Scheduling Weight Transitions for Quantization-Aware Training

Junghyup Lee[1,*] Jeimin Jeon[2,3,*] Dohyung Kim[4] Bumsub Ham[2,†]

[1] Samsung Research [2]Yonsei University [3]Articron Inc. [4] Samsung AI center

https://cvlab.yonsei.ac.kr/projects/TRS/

Abstract

Quantization-aware training (QAT) simulates a quantization process during training to lower bit-precision of weights/activations. It learns quantized weights indirectly by updating latent weights, i.e.*, full-precision inputs to a quantizer, using gradient-based optimizers. We claim that coupling a user-defined learning rate (LR) with these optimizers is sub-optimal for QAT. Quantized weights transit discrete levels of a quantizer, only if corresponding latent weights pass transition points, where the quantizer changes discrete states. This suggests that the changes of quantized weights are affected by both the LR for latent weights and their distributions. It is thus difficult to control the degree of changes for quantized weights by scheduling the LR manually. We conjecture that the degree of parameter changes in QAT is related to the number of quantized weights transiting discrete levels. Based on this, we introduce a transition rate (TR) scheduling technique that controls the number of transitions of quantized weights explicitly. Instead of scheduling a LR for latent weights, we schedule a target TR of quantized weights, and update the latent weights with a novel transition-adaptive LR (TALR), enabling considering the degree of changes for the quantized weights during QAT. Experimental results demonstrate the effectiveness of our approach on standard benchmarks.*

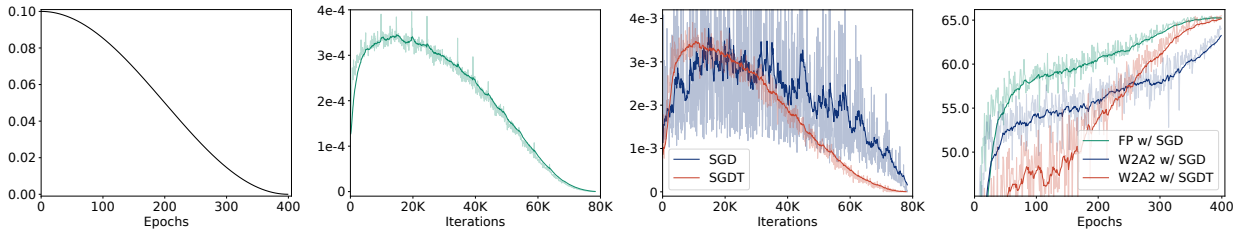
1. Introduction

Recent neural networks use wide and deep architectures [11, 16, 18, 41, 44] requiring millions of parameters and computations. Network quantization converts full-precision weights and/or activations into low-bit ones. This reduces storage usage and computational overheads drastically, but the low-bit models perform worse than the full-precision ones. To alleviate this problem, many approaches [5, 9, 20, 31, 36] adopt quantization-aware training (QAT) that simulates a quantization process during training. It is not straightforward to optimize discrete quantized weights using gradient-based optimizers (*e.g.*, stochastic gradient descent (SGD)) with continuous gradients. QAT instead exploits full-precision latent weights and a quantizer involving a discretization function (*e.g.*, a round function) to update the quantized weights indirectly.

QAT mainly consists of three steps: (1) In a forward propagation step, full-precision latent weights are converted to quantized weights using a quantizer, and the quantized weights are used to compute an output; (2) Gradients w.r.t an objective function are then back-propagated to the latent weights in a backward propagation step; (3) In an optimization step, the latent weights are updated with the gradients. Previous works mostly focus on the forward and backward propagation steps by designing quantizers [4, 5, 9, 20, 26, 36, 45, 47, 48] and addressing a vanishing gradient problem, caused by the discretization function in a quantizer [1, 10, 21, 25, 46], respectively. They do not pay attention to the optimization step, and simply exploit gradient-based optimizers with a user-defined learning rate (LR), such as SGD or Adam [22], to update the latent weights. This optimization strategy is, however, designed for training full-precision models, and does not consider how quantized weights are changed, which is sub-optimal for QAT.

When training a full-precision model with a gradient-based optimizer, we typically decay a LR progressively to update full-precision weights in a coarse-to-fine manner, which guarantees the convergence of the model [17, 23]. That is, we can control the degree of weight changes manually by scheduling the LR, since the magnitude of a single parameter change, so-called an *effective step size* [22], is highly correlated with the LR (Fig. 1a vs. Fig. 1b). For example, an average effective step size, which quantifies the degree of changes in weights, for a small LR is lower than that for a large one. We have found that this does not hold, when the optimizers coupled with a LR are applied to a quantized model in QAT. Since QAT updates quantized weights indirectly from latent weights and a quantizer, an average effective step size for quantized weights is less correlated with a LR for latent weights (Fig. 1a vs. the blue curve in Fig. 1c). To be more specific, different from full-

*Equal contribution. †Corresponding author.



(a) LR decay for SGD. (b) Avg. effective step sizes for FP weights (16^{th} layer). (c) Avg. effective step sizes for 2-bit weights (16^{th} layer). (d) Test accuracy (%).

Figure 1. Training curves of full-precision (FP) and quantized models for ResNet-20 [11] on CIFAR-100 [24]. Both weights (W) and activations (A) are quantized to a 2-bit precision (W2A2). With a gradient-based optimizer (SGD), we can control the average effective step size of FP weights roughly by scheduling a LR ((a) vs. (b)), while we could not for quantized weights (the blue curve in (c)). The curve for quantized weights is noisy, and decreases rapidly at the end of training, suggesting that 1) the quantized weights can alter significantly with a small LR and/or a small change of a LR, disturbing a coarse-to-fine parameter update and causing an unstable training, and 2) adopting a manually scheduled LR for QAT is sub-optimal. The optimizer coupled with our scheduling technique (SGDT) can control the average effective step size of quantized weights by adjusting the number of transitions explicitly (the red curve in (c)), showing better results in terms of accuracy and convergence (the red curve in (d)).

precision weights, quantized weights change their discrete levels, only when corresponding latent weights pass transition points, where the quantizer alters discrete states (we call this as *transitions*). This makes it hard to control an average effective step size of quantized weights by adjusting a LR, disturbing coarse-to-fine parameter updates and the convergence of a model, even at the end of training (the blue curve in Fig. 1d). For example, if latent weights are concentrated around a transition point, they can pass the point easily with a tiny LR, inducing significant changes of quantized weights.

