Scheduling Weight Transitions for Quantization-Aware Training - Supplementary Material

In the supplement, we give additional results in image classification and object detection (Sec. S1). We then provide in-depth analyses (Sec. S2) and discussions on our method (Sec. S3). Finally, we present the implementation details of our method (Sec. S4) and comprehensive description of a quantizer (Sec. S5). We summarize in Algorithm 1 an overall process of our method.

S1. Results

S1.1. Image classification

Comparison to the state of the art We present in Table S1 a quantitative comparison of our approach and state-of-the-art methods for QAT [2, 3, 7, 8, 12, 17, 23, 29] on ImageNet [4], in terms of a top-1 validation accuracy. For a fair comparison, we perform comparisons with the methods using uniform quantization schemes and a vanilla architecture. We can see from the table that our method (*i.e.*, using either SGDT or AdamT) achieves state-of-the-art performance for ResNet-18 under the 1/1 and 2/2 bit-width settings, and shows competitive performance for the 3/3 and 4/4 bit-width settings. In terms of performance gains or drops compared to the full-precision models, our results are better than or on a par with the state of the art. For MobileNetV2, our method also sets a new state of the art, except for the 3/3 bit-width setting. The work of [23] alleviates the oscillation problem in QAT by freezing latent weights or adding a regularization term. It adopts architecture-specific and bit-specific hyperparameters, such as a LR, a weight decay, and a threshold for weight freezing. This is because freezing weights and adding the regularizer could potentially degrade trainability and disturb the training process, making the QAT process sensitive to hyperparameters. In contrast, our method is more practical as it performs consistently well under the various bit-width settings with the same set of hyperparameters. For completeness, we also try to tune a TR factor λ specific to the 3/3-bit setting of MobileNetV2, and improve the performance with the TR factor λ of 2e-3.

Other QAT methods mostly focus on designing a quantizer in a forward step (*e.g.*, PACT [3], QIL [12], LSQ [7], LSQ+ [2]) or addressing a gradient mismatch problem in a backward step (*e.g.*, QNet [29], DSQ [8], EWGS [17]). Our approach is orthogonal to these methods in that we focus on an optimization step in QAT. Namely, we propose to schedule a target TR instead of a LR itself, and introduce a novel TALR to update latent weights. Compared to other methods that require computationally expensive operations for differentiable quantizers (*e.g.*, QNet[29], DSQ [8]) or approximating a hessian trace (*e.g.*, EWGS [17]), ours is relatively simple and efficient, since it adds few comparison and scalar operations only in the QAT process.

Algorithm 1 Optimization process using a TR scheduler.

Require: the number of iterations T; a target TR R^t; a momentum constant m.

```
1: while $t < T$ do
2:   Compute a current TR $k^t$ [Eq. (5)]:
       $k^t \leftarrow \frac{\sum_{i=1}^{N} \mathbb{I}\left[w_d^t(i) \neq w_d^{t-1}(i)\right]}{N}$.
3:   Estimate a running TR $K^t$ [Eq. (10)]:
       $K^t \leftarrow mK^{t-1} + (1-m)k^t$.
4:   Adjust a TALR $U^t$ [Eq. (11)]:
       $U^t \leftarrow \max\left(0, U^{t-1} + \eta\left(R^t - K^t\right)\right)$.
5:   Update latent weights $\mathbf{w}^t$ with a gradient term $\mathbf{g}^t$
     and the TALR $U^t$ [Eq. (12)]:
       $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t - U^t\mathbf{g}^t$.
6: end while
```

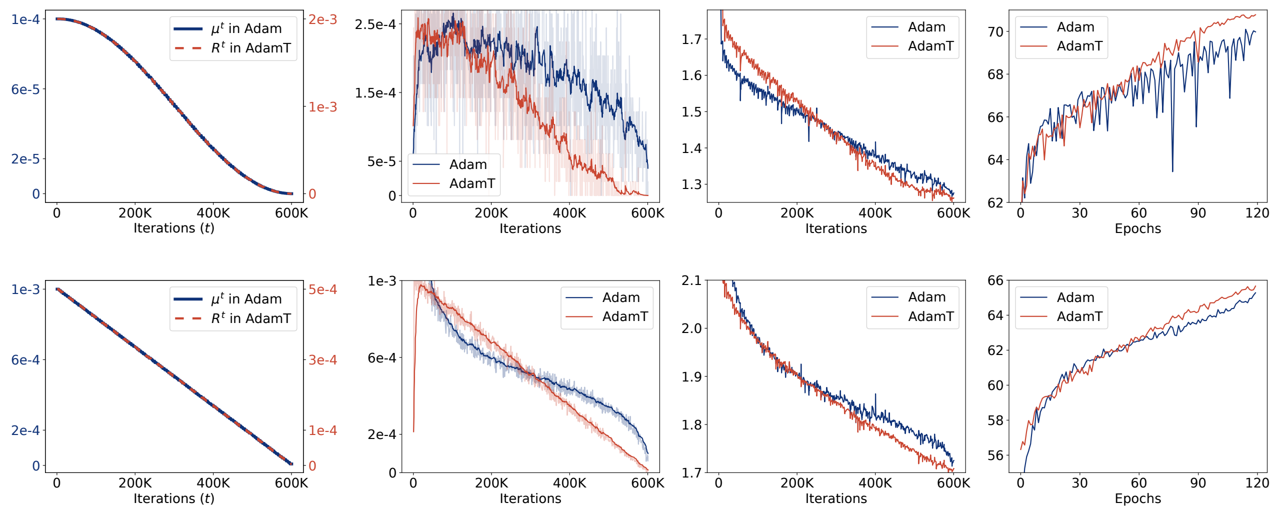
Quantitative results. We show in Fig. S1 training curves on ImageNet for MobileNetV2 and ReActNet-18 using 4-bit and binary weights/activations, respectively. We can see from Figs. S1a and S1b that the optimizer coupled with our TR scheduling technique (*i.e.*, AdamT) can control the average effective step size of quantized weights roughly using a target TR. On the other hand, the plain optimizer adopting a user-defined LR (*i.e.*, Adam) fails to control it by scheduling the LR. For example, the average effective step sizes for quantized weights in MobileNetV2 (*i.e.*, the blue curve in the first row of Fig. S1b) do not approach zero at the end of training. Such large changes in the quantized weights can lead to unstable batch normalization statistics [11], resulting in noisy and degraded test time performance [23, 24] (*i.e.*, the blue curve in the first row of Fig. S1d). On the contrary, the TR scheduling technique enables better training in terms of the convergence rate and performance, which can be verified by the training losses and validation accuracy in Figs. S1c and S1d, respectively.

