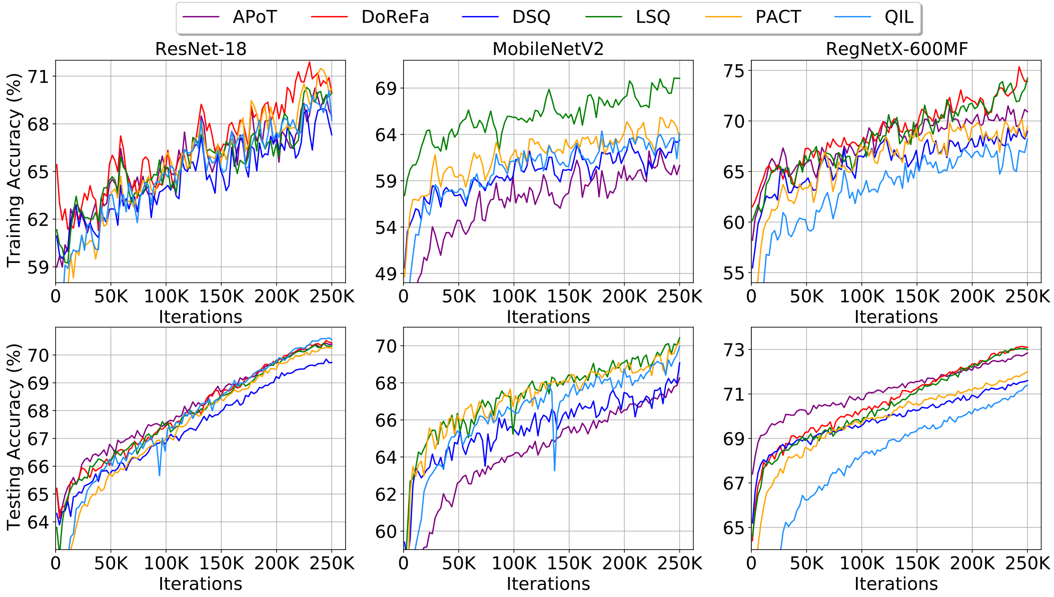


Figure 7: Visualization of training and testing accuracy for 6 different algorithms with academic setting on ResNet-18, MobileNetV2 and RegNetX-600MF. Note that DoReFa failed to converge in MobileNetV2 training.

A Choice of Hyper-parameters.

We utilize a single, fixed set of training hyper-parameters for all MQBench experiments. As we demonstrated in the experimental evaluation, the hyper-parameters and other training techniques can have a profound impact on the final performance. Thus we have to carefully select the optimal hyper-parameters.

Optimizer: In our preliminary experiments, we compare SGD and Adam optimizer. The results are somewhat counter-intuitive. The ranking of Adam and SGD is not consistent in all the experiments. Adam occasionally outperforms SGD in LSQ experiments. While for PACT, DoReFa quantization, SGD tends to have higher performances. Therefore we choose to use SGD for final optimizer. We suggest to study the optimizer's impact on quantization algorithm in future work.

Learning rate and scheduler: Following LSQ [15], we use the cosine learning rate decay [16] to perform the quantization-aware training. This scheduler is also adopted in other areas for training the model [19] and generally has good performance. For the initial learning rate, we run a rough grid search in $\{0.04, 0.01, 0.004, 0.001\}$. For ResNet-family (including ResNet-18, -50, RegNetX), we find 0.004 works best. For MobileNet-family (including MobileNetV2 and EfficientNet-Lite0), we find 0.01 works best.

Weight Decay ($L2$ Regularization): The weight decay choice also has great impact on the final performance of the quantization algorithms. Following existing state-of-the-art practice [15], 4-bit model can reach FP-level accuracy. Thus a same degree of regularization is appropriate. We find this intuition works well in ResNet-family with academic setting. Therefore, we set the weight decay the same as the full precision training for ResNet-family. For MobileNetV2, we find further decreasing the weight decay from $4e-5$ to $1e-5$ can improve the accuracy, therefore we choose to use this setting. However, we should point out that in hardware-aware settings, the 4-bit QAT cannot reach original FP-level performance, thus tuning the weight decay in hardware setting can further boost the performance. We do not explore the weight decay tuning for each setting in an effort to keep the training hyper-parameters unified.