In this paper, we introduce a transition rate (TR) scheduling technique for QAT, which allows to update latent weights w.r.t the transitions of quantized weights explicitly. We define the TR of quantized weights as the number of quantized weights changing discrete levels at each iteration for optimization, divided by the total number of the weights. Note that an effective step size of each quantized weight is either zero or a discrete value (*i.e.*, a distance between two quantization levels), indicating that an average effective step size for quantized weights is mainly affected by the number of transitions. We thus conjecture that the number of transitions, or similarly, the TR is a key for controlling the degree of parameter changes for quantized weights. Based upon this, we propose to schedule a target TR of quantized weights, instead of a LR for latent weights, and update the latent weights with a novel transition-adaptive learning rate (TALR) to adjust a TR of quantized weights accordingly. The TALR is changed adaptively to match the current TR of quantized weights with the target one. By scheduling the target TR, we are able to adjust the average effective step size of quantized weights (the red curve in Fig. 1c). This allows us to optimize quantized weights in a coarse-to-fine manner, and provides a stable training process (the red curve in Fig. 1d). To the best of our knowledge, scheduling a TR for QAT has not been explored, instead of a LR as in full-precision training. We demonstrate the superiority of our TR scheduling technique over the plain LR scheduling using various network architectures [11, 29, 31, 37, 43] and optimizers (*e.g.*, SGD, Adam [22], and AdamW [33]) on image classification [7, 24] and object detection [28]. In summary, the main contributions of our work are threefold:

- We claim the necessity of a training scheduler specialized for general QAT for the first time, where quantized weights are optimized indirectly by latent weights and a quantizer. To update latent weights in QAT, we propose to focus on the changes of quantized weights used for computing an output of a quantized network, which has not been covered by plain optimizers using a user-defined LR.
- We present a novel TR scheduling technique for QAT together with a TALR, which is adjusted considering a TR of quantized weights, controlling an average effective step size of quantized weights accordingly.
- We demonstrate the effectiveness and generalization ability of our approach on network quantization, boosting the performance of various models consistently.

2. Related Work

QAT. QAT methods simulate quantization during training by converting full-precision latent weights into quantized ones. Early works adopt fixed quantizers for binary [36, 48], ternary [26], and multi-bit [4, 48] representations, attempting to minimize quantization errors. Recent approaches introduce trainable quantizers that learn quantization parameters, such as intervals [5, 9, 20] or non-uniform levels [45, 47], significantly improving performance. These methods rely on the straight-through estimator (STE) [2] to handle non-differentiable quantization, but suffer from gradient mismatch [21, 46]. To address this, several works avoid exploiting the STE by using differentiable quantizers [10, 21, 46] or adjusting the gradients depending on the

latent weights [1, 25]. Despite advancements, they overlook the fact that quantized weights act differently from full-precision counterparts, and simply adopt the same optimization strategies for full-precision models, that exploit manual LR scheduling techniques (*e.g.*, step decay or cosine annealing [32]), to train a quantized model. We highlight that quantized weights are updated indirectly by full-precision latent weights and a quantizer. Based on this, we propose to focus on actual changes of quantized weights, and present a TR scheduling technique specialized for QAT.

Recently, the work of [34] points out that quantized weights tend to oscillate between adjacent quantization levels during QAT. To address the oscillation problem, it proposes to freeze latent weights during training or to incorporate a regularization term into the objective function. While this approach could alleviate the oscillations, it still relies on the conventional optimization method using a LR, and thus it cannot control the average effective step size of quantized weights explicitly. Moreover, hyperparameters are chosen carefully according to network architectures and bit-widths, since freezing weights or adding the regularization term could disturb the training process. We have observed that it is difficult to control the average effective step size of quantized weights with the LR scheduling, and the reason for this is closely related to the oscillation problem. Different from [34] that reduces oscillations themselves by freezing or regularizing the weights, we attempt to control the actual change of quantized weights by scheduling a target TR. In this way, we can achieve consistent performance gains under the various bit-width settings with the same set of hyperparameters for each architecture, which is not feasible for [34].

Most recently, pseudo-quantization training (PQT) [6, 38, 39] addresses oscillations by injecting pseudo-quantization noise into full-precision weights, instead of exploiting the quantization operations during training. While PQT stabilizes training, it introduces discrepancies between training and inference, as the actual quantization operations are not considered during training, leading to suboptimal performance. To overcome this, they leverage auxiliary techniques like knowledge distillation or mixed-precision quantization, but at the cost of computational overheads. In contrast, our method mitigates oscillations directly through TR scheduling, maintaining consistency between training and inference with minimal overhead, leading to consistent performance gains over plain optimization methods, with negligible overheads.