S1.2. Object detection

Quantitative results. We compare in Table S2 the quantization performance of detection models in terms of an

Table S1. Quantitative comparison of our method and the state of the art on ImageNet [4] in terms of a top-1 validation accuracy. The numbers in parentheses indicate the performance gains or drops compared to the full-precision (32/32) models. All numbers are taken from corresponding works except for LSQ [7], where the results are reproduced in [2] using a vanilla structure of ResNet [9].

Architecture	Methods	Optimizer	Weight/activation bit-widths				
			1/1	2/2	3/3	4/4	32/32
ResNet-18	PACT [3]	SGD	-	64.4 (−5.8)	68.1 (−2.1)	69.2 (−1.0)	70.2
	QIL [12]	SGD	-	65.7 (−4.5)	69.2 (−1.0)	70.1 (−0.1)	70.2
	QNet [29]	SGD	53.6 (−16.7)	-	-	-	70.3
	DSQ [8]	-	-	65.2 (−4.7)	68.7 (−1.2)	69.6 (−0.3)	69.9
	LSQ [7]	SGD	-	66.7 (−3.4)	$\underline{69.4}$ ($\underline{-0.7}$)	$\underline{70.7}$ ($\underline{+0.6}$)	70.1
	LSQ+ [2]	SGD	-	66.8 (−3.3)	69.3 (−0.8)	**70.8** (+**0.7**)	70.1
	EWGS [17]	SGD	55.3 (−14.6)	$\underline{67.0}$ ($\underline{-2.9}$)	**69.7** (−**0.2**)	70.6 (+**0.7**)	69.9
	Ours	SGDT	$\underline{55.8}$ ($\underline{-14.1}$)	66.9 (−3.0)	**69.7** (−**0.2**)	70.6 (+**0.7**)	69.9
	Ours	AdamT	**56.3** (−**13.6**)	**67.2** (−**2.7**)	**69.7** (−**0.2**)	70.4 (+0.5)	69.9
MobileNetV2	DSQ [8]	-	-	-	-	64.8 (−7.1)	71.9
	EWGS [17]	SGD	-	-	-	70.3 (−1.6)	71.9
	Nagel *et al.* [23]	SGD	-	-	$\underline{67.6}$ (−**4.1**)	$\underline{70.6}$ (−**1.1**)	71.7
	Ours	SGDT	-	$\underline{53.6}$ ($\underline{-18.3}$)	67.0 (−4.9)	70.5 ($\underline{-1.4}$)	71.9
	Ours	AdamT	-	**53.8** (−**18.1**)	67.3 ($\underline{-4.6}$)	**70.8** (−**1.1**)	71.9
	Ours (λ=2e-3)	AdamT	-	-	**67.8** (−**4.1**)	-	71.9



(a) LR μ and target TR R^t.

(b) Avg. effective step sizes of quantized weights.

(c) Running avg. of training losses.

(d) Top-1 validation accuracy.

Figure S1. Training curves for quantized models using Adam [14] and AdamT on ImageNet [4]. The results in the first and second rows are obtained with MobileNetV2 [26] and ReActNet-18 [20] using 4-bit and binary weights/activations, respectively. For the visualizations in (b), we monitor the average effective step sizes of quantized weights in the 19th and 17th layers of MobileNetV2 and ReActNet-18, respectively. (Best viewed in color.)

average precision (AP) on the validation split of MS COCO [18]. We train RetinaNet [19] with the ResNet-18/34 [9] backbones using either SGD or SGDT on the training split of MS COCO. From Table 4 of the main paper and Table S2, we can see that using a TR scheduling technique improves the performance of quantized models in terms of AP, compared to the plain optimization method, across different bit-width and backbone settings. This confirms once again the effectiveness and generalization ability of our approach.