Batch size and training iterations: We train each model for 100 epochs. This is broadly adopted in existing quantization literature. However, for MobileNet-family we find 150 epochs training can further improve the accuracy. We do not employ this choice since 150 epochs training is also time-consuming. As for the batch size choice, we think large batch training can reduce training time but will increase the cost to reproduce the results. Therefore our principle is to limit the maximum number of devices to 16 and to increase the batch size as large as possible. For ResNet-18, we only employ 8 GPUs.

Clarification: The hyper-parameters choice is based on the experience in prior works and our simple preliminary exploration. We search the hyper-parameters in academical setting, although we are aware of that the hardware settings can have a different optimal choice. Our intention is to build a benchmark with unified training settings and to eliminate the bias brought by training hyper-parameters.

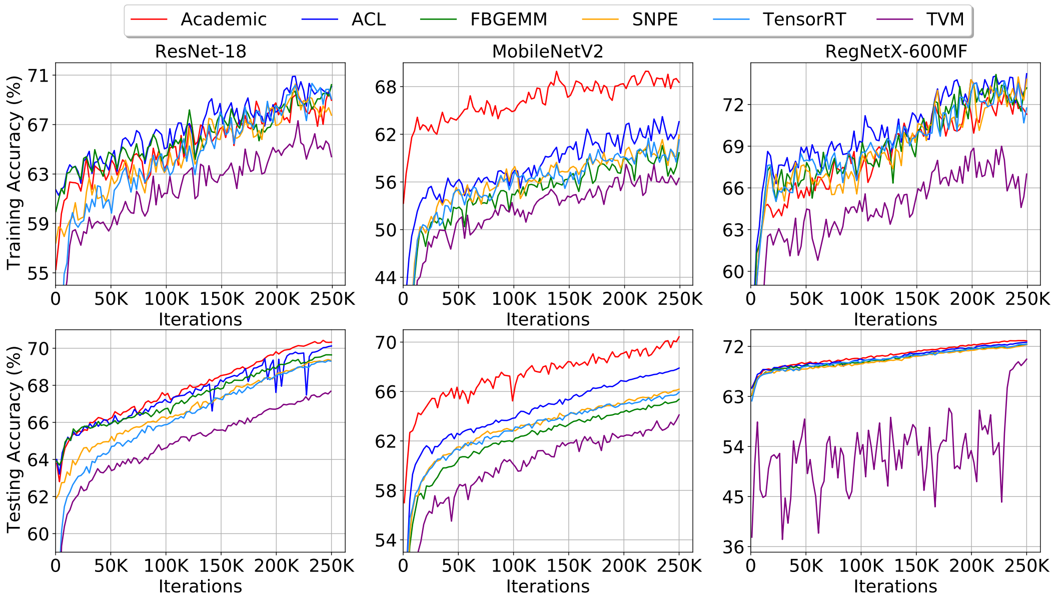


Figure 8: Visualization of training and testing accuracy for 6 different setting using LSQ on ResNet-18, MobileNetV2 and RegNetX-600MF.