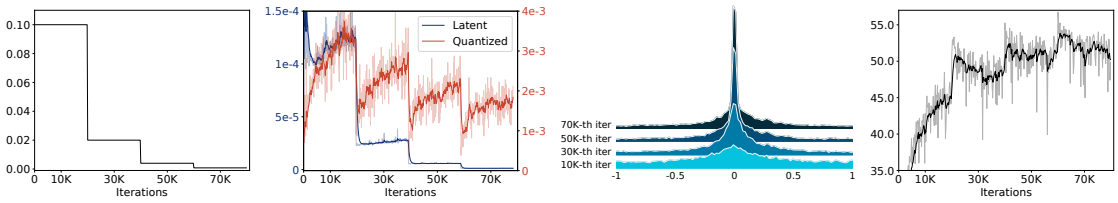
Optimization Methods. Neural networks are generally trained using a gradient-based optimizer coupled with a LR scheduling technique. Gradient-based optimizers update network weights based on the gradients w.r.t an objective function. SGD is a vanilla optimizer for minimizing the objective function, which often exploits the first moment of the gradients to accelerate update steps near local optima. Many works [8, 22, 42] propose advanced optimizers using adaptive gradients. They accumulate squared gradients using a historical sum [8] or a running average [22, 42] (*i.e.*, the second moment of gradients), which are then used for normalizing each gradient dimension adaptively. This enables highlighting infrequent features at training time, which is particularly useful for sparse gradients [8]. All of these optimizers exploit a LR to update full-precision weights. They typically use a large LR initially and decay it to a small value by a LR scheduling technique, such as step decay or cosine annealing [32]. The large initial LR encourages weights to explore local optima in a loss space, and the small LR at the end of training prevents the weights from overshooting from a local optimum [17, 23]. A recent work [27] also points out that the LR scheduling is important for both generalization and performance, since large and small LRs play complementary roles in memorizing different types of patterns. The aforementioned optimization strategies are, however, designed for training a full-precision model, directly updating full-precision weights used for inference. For QAT, quantized weights are used for forward/backward passes, but full-precision latent weights are instead updated to train the quantized weights. We propose to consider this unique property of QAT to optimize the latent weights, and introduce a novel TR scheduling technique, specially designed for network quantization.

Closely related to ours, few works [12, 40] present optimizers for binary networks that train binary weights directly, rather than using latent weights. To this end, they adopt the first moment of gradients [12] or its variant using the second moment [40] to flip binary values via thresholding. Although these methods do not use latent weights, the first moment and its variant have a role similar to the latent weights in that both accumulate gradients, and they schedule a threshold manually for flipping binary values. This suggests that they do not consider the degree of parameter changes in the binary weights explicitly, causing the same problem as the conventional optimizers using a LR. These methods are also applicable for binary networks only. On the contrary, our method adjusts a TR explicitly to control an average effective step size of quantized weights consequently, and it can be applied to general QAT, including both binary and multi-bit quantization schemes.

3. Preliminary

QAT inserts weight and/or activation quantizers into each layer of a neural network to simulate a quantization process at training time. Here we briefly describe a quantizer and an optimizer in QAT.

Quantizer. Weight and activation quantizers take full-precision latent weights and activations in a layer, respectively, and produce low-bit representations. Here we mainly



(a) LR decay for SGD. (b) Avg. effective step sizes. (c) Distribution of normalized latent weights. (d) Test accuracy (%).

Figure 2. Empirical analysis on QAT using SGD with a step LR decay. We binarize both weights and activations of ResNet-20 [11] and train the model on CIFAR-100 [24]. For the visualizations in (b) and (c), we track the latent and quantized weights in the 16$^{\text{th}}$ layer. We can see that the average effective step size of latent weights (the blue curve in (b)) is controlled by the LR in (a), while that for the quantized weights changes significantly even with a small LR (the red curve in (b)). This is because the change of quantized weights is also affected by the distribution of latent weights approaching the transition point (*i.e.*, zero in (c)). The large changes in the quantized weights at the end of training (the red curve in (b)) degrade the performance in (d). (Best viewed in color.)

explain the weight quantizer. The activation quantizer is similarly defined. Let us denote by $\mathbf{w}$ full-precision latent weights. The quantizer first normalizes and clips the latent weights to adjust their range:

$$\mathbf{w}_n = f(\mathbf{w}), \tag{1}$$

where we denote by $\mathbf{w}_n$ normalized weights. f is a normalization function involving scaling and clipping operations, which can be either hand-designed [48] or be trainable [5, 9, 20]. The normalized weights $\mathbf{w}_n$ are then converted to discrete ones $\mathbf{w}_d$ using a discretization function g:

$$\mathbf{w}_d = g(\mathbf{w}_n). \tag{2}$$

The discretization function g is typically a signum or a round function for binary or multi-bit quantization schemes, respectively. Note that STE [2] is usually adopted in a backward pass to avoid a vanishing gradient problem, caused by the discretization function, propagating the same gradient from $\mathbf{w}_d$ to $\mathbf{w}_n$. Lastly, the quantizer produces quantized weights $\mathbf{w}_q$ by de-normalizing the discrete weights $\mathbf{w}_d$:

$$\mathbf{w}_q = h(\mathbf{w}_d), \tag{3}$$

where h is a de-normalization function for post-scaling. The de-normalization could possibly be omitted (or fixed) when the quantized layer is followed by a normalization layer (*e.g.*, batch normalization [19]), since it imposes the scale invariance to the weights and activations [13, 14], suggesting that de-normalization has no effect on either the forward or backward pass.

Optimizer. In QAT, the latent weights $\mathbf{w}$ are updated, instead of optimizing the quantized weights $\mathbf{w}_q$ directly. That is, updating the latent weights in turn alters the quantized ones during training. More specifically, the quantized weights change their discrete levels if corresponding normalized latent weights $\mathbf{w}_n$ pass transition points of the discretization function g (*e.g.*, zero for the signum function) after updating the latent weights $\mathbf{w}$. Previous works typically use gradient-based optimizers with a user-defined LR to update the latent weights as follows:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \mu^t \mathbf{g}^t, \tag{4}$$

where the superscript t indicates an iteration step, and we denote by $\mathbf{g}$ and μ a gradient term and the LR, respectively. Note that the gradient term $\mathbf{g}$ is computed differently depending on the types of optimizers. For example, SGD uses the first moment of gradients.

4. Method

In this section, we first present a detailed analysis of a conventional optimization method using a manually scheduled LR in the context of QAT (Sec. 4.1). We then introduce a novel TR scheduling technique (Sec. 4.2).