Qualitative results. We compare in Fig. S2 qualitative results of baseline (SGD) and our (SGDT) models, using RetinaNet with the ResNet-50 backbone, with 4-bit

Table S2. Quantitative comparison of quantized models on object detection. We train RetinaNet [19] on the training split of MS COCO [18] with different backbone networks and quantization bit-widths, 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-34	FP	SGD	37.70	57.32	40.71	22.35	41.50	48.93
	4/4	SGD	37.99	57.29	40.34	**22.96**	41.16	49.32
		SGDT	**38.14**	**57.39**	**40.93**	22.05	**41.70**	**49.79**
	3/3	SGD	37.32	56.61	39.95	**22.22**	**40.54**	49.16
		SGDT	**37.64**	**56.78**	**40.28**	21.81	40.42	**49.89**
ResNet-18	FP	SGD	34.06	53.15	36.17	19.66	36.48	44.45
	4/4	SGD	35.09	54.05	37.40	**20.34**	**37.62**	46.69
		SGDT	**35.30**	**54.46**	**37.59**	19.92	37.61	**46.83**
	3/3	SGD	34.32	53.09	36.44	19.34	36.46	45.33
		SGDT	**34.72**	**53.63**	**36.84**	**19.91**	**37.21**	**45.64**

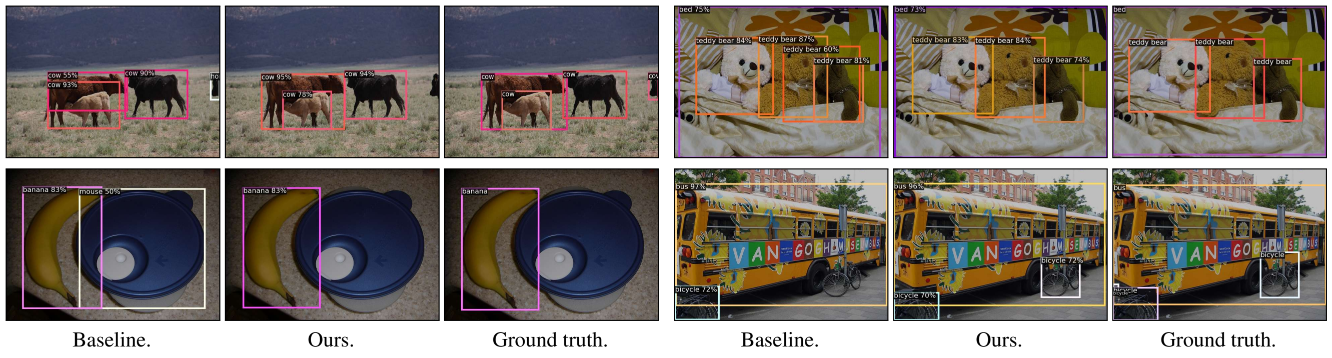


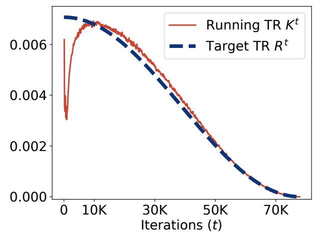
Figure S2. Qualitative comparison for object detection on MS COCO [18] using RetinaNet [19] with the ResNet-50 [9] backbone, where both weights and activations are quantized into 4-bit. The results of baseline and ours are obtained from the models trained with SGD and SGDT, respectively. (Best viewed in color.)

weights/activations. The baseline model trained with SGD provides incorrect bounding boxes in the presence of overlapped/adjacent instances (*e.g.*, the cow and teddy bear in the first and second examples, respectively), misclassifies object classes (*e.g.*, the food container in the third example), and fails to detect objects (*e.g.*, the bicycle in the last example). In contrast, the model trained with our approach offers accurate bounding boxes, and classifies object classes successfully.

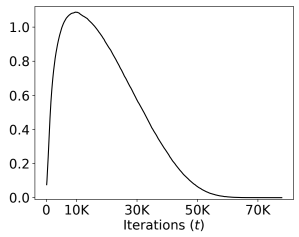
S2. Analysis

S2.1. Analysis on TR scheduling

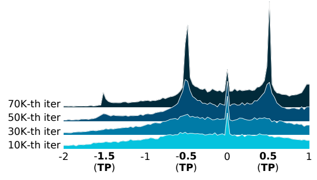
We show in Fig. S3 an in-depth analysis on how a TR scheduler works during QAT. We can see from Fig. S3a that the running TR K^t roughly follows the target TR R^t, indicating that we can control the average effective step size of quantized weights (the red curve in Fig. 1c of the main paper) by scheduling 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. S3b). 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. S3c and S3d distributions of normalized latent weights and their average distances to the nearest transition points, respectively. We can observe in Fig. S3c 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 in the main paper 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. S3d the average distances between the normalized latent weights and the nearest transition points are relatively small after the 50K-th iteration.



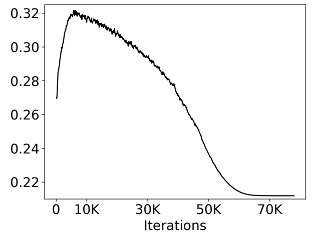
(a) Running TR K^t and target TR R^t.



(b) TALR U^t.



(c) Distributions of normalized latent weights.



(d) Average distances to the nearest transition points.

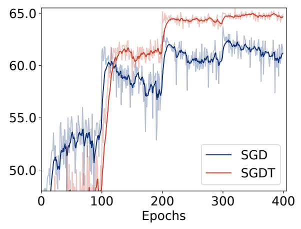
Figure S3. Analysis on TR scheduling. We train ResNet-20 [9] on CIFAR-100 [16] using SGDT, where we quantize both weights and activations with 2-bit representations. We visualize distributions of normalized latent weights in the 16th layer in (c), and average distances between normalized latent weights 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 of the main paper, respectively. (Best viewed in color.)

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.