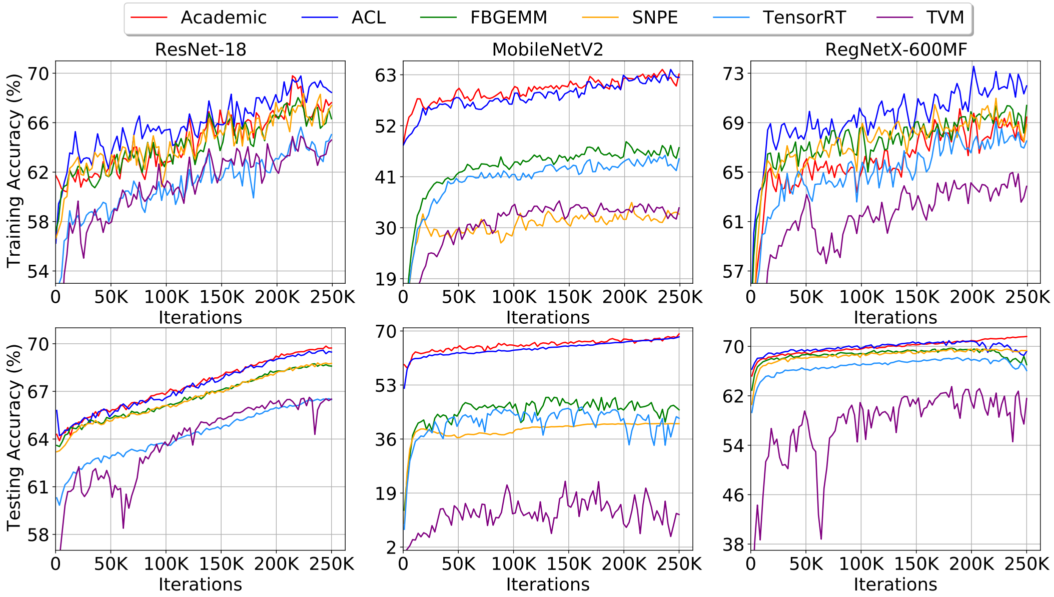


Figure 9: Visualization of training and testing accuracy for 6 different setting using DSQ on ResNet-18, MobileNetV2 and RegNetX-600MF.

B Diagnostic Information

The final test accuracy may be too sparse to evaluate a algorithm or a strategy. In MQBench, we also release the diagnostic information like [48], in order to provide more useful information in studying the quantization. Our diagnostic information includes ①: the training log file and training & testing curve, which reveals the convergence speed and ②: the final checkpoints as well as the quantization parameters, i.e. scale and zero point, which can be used to observe the quantization range.

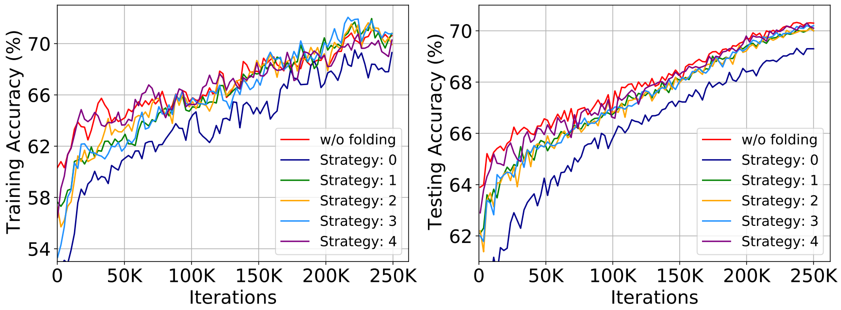


Figure 10: Visualization of training and testing accuracy of ResNet18 for 6 different folding BN strategies.

B.1 Training Curve

In this section we visualize the training curve as well as the test curve in the quantization-aware training. For training accuracy we record the training accuracy per 1k iterations. Then we use exponential moving average with momentum 0.3 to draw the evolution of the training accuracy. For test curve we directly record the validation accuracy in every 1k iterations.

Algorithm with Academic Setting. In Fig. 7, we visualize 6 algorithms (LSQ, PACT, DoReFa, QIL, DSQ, APoT) on three architectures, including ResNet-18, MobileNetV2 and RegNetX-600MF. All these quantization algorithms are trained with academic setting. Generally, QIL and PACT has the relatively low initialization accuracy. In terms of convergence speed, we find LSQ, DoReFa perform well. In MobileNetV2 curve, we observe the increasing instability of the test accuracy. QIL even drops 5% accuracy occasionally.

Hardware. We visualize the LSQ and DSQ algorithms under 6 different setting, as shown in Fig. 8 and Fig. 9. In a nutshell, we find that ACL (blue) curve has the closest distance with the academic (red) curve. TVM has the lowest curve. This is because the scale in TVM is power-of-two, and the quantizer is per-tensor. On RegNetX-600MF, we observe significant fluctuations of LSQ TVM, indicating the challenge of the real world hardware.

Folding BN Strategy. In Fig. 10, we compare different folding BN strategies on ResNet-18 with LSQ algorithm. The strategy 4 has a similar convergence route with regular BN training. However we observe no obvious difference between 1-4 for final convergence. Strategy 0, on the contrary, has the lowest performance.