4.1. Empirical analysis

Conventional optimizers use a LR decay technique when training a full-precision model. They update model parameters gradually in a coarse-to-fine manner, which encourages a model to find a better local optimum in a loss space, and prevents overshooting from a local optimum [17, 23]. This suggests that the optimizers control an average effective step size (*i.e.*, the degree of parameter changes) of full-precision weights by adjusting the LR. We have empirically found that this does not hold for QAT. Namely, the average effective step size of quantized weights in QAT is hardly controlled by a conventional LR scheduling technique in gradient-based optimizers.

To understand this problem in detail, we show an empirical analysis on 1) how a gradient-based optimizer, coupled with a manually scheduled LR, changes latent and quantized weights within a framework of QAT, and 2) the influence of the changes on the classification accuracy of a quantized model (Fig. 2). We train ResNet-20 [11] with binary weights and activations on CIFAR-100 [24] using a SGD optimizer with a step LR decay method. We can see from Fig. 2a and the blue curve in Fig. 2b that the average effective step size of latent weights is controlled by a LR, which is consistent with the result in a full-precision model (*e.g.*, Fig. 1a vs. Fig. 1b). The reason is that the latent weights in

QAT and the weights in a full-precision model are continuous values, and the LR is responsible directly for updating the weights, *e.g.*, as in Eq. (4). On the contrary, quantized weights alter significantly, even with a small LR (the red curve in Fig. 2b). Since QAT uses quantized weights in a forward propagation step to compute gradients w.r.t an objective function, the large changes of quantized weights at the end of training make a training process unstable, disturbing a quantized model to converge (Fig. 2d).

To delve deeper into this problem, let us suppose that a quantized weight needs to alter its discrete level (*e.g.*, from a negative value to a positive one in the binary quantization) in order to minimize a training loss. A corresponding latent weight then keeps accumulating gradients to move towards a transition point, and once a transition occurs, the latent weight might stay near the transition point. We can observe in Fig. 2c that the normalized latent weights (*i.e.*, $\mathbf{w}_n$ in Eq. (1)) are approaching the transition point (*i.e.*, zero in this case) progressively according to the number of iterations. The quantized weight is hence likely to oscillate between adjacent discrete levels with small LRs in later training iterations (see the high peak at the 70K-th iteration in Fig. 2c). This coincides with the recent finding in [34, 35] that the quantized weights tend to oscillate during QAT, making it difficult to stabilize the batch normalization statistics [19], and degrading the performance at test time. This analysis indicates that 1) the average effective step size of quantized weights is largely affected by the distribution of latent weights, and 2) the reason why the LR is not a major factor for controlling the average effective step size in QAT, contrary to an optimization process of a full-precision model, is that the quantized weight alters only when the latent weight passes a transition point of a quantizer, but the LR cannot adjust the number of transitions explicitly. Consequently, our empirical analysis suggests the necessity of a training scheduler specific to QAT that allows to update latent weights adaptively considering the transitions in quantized weights.

4.2. TR scheduler

Here we present a relationship between an effective step size and transitions in quantized weights, and describe our approach to TR scheduling in a single layer.

TR of quantized weights. We say that a transition occurs if a latent weight passes a transition point of a quantizer after a single update. The number of transitions is hence equal to that of quantized weights changing discrete levels after the update. We can count the number of transitions by observing whether discrete weights (*i.e.*, $\mathbf{w}_d$ in Eq. (2)) are changed or not after the update. Here we focus on a TR, the number of transitions divided by the total number of quantized weights, defined as follows:

$$k^t = \frac{\sum_{i=1}^{N} \mathbb{I}\left[w_d^t(i) \neq w_d^{t-1}(i)\right]}{N}, \tag{5}$$

where we denote by k^t and $w_d^t(i)$ the TR and the i-th element of discrete weights at the t-th iteration step, respectively, and N is the total number of quantized weights. $\mathbb{I}[\cdot]$ is an indicator function that outputs one if a given statement is true and zero otherwise.

Relation between an effective step size and a transition. An *effective step size* [22] indicates the magnitude of a single parameter change. We can compute the effective step size of a quantized weight w_q by measuring its absolute difference before and after a single update as follows:

$$\left|\triangle w_q^t\right| = \left|w_q^t - w_q^{t-1}\right|, \tag{6}$$

where we denote by $|\triangle w_q^t|$ an effective step size of the quantized weight at the t-th iteration step. We will show that the effective step size is related to a transition of the quantized weight. Let us denote by δ^t a post-scaling factor of the de-normalization function h in Eq. (3) at the t-th iteration step. If the discretization function g in Eq. (2) is a rounding function for multi-bit quantization (*i.e.* a discrete weight w_d^t is an integer value), we can rewrite Eq. (6) as follows:

$$\left|\triangle w_q^t\right| = \left|\delta^t w_d^t - \delta^{t-1} w_d^{t-1}\right|. \tag{7}$$

If g is a signum function (*i.e.*, $w_d^t \in \{-1, 1\}$) for binary quantization, Eq. (6) can be represented as follows:

$$\left|\triangle w_q^t\right| = \frac{1}{2}\left|\delta^t w_d^t - \delta^{t-1} w_d^{t-1}\right|. \tag{8}$$

Note that the change of δ^t in a single update is typically small (*i.e.*, $\delta^t \approx \delta^{t-1}$) or we can set the post-scaling factor δ^t as a constant value if the quantized layer is followed by a normalization layer [13, 14] (*e.g.*, as in [25]). Assuming that the change of δ^t is negligible within a single update and a latent weight passes a single transition point when a transition occurs, we can approximate the effective step size of the quantized weight as follows:

$$\left|\triangle w_q^t\right| \approx \delta^t \mathbb{I}\left[w_d^t \neq w_d^{t-1}\right]. \tag{9}$$

That is, the effective step size of the quantized weight is at most δ^t if a transition occurs, and zero otherwise. This indicates that individual effective step sizes of quantized weights are discrete values (*i.e.*, zero or δ^t) determined by the quantizer. Note that the effective step size for each full-precision weight can be adjusted by a LR, since the weight is a continuous value, which is however not applicable for the quantized weight changing discretely. Accordingly, adjusting the number of transitions, or equivalently a TR, is

important to control an average effective step size of quantized weights. Based upon this, we design a TR scheduling technique adjusting a TR of quantized weights explicitly, allowing us to control the degree of parameter changes in the quantized weights accordingly.