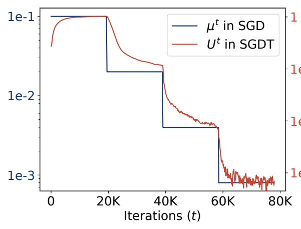
S2.2. Comparison using a step decay scheduler

We provide in Table S3 a quantitative comparison between plain optimization methods (SGD and Adam [14]) and ours (SGDT and AdamT) using a step scheduler. We use the same training setting in Table 2 of the main paper, while replacing a cosine scheduler [21] with the step scheduler. Both LRs and target TRs are divided by 5 after every 50 and 100 epochs for ReActNet-18 [20] and ResNet-20 [9], respectively. We can see from this table that the plain optimizers using the step scheduler suffer from significant accuracy drops, compared to the cases for the the cosine scheduler. The main reason is that it is difficult to control the average effective step size of quantized weights using a user-defined LR, especially when the LR is decayed by the step scheduler. On the other hand, the optimizers using our TR scheduling technique are relatively robust to the step scheduler, showing less performance drops than the plain ones for most cases. We can also observe that the performance degrades more severely for the models using the ResNet-20 [9] architecture, where the number of parameters is much less than that of the ReActNet-18 [20] architecture (276K for ResNet-20 vs. 11,235K for ReActNet-18). This indicates that training a small quantized model with the step scheduler is more challenging, possibly because the scheduler can cause large parameter changes even at the end of the training.

To further analyze the difference between the LR and TR scheduling techniques with the step scheduler, we compare in Fig. S4 the LR in SGD and TALR in SGDT during training, with corresponding test accuracies. We can observe from Fig. S4a that SGDT provides better results in terms of both convergence rate and accuracy. As discussed in Sec. 4.1 of the main paper, latent weights approach transition points progressively during QAT, and thus it is difficult to adjust the number of transitions explicitly using a user-defined LR. This could be particularly problematic when the LR is fixed for a number of iterations. Recent QAT methods [7, 13, 17, 20, 28] circumvent this issue by decaying a LR to zero gradually, *e.g.*, using the cosine annealing technique [21], but this does not fully address the problem. On the other hand, our method is relatively robust to the use of the step scheduler. Since we optimize latent weights using a TALR, we can control the degree of parameter changes for quantized weights by scheduling the target TR in our method. In contrast to the LR, the TALR decreases gradually even when the scheduler does not alter the target TR (Fig. S4b), confirming once more the fact that the TALR is adjusted adaptively considering the distribution of latent weights. This result demonstrates the robustness and



(a) Test accuracy (%).



(b) LR μ^t and TALR U^t.

Figure S4. Comparison between SGD and SGDT using a step scheduler. We train ResNet-20 [9] with 2-bit weights and activations on CIFAR-100 [16]. Both the LR for SGD and target TR for SGDT are divided by 5 after every 100 epochs. For comparison, we show in (b) the LR μ^t in SGD and TALR U^t in SGDT, where both are used for updating latent weights in Eqs. (4) and (12) of the main paper. (Best viewed in color.)

Table S3. Quantitative comparison of optimization methods for QAT using a step decay scheduler on CIFAR-100/10 [16]. Both LRs and target TRs for SGD and SGDT are divided by 5 after every 50 and 100 epochs for ReActNet-18 [20] and ResNet-20 [9], respectively. Numbers in parentheses indicate accuracy drops compared to the models in Table 2 of the main paper trained with a cosine scheduler.

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	69.0 (−0.7)	45.8 (−9.1)	61.3 (−2.8)	90.9 (**−0.0**)	82.9 (−2.3)	89.8 (**−0.4**)
SGDT	**71.9** (**−0.3**)	**55.3** (**−0.5**)	**64.9** (**−0.6**)	**93.0** (**−0.0**)	**85.2** (**−0.4**)	**90.3** (**−0.4**)
Adam	68.1 (−1.4)	49.9 (−4.9)	60.9 (−2.4)	82.5 (−7.9)	82.4 (−2.4)	89.6 (**−0.6**)
AdamT	**71.7** (**−0.1**)	**54.4** (**−1.5**)	**64.6** (**−0.6**)	**92.9** (**−0.0**)	**85.3** (**−0.4**)	**90.3** (−0.8)

Table S4. Quantitative comparison of different TR factors λ (*i.e.*, initial target TRs). We quantize ResNet-20 [9] using SGDT with 2-bit weights/activations, and report a top-1 test accuracy on CIFAR-100 [16].

TR factor λ	1e-3	2e-3	3e-3	4e-3	5e-3	6e-3	7e-3	8e-3	9e-3	1e-2
Test accuracy	62.5	64.2	64.3	65.3	65.5	65.1	63.1	63.6	63.6	64.0

Table S5. Quantitative comparison for plain optimization methods and our TR scheduling technique with various optimizers. We report a top-1 test accuracy on CIFAR-100 [16] using ResNet-20 [9] with 2-bit weights/activations.

	SGD	Adam	NAdam	Adamax	AdamW	RMSProp	Adagrad
Plain	64.1	63.3	63.3	62.4	63.8	64.6	54.2
Ours	65.5	65.2	65.1	64.7	66.2	65.1	61.1

generalization ability of our approach on various types of schedulers.

S2.3. Analysis on initial target TRs

To better understand how an initial target TR influences the quantization performance, we present in Table S4 a quantitative comparison of different TR factors λ adjusting the initial TRs. We train ResNet-20 [9] models on CIFAR-100 [16] using 2-bit weights and activations with SGDT and report top-1 test accuracies. We can observe that the TR factors within a wide range of intervals (*i.e.*, [4e-3, 6e-3]) provide satisfactory performance, outperforming the LR-based optimization strategy (*i.e.*, SGD in Table 2 of the main paper) by significant margins (1.0∼1.4). Similar to the LR, extremely large or small TR factors degrade the quantization performance, which could make the training unstable. Therefore, setting an appropriate value for the TR factor is crucial in the TR scheduling technique, analogous to determining an initial LR to train full-precision models.

