQ (Zhou et al., 2017)			Non-UQ			
PoT (Miyashita et al., 2016)	✓		Non-UQ			
HWGQ (Cai et al., 2017)	✓		Non-UQ	✓	✓	✓
QIL (Jung et al., 2019)	✓		Learned	✓	✓	
LQ-Net (Zhang et al., 2018a)	✓		Learned	✓		
DDQ (ours)	✓	✓	Learned	✓	✓	✓

Table 4: Overall summary of state-of-art quantiztaion methods. "Differentiablity" column shows whether this method can be implemented with one-stage and gradient-based methods. "UQ" and "Non-UQ" indicate uniform / non-uniform quantization respectively. "Step Size" column denotes the ability to adjust quantization step size. "Quantizer Calibration" means if the method calibrates the quantizer with centre points and thresholds. "Gradient Calibration" shows if the quantization gradients for parameters in quantizer are corrected.

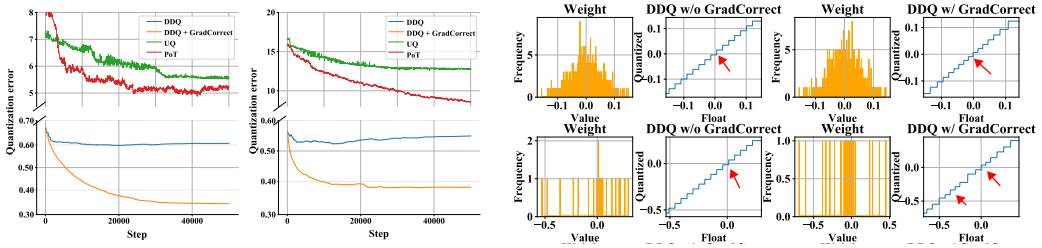
A.3 Summary of Existing Quantization Approach

Table 4 gives an overall summary of existing quantization methods. For uniform quantization methods, to reduce quantization error, (Zhou et al., 2016; Mishra et al., 2017; Choi, 2018) use tanh function to project quantization levels, but they restrict quantization levels in specific patterns. Besides, other methods such as (Choi, 2018; McKinstry et al., 2018; Jain et al., 2019), calibrate quantizer with estimated or learned centre points and thresholds, also yielding better performance. (Miyashita et al., 2016; Zhou et al., 2017; Cai et al., 2017) show that non-uniform quantization levels can outperform uniform counterparts in specific situations, and they can perform better if we learn them from data, as discussed in (Zhang et al., 2018a; Jung et al., 2019). More recently, Mixed precision quantization techniques are introduced by (Wang et al., 2019; Yazdanbakhsh et al., 2018) and (Uhlich et al., 2020), further improving quantization methods by assigning different bitwidth to each layer using Reinforcement-Learning or Gradient-based methods. As shown in Table 4, the proposed DDQ can integrate main properties of above methods, learning to select optimal quantization policy according to corresponding data and model architectures.

A.4 Experimental Details

A.4.1 Evaluation on ImageNet

The ImageNet dataset consists of 1.2M training and 50K validation images. For ResNet and MobileNet, we adopt standard data preprocessing in the original paper (He et al., 2016; Sandler et al., 2018). All DNN+DDQs are trained for 30 epochs with cosine learning rate scheme(Loshchilov & Hutter, 2016) like (Esser et al., 2020). We choose PACT (Choi, 2018) with gradient calibration (Jain



(a) Training dynamics of quantization error. (b) Learned quantization level for channels.

Figure 4: Training dynamics of quantization error in MobileNetV2. (a) compares the quantization errors of PACT+UQ, PACT+PoT, and DDQ with/without gradient correction. DDQ with gradient correction shows stable convergence and lower quantization errors than counterparts. (b) compares the converged quantization levels of DDQ for each channel with/without gradient correction, and dense regions are marked by arrows. Here we can also see that quantization levels learned by DDQ with gradient correction could fit original data distribution better.