TR scheduler. We incorporate our TR scheduling technique into an optimization process by introducing a transition-adaptive learning rate (TALR) to update latent weights, allowing to adjust a TR of quantized weights manually, w.r.t a target TR. To this end, we mainly apply three operations at every iteration: Estimating a running TR using a momentum estimator, adjusting a TALR w.r.t a target value, and updating latent weights. Specifically, we first compute a running TR of quantized weights for each iteration t using an exponential moving average with a momentum of m:

$$K^t = mK^{t-1} + (1-m)k^t, \tag{10}$$

where we denote by K^t a running TR. Motivated by the running statistics in *e.g.*, batch normalization [19], we use the momentum estimator to obtain the running TR, which roughly averages the TRs over recent training iterations, instead of using the TR, k^t in Eq. (5), directly. This allows us to use a stable statistic of the TR, and alleviates the influence from outliers. We then adjust a TALR based on the running TR K^t and a target one:

$$U^t = \max\left(0, U^{t-1} + \eta\left(R^t - K^t\right)\right), \tag{11}$$

where we denote by U^t and R^t the TALR and the target TR at the iteration step t, and η is a hyperparameter controlling the extent of the TALR update. Note that we can schedule the target TR R^t using typical schedulers (*e.g.*, step decay), which is analogous to the LR scheduling technique. With the TALR U^t at hand, we update the latent weights $\mathbf{w}^t$ as follows:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - U^t\mathbf{g}^t, \tag{12}$$

where $\mathbf{g}^t$ is a gradient term computed depending on the type of an optimizer (*e.g.*, the first moment of gradients in SGD). Updating the latent weights $\mathbf{w}^t$ with the TALR U^t enables controlling the running TR of quantized weights K^t w.r.t the target TR R^t. For example, if a current running TR K^t is smaller than the target one R^t, the TALR U^t increases according to Eq. (11). The latent weights in Eq. (12) are then updated largely, compared to the previous iteration. This encourages more latent weights to pass transition points of a quantizer, which in turn raises the TR in the next step. Similarly, in the opposite case, the TALR decreases to reduce the TR. Note that one can adjust the TALR in a different way from Eq. (11) while achieving the same effect, and we discuss the variants of update algorithms for TALR in the Sec. S3.2 of the supplement. Our approach connects the latent and quantized weights, in contrast to conventional optimization methods, making it possible to control an average effective step size of quantized weights via scheduling a target TR.

4.3. Quantization scheme

We apply the TR scheduler to QAT with various bit-width settings, including binary and multi-bit representations. In the following, we describe quantization schemes used in our experiments.

Multi-bit quantization. We modify LSQ [9], the state-of-the-art method for multi-bit uniform quantization[1]. We define our b-bit quantizer as follows:

$$\mathbf{x}_q = \frac{1}{\gamma}\left\lceil \text{clip}\left(\frac{\gamma\mathbf{x}}{s}, \alpha, \beta\right)\right\rfloor, \tag{13}$$

where $\mathbf{x}_q$ is an output of the quantizer. We denote by $\mathbf{x}$ an input to the quantizer, which can be either latent weights or input activations. $\text{clip}(\cdot, \alpha, \beta)$ is a clipping function with lower and upper bounds of α and β, respectively, and $\lceil\cdot\rfloor$ is a round function. Following LSQ, we employ a learnable scale parameter s for each quantizer, adjusting the range of quantization interval[2]. We set the bit-specific constants $(\alpha,\ \beta,\ \gamma)$ as $\left(-2^{b-1},\ 2^{b-1}-1,\ 2^{b-1}\right)$ and $\left(0,\ 2^b-1,\ 2^b\right)$ for weight and activation quantizers, respectively. We do not perform a post-scaling with the learnable scale parameter s after the round function in contrast to LSQ. That is, we fix the output range of a quantizer, enforcing the output of the quantizer $\mathbf{x}_q$ to be fixed-point numbers, regardless of the range of an input $\mathbf{x}$, which is more suitable for hardware implementation. Note that the scale difference between the input and output of a quantizer does not matter if each convolutional/fully-connected layer is followed by a normalization layer (*e.g.*, batch normalization [19]), imposing the scale invariance after every quantized layer [13, 14]. This ensures that post-scaling does not affect either the forward or backward pass. When the normalization is not used, we optionally apply a learnable post-scaling technique to outputs of convolutional/fully-connected layers [25].

Binary quantization. We apply two binarization methods. First, we use the network architecture of ReActNet [31] and its quantization scheme, which is the state of the art on binary quantization. ReActNet modifies the ResNet [11] or MobileNet-V1 [15] architectures by adopting the Bi-Real structure [30] that adds more residual connections, while exploiting real-valued 1×1 convolutions in

[1] Using the same network architecture (*i.e.*, a vanilla version of ResNet), our modifications provide similar or better baseline results on ImageNet [7], compared to the performance of LSQ, reproduced in [3].

[2] We train scale parameters in activation quantizers only, and do not train them in weight quantizers, when the TR scheduling technique is adopted. Otherwise, transitions could occur, even when the latent weights are not updated. For a fair comparison, we use learnable scale parameters for weight quantizers, when using plain optimizers without TR scheduling. See the Sec. S5.2 of the supplement for details.