Table S6. Quantitative comparison of quantized models trained with SGDT using different final target TRs. We report mean and standard deviation of top-1 test accuracies on CIFAR-100 [16] over five random runs, obtained with ResNet-20 [9] using 2-bit weights and activations.

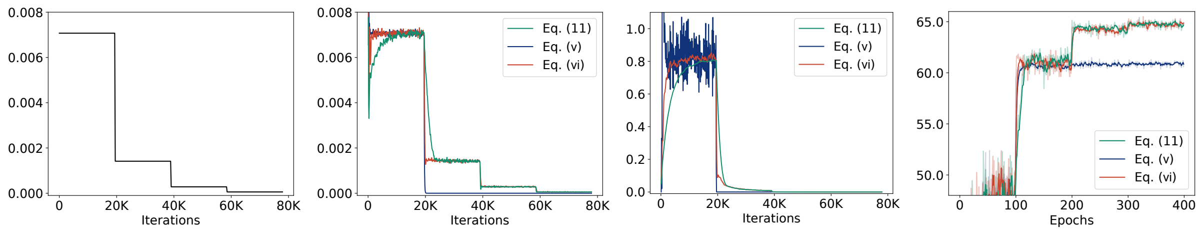
Final target TR	0	1e-5	1e-4	1e-3
Final target average effective step size	0	5e-6	5e-5	5e-4
Test accuracy	65.61 (±0.21)	65.34 (±0.26)	64.63 (±0.59)	62.12 (±0.70)

Table S7. Training time comparison on ImageNet with 4 A5000 GPUs, in terms of GPU hours. We quantize ResNet-18 except for AdamW/-T, where we quantize DeiT-T.

	SGD	Adam	AdamW
Plain	174	175	477
Ours	176	178	492

S2.4. Comparison with different optimizers

To verify the generalization ability of our TR scheduling technique to other optimizers, we compare in Table S5 the quantization performance between plain optimization methods and ours with various optimizers, including SGD, Adam [14], NAdam [5], Adamax [14], AdamW [22], RMSProp [27], and Adagrad [6]. Specifically, we train ResNet-20 on CIFAR-100 [16] with 2-bit weights and activations for comparison. For the optimizers other than SGD, we exploit the same hyperparameters as the ones for Adam in Table 2 of the main paper, while we change the weight decay to 1e-2 for AdamW. Note that AdamW decouples the weight decay from a parameter update step, suggesting that it requires a different hyperparameter for the weight decay. For RMSProp, we use the momentum gradient with a momentum value of 0.9, which is analogous to SGD. From the table, we can clearly see that the TR scheduling technique improves the quantization performance consistently across all optimizers. Our method achieves 0.5%∼6.9% gains over



(a) Target TR R^t. (b) Running TR K^t. (c) TALR U^t. (d) Test accuracy.

Figure S5. Comparison of different update schemes for a TALR. We train ResNet-20 [9] on CIFAR-100 [16] with 2-bit weights and activations, while adjusting the TALR based on different methods in Eqs. (11), (S1) and (S2). (Best viewed in color.)

the plain optimization methods, which demonstrates the effectiveness of our method and its generalization ability to various optimizers. In particular, AdamW coupled with our TR scheduling technique shows the best performance, indicating that we could further improve the quantization performance by carefully selecting an optimizer and corresponding hyperparameters.

S2.5. Training time comparison

We compare in Table S7 training time of optimization methods. Our method takes training time, similar to plain optimization methods, demonstrating its efficiency. The marginal overheads of ours is due to the computation of the TR and the adjustment of the TALR, both of which involve lightweight operations such as element-wise comparisons and scalar updates, which are computationally inexpensive compared to the overall training process.

S3. Discussion

S3.1. Importance of reducing average effective step size

Conventional optimization methods for full-precision models typically decay a LR to reduce the degree of parameter changes (*i.e.*, average effective step sizes). This prevents full-precision weights from overshooting from a local optimum, and improves the convergence of the model [10, 15]. Similarly, when training a quantized model, small average effective step sizes for quantized weights are preferred in later training iterations. Large changes in the quantized weights at the end of training could disturb convergence and degrade the quantization performance [23, 24], as shown in Figs. 1, 2, and S1. To further support this, we perform an experiment on the influence of the average effective step size of quantized weights on the quantization performance at the end of training. To this end, we train ResNet-20 [9] on CIFAR-100 [16] with 2-bit weights and activations using SGDT for different final target TRs. We report in Table S6 the mean and standard deviation of top-1 test accuracies over five random runs for each model. We can clearly see that the smaller average effective step sizes at the end of training provide better performance with less deviation, suggesting that the large step sizes in later training iterations disturb convergence. These results demonstrate once more the importance of the TR scheduling technique, since it is difficult to control the average effective step size of quantized weights, especially in later training iterations, with conventional LR scheduling in QAT.