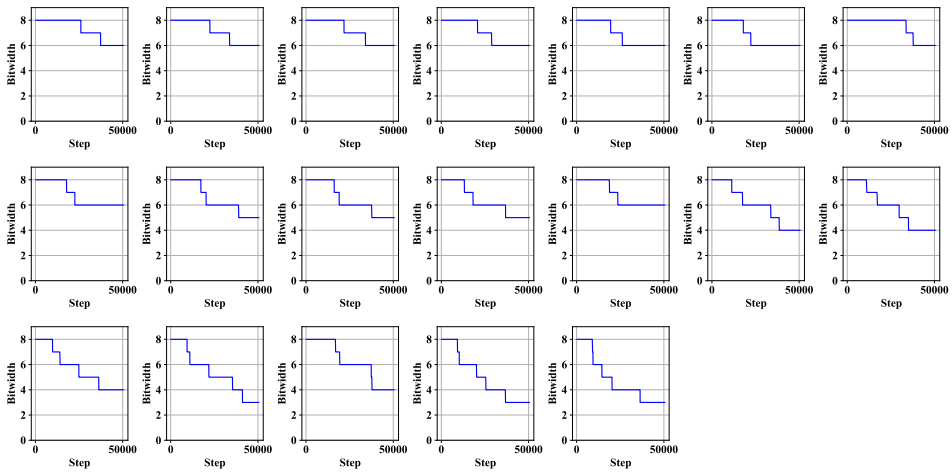


Figure 5: Evolution of bitwidth of each layer when training ResNet18. We can see that DDQ can learn to assign bitwidth to each layer under given memory footprint constraints.

et al., 2019; Esser et al., 2020; Jin et al., 2019; Li et al., 2020) pipeline as baselines. All hyper-parameters follow prior arts such as PACT(Choi, 2018) for fair comparisons, *e.g.* l2-regularization coefficient $\lambda = 1e-2$. Network weights are quantized using uniform quantization (UQ), power-of-two quantization (PoT) and DDQ respectively, and all activations are uniformly quantized for fair comparison. For PACT, parameterized clipping values are initialized to 6.0 for activations and 3.0 for weights. We adopt per-tensor quantization for activations and per-channel quantiztaion for weights as recommended in (Rastegari et al., 2016; Krishnamoorthi, 2018; Goncharenko et al., 2019) to handle the widely-varying range between channels. Note that weights of all layers are quantized with UQ/PoT/DDQ directly except those of first and last layer, for which we employ 8-bit uniform quantization to observe standard practice of all state-of-the-art works. In addition, training DDQ will cause extra computation, but is still efficient and comparable to training UQ and PoT. Specially, training DDQ-MobileNetV2 cost 29.3 min averagely for each epoch on 8 Nvidia GTX 1080Ti, less than 5% longer than PoT (28.4 min) and UQ (27.9 min).

Mixed Precision Quantization. Mixed-precision quantization is new in the literature and proven to be superior to their fixed bitwidth counterparts (Wang et al., 2019; Uhlich et al., 2020; Cai & Vasconcelos, 2020; Habi et al., 2020). DDQ is naturally used to perform mix-precision training by a binary block-diagonal matrix $\boldsymbol{U}$. In DDQ, each layer is quantized between 2-bit and a given maximum precision, which may be 4-bit / 6-bit / 8-bit. Items of gate $\{\hat{g}_i\}_{i=1}^{b}$ are initialized all positively to $1e-8$, which means $\boldsymbol{U}$ is identity matrix and precision of layers are initialized to their