Table 1. Quantitative comparison of quantized models on ImageNet [7] in terms of a top-1 validation accuracy. We train quantized models with plain optimization methods (SGD and Adam [22]) or ours using a TR scheduler (SGDT and AdamT). The bit-widths of weights (W) and activations (A) are represented in the form of W/A. For comparison, we report the performance of full-precision (FP) and activation-only binarized (W32A1) models. The results of ReActNet-18 [31] for the plain optimizers are reproduced with an official source code.

Optimizer	MobileNetV2 (FP: 71.9)			ReActNet-18 (W32A1: 66.8)	ResNet-18 (FP: 69.9)			
	2/2	3/3	4/4	1/1	1/1	2/2	3/3	4/4
SGD	46.9	65.6	69.9	65.0	55.3	66.8	69.5	70.5
SGDT	**53.6**	**67.0**	**70.5**	**65.3**	**55.8**	**66.9**	**69.7**	**70.6**
Adam	49.6	66.5	70.0	65.3	56.1	66.7	69.5	70.1
AdamT	**53.8**	**67.3**	**70.8**	**65.7**	**56.3**	**67.2**	**69.7**	**70.4**

Table 2. Quantitative comparison of quantized models on CIFAR-100/10 [24] in terms of a top-1 test accuracy.

Optimizer	CIFAR-100			CIFAR-10		
	ReActNet-18 (W32A1: 69.6)	ResNet-20 (FP: 65.1)		ReActNet-18 (W32A1: 91.3)	ResNet-20 (FP: 91.1)	
	1/1	1/1	2/2	1/1	1/1	2/2
SGD	69.7	54.9	64.1	90.9	85.2	90.2
SGDT	**72.2**	**55.8**	**65.5**	**93.0**	**85.6**	**90.7**
Adam	69.5	54.8	63.3	90.4	84.8	90.2
AdamT	**71.8**	**55.9**	**65.2**	**92.9**	**85.7**	**91.1**

Table 3. Quantitative comparison of quantized models on ImageNet [7] in terms of a top-1 validation accuracy. We train quantized models with plain optimization method (AdamW [33]) or ours using a TR scheduler (AdamWT).

Optimizer	DeiT-T (FP: 72.0)		DeiT-S (FP: 79.9)	
	2/2	3/3	2/2	3/3
AdamW	54.6	68.1	68.4	77.6
AdamWT	**57.4**	**69.5**	**71.8**	**78.5**

the residual connections. This approach also uses learnable shift operations before quantization and activation functions. Second, we binarize vanilla ResNet models to compare binary and multi-bit quantization schemes under a fair training setting. To this end, we design a binary quantizer using Eq. (13). For a weight quantizer, we set α, β, and γ, to -1, 1, and 1, respectively, and replace the round operator with a signum function to obtain a binary value of -1 or 1. For an activation quantizer, we set those values as 0, 1, and 1, respectively, to generate a binary activation of 0 or 1.

5. Experiments

We describe our experimental settings (Sec. 5.1) and show results on image classification and object detection (Sec. 5.2). We then analyzze the TR scheduling technique (Sec. 5.3). More detailed analyses and discussions are provided in the supplement.

Table 4. Quantitative results on object detection. We train RetinaNet [29] on the training split of MS COCO [28] using either the plain optimization method (SGD) or ours (SGDT). We report the average precision (AP) on the validation split.

Backbone	W/A	Optimizer	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
ResNet-50	FP	SGD	37.80	57.62	40.50	23.12	41.39	49.70
	4/4	SGD	38.05	57.75	40.23	**22.51**	41.46	49.68
		SGDT	**38.36**	**58.01**	**40.76**	22.46	**41.87**	**49.71**
	3/3	SGD	37.32	56.87	39.71	**21.90**	40.82	48.97
		SGDT	**37.59**	**56.89**	**40.18**	21.51	**40.98**	**49.07**

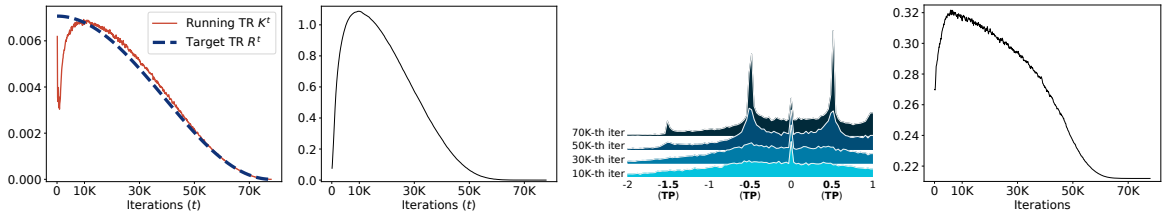
5.1. Experimental settings

For image classification, we train quantized models for MobileNetV2 [37], ResNet families [11], ReActNet-18 [31], and DeiT-T/S [43] on CIFAR-10/100 [24] and/or ImageNet [7]. We train them using a cross-entropy loss, except for ReActNet-18 on ImageNet, where we use a distributional loss [31] following the work of [31]. For object detection, we adopt RetinaNet [29] with ResNet backbones on MS COCO [28]. Unlike the previous QAT methods [45, 49], we use a shared prediction head to handle features of different resolutions, analogous to the original RetinaNet [29]. For ease of activation quantization, we add a ReLU layer after each convolutional layer in the prediction head, so that all inputs of activation quantizers are non-negative. For more details, please refer to the Sec. S4.1 of the supplement.

While our method requires additional computations (*i.e.*, element-wise comparison in Eq. (5) and scalar operations in Eqs. (10)-(11)), they are computationally cheap compared to the whole training process. The training time increases by only 2% compared to the plain optimization methods with the same machine (Sec. S2.5 of the supplement).