S3.2. Comparison of update schemes for TALR

We adjust a TALR adaptively to match a current running TR with a target one. To this end, the update rule for the TALR should meet the following criteria: (1) If the running TR K^t is smaller than the target one R^t, the TALR U^t should increase to allow more latent weights to pass transition points. (2) If the running TR K^t is larger than the target one R^t, the TALR U^t should decrease to reduce the number of transitions. (3) If the current running TR K^t matches the target one R^t, the TALR U^t should remain unchanged. We currently adjust the TALR in an additive manner in Eq. (11) of the main paper, similar to the weight update in gradient-based optimizers. Here, we consider two alternative variants. First, we update the TALR in a multiplicative manner as follows:

$$U^t = U^{t-1} \frac{R^t}{K^t}. \tag{S1}$$

While the multiplication allows fast adaptation for the TALR, it might be sensitive to outliers from the running TR K^t or drastic changes in the target TR R^t. To stabilize the update, we adjust the TALR in a similar way to Eq (S1), but with momentum, as follows:

$$U^t = m' U^{t-1} + (1 - m') \, U^{t-1} \frac{R^t}{K^t}, \tag{S2}$$

where m' is a momentum hyperparameter, which is set to the same value as m in Eq. (10) of the main paper for simplicity. To compare the update schemes, we analyze in Fig. S5 the QAT process of ResNet-20 [9] on CIFAR-100 [16]. We can see from Figs. S5a and S5b that the running TRs K^t closely follow a target one R^t with the update schemes of Eqs. (11) and (S2). The scheme in Eq. (S1) fails to follow the target TR, resulting in the worst performance (Fig. S5d). The multiplicative update in Eq. (S1) enables fast adaptation, but it is unstable and particularly problematic when the target TR R^t changes abruptly by a step scheduler. The momentum update in Eq. (S2), on the other hand, offers a good compromise between stability and adaptation speed, achieving the quantization performance comparable to the update scheme of Eq. (11) of the main paper. Although we adopt the scheme of Eq. (11) in our TR scheduling technique for conciseness, Fig. S5 suggests that how to adjust the TALR is not unique, and it can be updated in various ways.

S3.3. Average effective step size

The TR scheduling technique adjusts a TR of quantized weights (*i.e.*, Eq. (5) of the main paper) explicitly, which in turn controls the degree of parameter changes for the quantized weights. As a variant of our method, we could instead formulate k^t in Eq. (5) the main paper with the average effective step size of the quantized weights as follows:

$$k^t = \frac{\sum_{i=1}^{N} \left| w_q^t(i) - w_q^{t-1}(i) \right|}{N}. \tag{S3}$$

In this way, we could control the average effective step size of the quantized weights directly during training, similar to the TR in our method. Note that the effects of our method and its variant are nearly the same, if the quantized weights transit between adjacent discrete levels as discussed in Eq. (9) of the main paper. This is always true for binary weights, or usually happens especially when the parameter updates for latent weights are small. When adopting the variant in Eq. (S3), however, it could be difficult to set a hyperparameter (*i.e.*, an initial target value for the average effective step size) due to different scales in quantized weights. For example, several quantization methods [7, 25] provide quantized weights whose min-max ranges are different depending on layers in a model. In this case, we should use different search ranges for individual layers, to set initial target values, since the scales of average effective step sizes could vary in different layers. It is however computationally demanding to find an optimal initial target value for each layer separately. Accordingly, we focus on scheduling the TR rather than the average effective step size, which is more easy to use and generalizable to various quantization schemes.

S4. Implementation Details

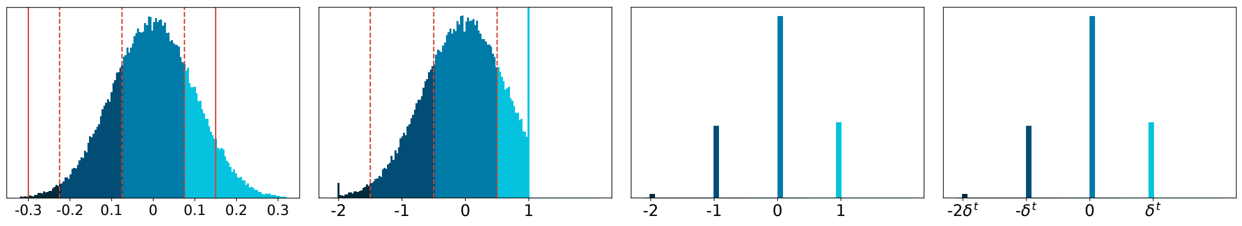
S4.1. Experimental settings

Following the experimental protocol in [12, 17, 20, 28, 30], we initialize network weights from pretrained full-precision models for MobileNetV2, ResNets, DeiTs and RetinaNet. Similarly, the weights in ReActNet-18 are initialized using the pretrained activation-only binarized model. We do not quantize the first and last layers. We use either plain optimizers using a LR (SGD, Adam, and AdamW), or their variants using our TR scheduling technique (SGDT, AdamT, and AdamWT), whose gradient terms are the same as SGD, Adam, and AdamW, respectively. Note that we use a TR scheduler to the latent weights only (*i.e.*, full-precision weights coupled with quantizers). Other parameters that are not quantized (*e.g.*, the weights in the first and last layers) are updated with the plain optimizers using a LR. We adopt the cosine scheduler [21] for both the LR and the target TR, decaying them to zero gradually. For ReActNet-18 on ImageNet, we use the linear scheduler, following the training scheme in [20]. We exploit STE [1] to propagate gradients in discretization functions. To set an initial value of the learnable scale parameter s in Eq. (13) of the main paper, we follow the initialization technique in [17]. For an initial TALR, we use the same value as an initial LR for full-precision parameters. We also set η in Eq. (11) of the main paper to the same value as an initial TALR (*i.e.*, $\eta = U^0$), adjusting the TALR according to its initial value, *e.g.*, updating the TALR more finely when the initial value is small. Note that the number of transition points increases as bit-widths of weights get larger, suggesting that a larger target TR is desirable for larger bit-widths. Accordingly, motivated by [7] that uses different gradient scales according to bit-widths, we set the initial target TR to $\lambda\sqrt{b_w}$, where λ is a TR factor, a hyperparameter to set, and b_w is the bit-width of weights.