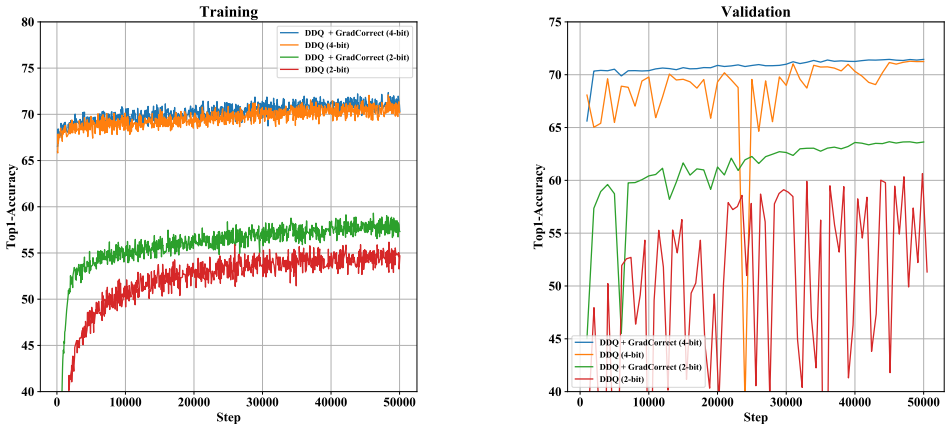


Figure 6: Training dynamics of fixed precision DDQ w/ (or w/o) gradient correction.

Methods	bitwidth(w/a)	Mixed-precision	Latency(ms)	Top-1(%)
FP	32/32		7.8	71.9
UQ	4/8		3.9	67.1
DDQ (fixed)[1]	4/8		4.5	71.6
DDQ (mixed UQ)[2]	4/8	✓	4.1	70.8
DDQ[3]	4/8	✓	5.1	71.7

Table 5: Comparison of Quantized MobileNetv2 runing on mobile DSPs. [1] Fixed precision DDQ. [2] Mixed precision DDQ with uniform quantizer constraints. [3] Original DDQ. "w/a" means the bitwidth for network weights and activations respectively.

maximum values. We set target bitwidth as 4, constraining models to 4-bit memory footprint, and then jointly train $\{\hat{g}_i\}_{i=1}^{b}$ with other parameters of corresponding model and quantizers. For memory constraints, α is set to -0.02 empirically. We use learning rates $1e-8$ towards $\{\hat{g}_i\}_{i=1}^{b}$, ensuring sufficient training when precision decreasing. Fig. 5 depicts the evolution of bitwidth for each layer when quantizing a 4-bit ResNet18 using DDQ with maximum bitwidth 8. As demonstrated, DDQ could learn to assign bitwidth to each layer, in a data-driven manner.

Gradient Correction. For fixed-precision DDQ, we have an interesting observation that the proposed gradient correction could stabilize training. For instance, Fig. 6 illustrates training dynamics for 2/4-bit DDQ-quantized MobileNetV2. With gradient correction, the quantized model not only yields better performance (both training and validation), but also converges with less jitters in validation accuracy.

A.4.2 Evaluation on Mobile Devices

In Table 5. we further evaluate DDQ on mobile platform to trade off of accuracy and latency. We deploy the trained DDQ model on Qualcomm Snapdragon 865 processor, evaluating model latency using SNPE engine (Qualcomm, 2019). With implementation stated in section 2.2, DDQ achieves over 3% accuracy gains compared to UQ baseline. Moreover, in contrast to FP model, DDQ runs with about 40% less latency with just a small accuracy drop (<0.3%). Note that all tests here are running under INT8 simulation due to the support of the platform. We believe the acceleration ratio can be larger in the future when deploying DDQ on more compatible hardwares.

	Methods	Accuracy		
		2-bit	3-bit	4-bit
ResNet20 (FP : 92.4)	DoReFa-Ne (Zhou et al., 2016)	88.2	89.9	90.5
	PACT (Choi, 2018)	89.7	91.1	91.7
	LQ-Net (Zhang et al., 2018a)	90.2	91.6	-
	SAWB (Choi et al., 2018)	90.5	-	-
	TQT (Jain et al., 2019)	91.2	-	-
	Uhlich *et al.* (Uhlich et al., 2020)	91.4	-	-
	DDQ (mixed)	**91.6**	**92.2**	**92.7**

Table 6: Comparison of Cifar-10 Top1-accuracy towards existing quantization methods. All the reported results use 32-bit activation by following prior work.

A.4.3 EVALUATION ON CIFAR-10

Additionally, we quantize ResNet20 on Cifar-10 with mixed precision. For weight quantization, we adopt 2-/3-/4-bit target bitwidth and initialize DDQ with maximum bitwidth 8. Tabel 6 compares our results with other weight-only quantization methods. For Cifar-10, all layers of the model are quantized using DDQ. The quantized models are trained for 200 epochs with learning rate 0.01, batch size 1024 and cosine scheduler.