5.2. Results

Image classification. We provide in Tables 1-3 quantitative comparisons of quantized models trained with optimizers using plain optimization methods and our approach. We report a top-1 classification accuracy on ImageNet [7] and CIFAR-100/10 [24] using the MobileNetV2 [37], ReActNet-18 [31], ResNet-18/20 [11], and DeiT-T/S [43] architectures. From these tables, we observe three things: (1) Our method provides substantial accuracy gains over the plain optimizers, regardless of the datasets, network architectures, and quantization bit-widths. This indicates that scheduling a target TR is a better choice for the optimization process in QAT compared to the conventional strategy scheduling a LR. (2) The performance gaps on ImageNet using light-weight MobileNetV2 (0.6∼6.7%) are more significant than the ones using ReActNet-18 or ResNet-18 (0.1∼0.5%). Moreover, the performance gaps become larger for smaller bit-widths of MobileNetV2. These results suggest that the TR scheduling technique is especially useful for compressing networks aggressively, such as quantiz-



(a) Running TR K^t and target TR R^t.

(b) TALR U^t.

(c) Distributions of normalized latent weights.

(d) Average distances to the nearest transition points.

Figure 3. Analysis on TR scheduling. We train ResNet-20 [11] on CIFAR-100 [24] using SGDT, where we quantize both weights and activations with 2-bit representations. We visualize distributions of normalized latent weights in the 16$^{\text{th}}$ layer in (c), and average distances between normalized latent weights and the nearest transition points in (d). The transition points in (c) are denoted by TPs in the x-axis. The top-1 test accuracy and average effective step sizes of quantized weights are shown by the red curves in Figs. 1d and 1c, respectively.

ing a light-weight model or extremely low-bit quantization. (3) Considering the results for ReActNet-18 and the ResNet families, our approach outperforms the conventional optimization methods by significant margins (0.4∼2.5%) on the small dataset (*i.e.*, CIFAR-100/10). On the large-scale dataset (*i.e.*, ImageNet), it also shows superior results, achieving 0.1∼0.5% accuracy gains. The overall performance gaps decrease on ImageNet, possibly because the plain optimizers with a gradually decaying LR (*e.g.*, cosine annealing LR [32]) benefit from lots of training iterations on ImageNet (roughly 600K). They, however, do not show satisfactory results within a small number of iterations on CIFAR-100/10 (roughly 80K), compared to ours.

Object detection. We compare in Table 4 the quantization performance of detection models in terms of an average precision (AP) on the validation split of MS COCO [28]. We train RetinaNet [29] with the ResNet-50 [11] backbone using either SGD or SGDT on the training split of MS COCO. We can observe in Table 4 that the TR scheduling technique boosts the AP consistently over the SGD baselines across different bit-widths, similar to the results on image classification. This suggests that the TR scheduling technique is also useful for the object detection task involving both regression and classification, demonstrating once more the effectiveness of our method and its generalization ability to various tasks. Additional quantitative results on object detection with different backbone networks (*e.g.*, ResNet-18/34) and qualitative results are provided in the Sec. S1.2 of the supplement.

5.3. Analysis

We show in Fig. 3 an in-depth analysis on how a TR scheduler works during QAT. We can see from Fig. 3a that the running TR K^t roughly follows the target TR R^t, indicating that we can control the average effective step size of quantized weights (the red curve in Fig. 1c) by scheduling the target TR. This is possible because the TALR U^t is adjusted adaptively to match the running TR K^t with the target one R^t (Fig. 3b). We can see that the TALR U^t increases initially, since the running TR K^t is much smaller than the target TR R^t. The TALR U^t then decreases gradually to reduce the number of transitions, following the target TR R^t. Note that the TALR U^t approaches zero rapidly near the 50K-th iteration. To figure out the reason, we show in Figs. 3c and 3d distributions of normalized latent weights and their average distances to the nearest transition points, respectively. We can observe in Fig. 3c that latent weights tend to be concentrated near the transition points of a quantizer as the training progresses, similar to the case in Sec. 4.1 using a user-defined LR. This implies that transitions occur more frequently in later training iterations if we do not properly reduce the degree of parameter change for latent weights. In particular, we can see in Fig. 3d the average distances between the normalized latent weights and the nearest transition points are relatively small after the 50K-th iteration. Under such circumstance, the TALR should become much smaller in order to reduce the running TR, as in the sharp decline around the 50K-th iteration. We can thus conclude that our approach adjusts the TALR by considering the distribution of the latent weights implicitly.

6. Conclusion

We have discussed the problem of a conventional optimization method using a LR in QAT. To overcome this, we have presented a TR scheduling technique specialized for general QAT, which controls the number of transitions in quantized weights explicitly. We have shown that the optimizers coupled with our TR scheduling technique outperform the plain ones using a LR by significant margins under various QAT settings. We have also verified that our approach enables stable training with different types of optimizers and schedulers, which is generally difficult in the conventional optimization methods, indicating that the TR scheduling technique offers more diverse training options for QAT. We expect that our method would be adopted to boost the performance of other quantized models, bringing a significant advance in the field of network quantization.

Acknowledgements

This work was supported in part by NRF and IITP grants funded by the Korea government (MSIT) (No. 2023R1A2C2004306, No.RS-2022-00143524, Development of Fundamental Technology and Integrated Solution for Next-Generation Automatic Artificial Intelligence System, RS-2021-II212068, Artificial Intelligence Innovation Hub) and the Yonsei Signature Research Cluster Program of 2025 (2025-22-0013).