We summarize in Table S8 hyperparameter settings for our experiments. If available, we follow the previous works [7, 17, 19, 20, 23] to set hyperparameters (*e.g.*, number of training epochs/iterations, batch size, LR, weight decay, and decay scheduler). Different from the original work of RetinaNet [19], we do not freeze the ResNet [9] backbones to optimize quantized weights via QAT, and adopt the cosine decay scheduler [21]. For the models exploiting our TR scheduling technique, we fix the TR momentum constant m to 0.99 for simplicity. We choose the TR factor λ in a set of $\{$5e-3, 1e-3, 5e-4, 1e-4$\}$. When we train a scale parameter s in Eq. (13) of the main paper, we set the LR to the value ten times smaller than the one for weight parameters (*i.e.*, the LR in Table S8), similar to the works of [12, 17], for simplicity, instead of using the heuristic gradient scaling in LSQ [7]. Note that when we apply our TR scheduling technique, we do not train the scale parameters s for weight quantizers and fix the initial values, since training them makes it difficult to control a TR of quantized weights. We will discuss this in more detail in the following section.

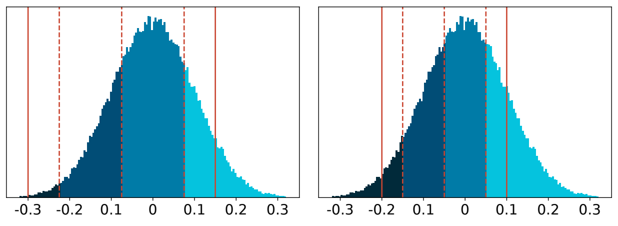
Table S8. Hyperparameter settings in our experiments.

Dataset	Architecture	Optimizer	Epoch	Batch size	LR	Weight decay	Decay scheduler	TR factor λ	TR momentum m
CIFAR-10/100	ResNet-20 (ReActNet-18)	SGD	400 (200)	256	1e-1	1e-4	cosine	-	-
		SGDT						5e-3	0.99
		Adam			1e-3			-	-
		AdamT						5e-3	0.99
ImageNet	ResNet-18	SGD	120	256	1e-2	1e-4	cosine	-	-
		SGDT						1e-3	0.99
		Adam			1e-4			-	-
		AdamT						1e-3	0.99
	ReActNet-18	SGD	120	256	1e-1	0	linear	-	-
		SGDT						5e-4	0.99
		Adam			1e-3			-	-
		AdamT						5e-4	0.99
	MobileNetV2	SGD	120	256	1e-2	2.5e-5	cosine	-	-
		SGDT						1e-3	0.99
		Adam			1e-4			-	-
		AdamT						1e-3	0.99
	DeiT-T/S	AdamW	300	256	3e-4	5e-2	cosine	-	-
		AdamWT						1e-3	0.99
MS COCO	RetinaNet w/ ResNet backbone	SGD	90K (iter.)	16	1e-2	1e-4	cosine	-	-
		SGDT						1e-3	0.99



(a) Latent weights $\mathbf{w}$. (b) Normalized latent weights $\mathbf{w}_n$. (c) Discrete weights $\mathbf{w}_d$. (d) Quantized weights $\mathbf{w}_q$.

Figure S6. Visualization of the quantization process for a 2-bit weight quantizer. Solid vertical lines in (a) indicate clipping points w.r.t latent weights, which adjust a quantization interval. Dashed vertical lines in (a) and (b) represent transition points of the quantizer and a discretization function (*i.e.*, a round function) w.r.t the latent weights and normalized ones, respectively. The weights with the same color belong to the same discrete level of the quantizer. We denote by δ^t in (d) the distance between adjacent discrete levels of the quantizer at the t-th iteration step. (Best viewed in color.)



(a) Quantization interval w.r.t latent weights ($s = 0.3$). (b) Quantization interval w.r.t latent weights ($s = 0.2$).

Figure S7. Comparison between quantization intervals for the same latent weights using different scale parameters. Solid and dashed vertical lines represent clipping and transition points of quantizers, respectively. The weights with the same color belong to the same discrete level of the quantizer. (Best viewed in color.)

S5. Detailed Description of a Quantizer

In this section, we explain the quantization process for a single quantizer. For simplicity, we consider a 2-bit weight quantizer based on Eq. (13) of the main paper, and use latent weights randomly drawn from the zero-mean Gaussian distribution with a standard deviation of 0.1. In the following, we describe a detailed procedure of quantization according to the steps described in Eqs. (1)-(3) of the main paper. We visualize in Fig. S6 an overall quantization process for the quantizer. First, the latent weights $\mathbf{w}$ (Fig. S6a) are normalized and clipped by a normalization function, producing normalized latent weights $\mathbf{w}_n$ (Fig. S6b):

$$\mathbf{w}_n = \text{clip}\left(\frac{\gamma \mathbf{w}}{s}, \alpha, \beta\right), \tag{S4}$$

where we set $\alpha = -2$, $\beta = 1$, and $\gamma = 2$ for 2-bit weight quantization. s is a learnable scale parameter adjusting a quantization interval. In Fig. S6, we set $s = 0.3$ for the purpose of visualization. Second, the normalized latent weights $\mathbf{w}_n$ are converted to discrete ones $\mathbf{w}_d$ (Fig. S6c) using a round function:

$$\mathbf{w}_d = \lceil \mathbf{w}_n \rfloor . \tag{S5}$$

Lastly, the discrete weights $\mathbf{w}_d$ are fed into a de-normalization function that applies post-scaling and outputs quantized weights $\mathbf{w}_q$ (Fig. S6d):

$$\mathbf{w}_q = \frac{1}{\gamma} \mathbf{w}_d. \tag{S6}$$

Note that the post-scaling in Eq. (S6) is fixed, dividing the discrete weights with a constant value γ of 2. In this case, the distance between the adjacent discrete levels of the quantizer (*i.e.*, δ^t in Fig. S6d) is fixed for all training iterations, *i.e.*, $\delta^t = 0.5$ for all t.

S5.1. Counting transitions

In Eq. (5) in the main paper, we count the number of transitions by observing whether discrete weights, *i.e.*, integer numbers resulting from a round or a signum function (*e.g.*, $\mathbf{w}_d$ in Eq. (S5)), are changed or not after a single parameter update. As an example, suppose a case that a quantized weight at the t-th iteration step w_q^t belongs to the first level of the quantizer, *e.g.*, $w_q^t = -2\delta^t$ in Fig. S6d, where corresponding discrete weight w_d^t in Fig. S6c is -2. If the quantized weight transits from the first to the second level of the quantizer after a parameter update (*i.e.*, $w_q^{t+1} = -\delta^{t+1}$), we can detect the transition using the discrete weight, since it is changed from -2 to -1. Similarly, if the quantized weight remains in the same level after a parameter update (*i.e.*, $w_q^{t+1} = -2\delta^{t+1}$), we can say that the transition does not occur, because the discrete weight retains the same value. Note that we could use quantized weights $\mathbf{w}_q$ instead of discrete weights $\mathbf{w}_d$ for counting the number of transitions in Eq. (5) in the main paper, only when δ^t is fixed for all training iterations (*e.g.*, as in our quantizer in Eq. (13) of the main paper). Otherwise this could be problematic. For example, even if a quantized weight does not transit the discrete level after a parameter update, *e.g.*, $w_q^t = -2\delta^t$ and $w_q^{t+1} = -2\delta^{t+1}$, the quantized weight can be changed if δ^t and δ^{t+1} are not the same. This indicates that we cannot detect a transition with the condition of $\mathbb{I}\left[w_q^{t+1} \neq w_q^t\right]$, since the statement ($w_q^{t+1} \neq w_q^t$) could be always true, regardless of whether a transition occurs or not, if $\delta^{t+1} \neq \delta^t$ for all training iterations. Consequently, we count the number of transitions using discrete weights in Eq. (5) in the main paper, which is valid for general quantizers.

S5.2. Quantization interval

Following the recent state-of-the-art quantization methods [7, 12, 17], we introduce in Eq. (13) of the main paper (or in Eq. (S4)) a learnable scale parameter s. Given that α, β and γ in Eq. (13) of the main paper are bit-specific constants, the scale parameter s is the only factor that controls a quantization interval (*i.e.*, a clipping range) w.r.t quantizer inputs. We can see from Fig. S6a that the scale parameter ($s = 0.3$) is responsible for the positions of clipping points w.r.t latent weights (solid vertical lines in Fig. S6a). It also decides transition points accordingly (dashed vertical lines in Figs. S6a), since the points are set by splitting the clipping range uniformly. This suggests that different scale parameters would give different sets of transition points. To verify this, we compare in Fig. S7 the quantization intervals using different scale parameters s for the same latent weights. We can see that the quantization interval shrinks if a smaller scale parameter ($s = 0.2$) is used, and the transition points are altered consequently. This again suggests that transitions could occur if the scale parameter s is changed during training, even when the latent weights are not updated. For example, a latent weight of -0.2 in Fig. S7a belongs to the second level of the quantizer, whereas that in Fig. S7b belongs to the first level. Within our TR scheduling technique, we attempt to control the number of transitions by updating latent weights with a TALR, but the transitions could also be affected by the scale parameter. For this reason, we do not train the scale parameters in weight quantizers, when the TR scheduling technique is adopted, fixing the transition points of the quantizers and controlling the transitions solely with the latent weights.

References

[1] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv*, 2013. 7

[2] 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. 1, 2

[3] 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] 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. 1, 2

[5] Timothy Dozat. Incorporating Nesterov momentum into Adam. In *Int. Conf. Learn. Represent. Workshop*, 2016. 5

[6] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12(7), 2011. 5

[7] 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, 7, 9

[8] 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

[9] 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. 2, 3, 4, 5, 6, 7

[10] 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. 6

[11] 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. 1

[12] 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, 7, 9

[13] Dohyung Kim, Junghyup Lee, and Bumsub Ham. Distance-aware quantization. In *Int. Conf. Comput. Vis.*, pages 5271–5280, 2021. 4

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Int. Conf. Learn. Represent.*, 2014. 2, 4, 5

[15] 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. 6

[16] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, 2009. 4, 5, 6

[17] 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, 2, 4, 7, 9

[18] 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, 3

[19] 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, 3, 7

[20] 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. 2, 4, 5, 7

[21] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *Int. Conf. Learn. Represent.*, 2017. 4, 7

[22] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Int. Conf. Learn. Represent.*, 2019. 5

[23] Markus Nagel, Marios Fournarakis, Yelysei Bondarenko, and Tijmen Blankevoort. Overcoming oscillations in quantization-aware training. In *Int. Conf. Mach. Learn.*, 2022. 1, 2, 6, 7

[24] 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. 1, 6

[25] 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. 7

[26] 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

[27] 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. 5

[28] Kohei Yamamoto. Learnable companding quantization for accurate low-bit neural networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5029–5038, 2021. 4, 7

[29] 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

[30] 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. 7