References

[1] Yu Bai, Yu-Xiang Wang, and Edo Liberty. ProxQuant: Quantized neural networks via proximal operators. In *Int. Conf. Learn. Represent.*, 2019. 1, 3

[2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv*, 2013. 2, 4

[3] Yash Bhalgat, Jinwon Lee, Markus Nagel, Tijmen Blankevoort, and Nojun Kwak. LSQ+: Improving low-bit quantization through learnable offsets and better initialization. In *IEEE Conf. Comput. Vis. Pattern Recog. Workshops*, pages 696–697, 2020. 6

[4] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5918–5926, 2017. 1, 2

[5] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. PACT: Parameterized clipping activation for quantized neural networks. *arXiv*, 2018. 1, 2, 4

[6] Alexandre Défossez, Yossi Adi, and Gabriel Synnaeve. Differentiable model compression via pseudo quantization noise. *TMLR*, 2022. 3

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 248–255, 2009. 2, 6, 7

[8] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12(7), 2011. 3

[9] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. In *Int. Conf. Learn. Represent.*, 2020. 1, 2, 4, 6

[10] 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 *Int. Conf. Comput. Vis.*, pages 4852–4861, 2019. 1, 2

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 770–778, 2016. 1, 2, 4, 6, 7, 8

[12] Koen Helwegen, James Widdicombe, Lukas Geiger, Zechun Liu, Kwang-Ting Cheng, and Roeland Nusselder. Latent weights do not exist: Rethinking binarized neural network optimization. In *Adv. Neural Inform. Process. Syst.*, pages 7533–7544, 2019. 3

[13] Byeongho Heo, Sanghyuk Chun, Seong Joon Oh, Dongyoon Han, Sangdoo Yun, Gyuwan Kim, Youngjung Uh, and Jung-Woo Ha. AdamP: Slowing down the slowdown for momentum optimizers on scale-invariant weights. In *Int. Conf. Learn. Represent.*, 2021. 4, 5, 6

[14] Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. Norm matters: efficient and accurate normalization schemes in deep networks. In *Adv. Neural Inform. Process. Syst.*, 2018. 4, 5, 6

[15] 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*, 2017. 6

[16] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 7132–7141, 2018. 1

[17] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. In *Int. Conf. Learn. Represent.*, 2017. 1, 3, 4

[18] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 4700–4708, 2017. 1

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Int. Conf. Mach. Learn.*, pages 448–456, 2015. 4, 5, 6

[20] 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 *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 4350–4359, 2019. 1, 2, 4

[21] Dohyung Kim, Junghyup Lee, and Bumsub Ham. Distance-aware quantization. In *Int. Conf. Comput. Vis.*, pages 5271–5280, 2021. 1, 2

[22] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Int. Conf. Learn. Represent.*, 2014. 1, 2, 3, 5, 7

[23] Bobby Kleinberg, Yuanzhi Li, and Yang Yuan. An alternative view: When does SGD escape local minima? In *Int. Conf. Mach. Learn.*, pages 2698–2707, 2018. 1, 3, 4

[24] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, 2009. 2, 4, 7, 8

[25] Junghyup Lee, Dohyung Kim, and Bumsub Ham. Network quantization with element-wise gradient scaling. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 6448–6457, 2021. 1, 3, 5, 6

[26] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv*, 2016. 1, 2

[27] Yuanzhi Li, Colin Wei, and Tengyu Ma. Towards explaining the regularization effect of initial large learning rate in training neural networks. In *Adv. Neural Inform. Process. Syst.*, 2019. 3

[28] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Eur. Conf. Comput. Vis.*, pages 740–755, 2014. 2, 7, 8

[29] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Int. Conf. Comput. Vis.*, pages 2980–2988, 2017. 2, 7, 8

[30] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-Real Net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *Eur. Conf. Comput. Vis.*, pages 722–737, 2018. 6

[31] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. ReActNet: Towards precise binary neural network with generalized activation functions. In *Eur. Conf. Comput. Vis.*, pages 143–159, 2020. 1, 2, 6, 7

[32] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *Int. Conf. Learn. Represent.*, 2017. 3, 8

[33] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Int. Conf. Learn. Represent.*, 2019. 2, 7

[34] Markus Nagel, Marios Fournarakis, Yelysei Bondarenko, and Tijmen Blankevoort. Overcoming oscillations in quantization-aware training. In *Int. Conf. Mach. Learn.*, 2022. 3, 5

[35] Eunhyeok Park and Sungjoo Yoo. PROFIT: A novel training method for sub-4-bit MobileNet models. In *Eur. Conf. Comput. Vis.*, pages 430–446, 2020. 5

[36] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet classification using binary convolutional neural networks. In *Eur. Conf. Comput. Vis.*, pages 525–542, 2016. 1, 2

[37] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 4510–4520, 2018. 2, 7

[38] Pedro Savarese, Xin Yuan, Yanjing Li, and Michael Maire. Not all bits have equal value: Heterogeneous precisions via trainable noise. In *NeurIPS*, 2022. 3

[39] Juncheol Shin, Junhyuk So, Sein Park, Seungyeop Kang, Sungjoo Yoo, and Eunhyeok Park. Nipq: Noise proxy-based integrated pseudo-quantization. In *CVPR*, 2023. 3

[40] Cuauhtemoc Daniel Suarez-Ramirez, Miguel Gonzalez-Mendoza, Leonardo Chang, Gilberto Ochoa-Ruiz, and Mario Alberto Duran-Vega. A bop and beyond: A second order optimizer for binarized neural networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1273–1281, 2021. 3

[41] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 2818–2826, 2016. 1

[42] T. Tieleman and G. Hinton. Lecture 6.5 - RMSProp: Divide the gradient by a running average of its recent magnitude, COURSERA: Neural Networks for Machine Learning. Technical report, 2012. 3

[43] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*, 2021. 2, 7

[44] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1492–1500, 2017. 1

[45] Kohei Yamamoto. Learnable companding quantization for accurate low-bit neural networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5029–5038, 2021. 1, 2, 7

[46] Jiwei Yang, Xu Shen, Jun Xing, Xinmei Tian, Houqiang Li, Bing Deng, Jianqiang Huang, and Xian-sheng Hua. Quantization networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 7308–7316, 2019. 1, 2

[47] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. LQ-Nets: Learned quantization for highly accurate and compact deep neural networks. In *Eur. Conf. Comput. Vis.*, pages 365–382, 2018. 1, 2

[48] 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*, 2016. 1, 2, 4

[49] Bohan Zhuang, Lingqiao Liu, Mingkui Tan, Chunhua Shen, and Ian Reid. Training quantized neural networks with a full-precision auxiliary module. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1488–1497, 2020. 7