B.2 Quantization Range

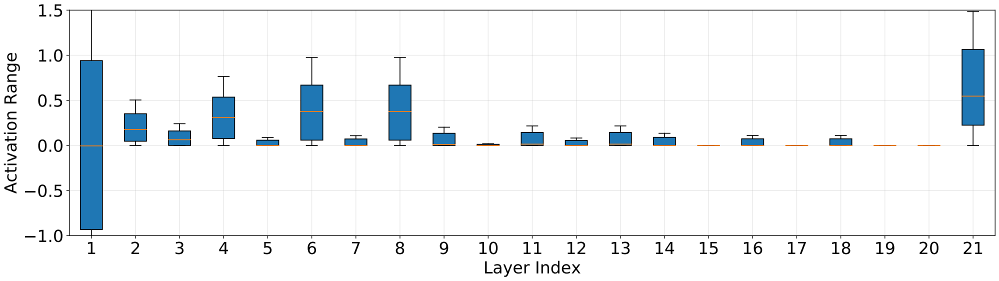
The quantization range, also called *the clipping range* is essential to quantization performance. The quantization range must be large enough to cover the majority of the activation (clipping error), at the same time, it should be small enough to ensure the rounding error won't get too large. In this section we visualize the activation quantization range (weights quantization range is less informative because some algorithms transform the weights to different distribution) under different architectures, algorithms, and hardware settings. This visualization might help to inspire future research in developing the quantization algorithms.

B.2.1 Models

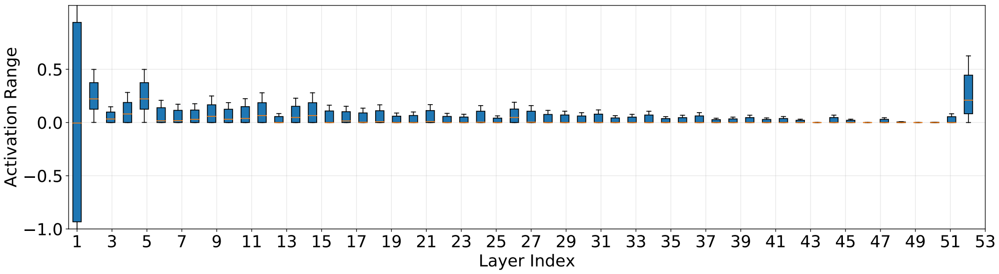
In this subsection we include the visualization of activation distribution in academic setting. We use LSQ, on 5 different architectures: ResNet-family including (ResNet-18, ResNet-50, RegNetX-600MF), MobileNet-family including (MobileNetV2, EfficientNet-Lite0). The results are visualized in Fig. 11. We should point out that the first layer is the input image and the last layer is the input to the final fully-connected layers, which may have larger range and are quantized to 8-bit. By excluding the first and the last layer we find that the most layers in ResNet-family have less than 1 range. Moreover, in certain layers, the activation has an extremely small range, e.g. (0, 0.1). Such distribution may be lossless if quantization is applied to it. Next, we find the activation in MobileNet-family has much more larger variance. For example, the activation in EfficientNet-Lite0 can range from -10 to $+10$. We think two factors contribute to this abnormal distribution. First, the depthwise convolution prevents the communication between channels. As a result, each channel will learn its own range. Second, the shortcut-add and the linear output layer of the block is easy to accumulate the activations across blocks. The formulation is given by

$$F(\boldsymbol{x}) = f(\boldsymbol{x}) + \boldsymbol{x}, \tag{5}$$

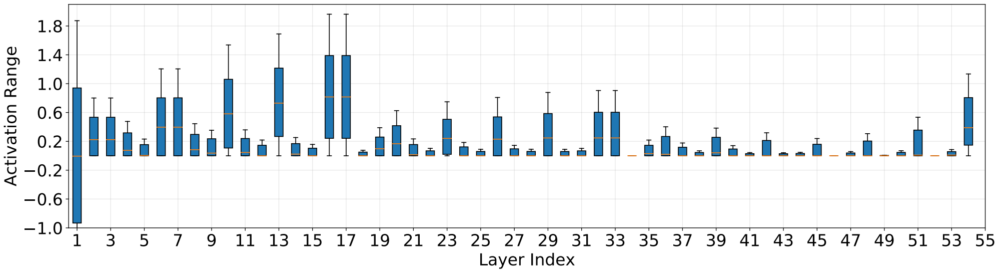
We can see that the block output contains the input, so several blocks' output are accumulated. In ResNet-18, the block output will be activated by ReLU, thus restricting the range of the activation.



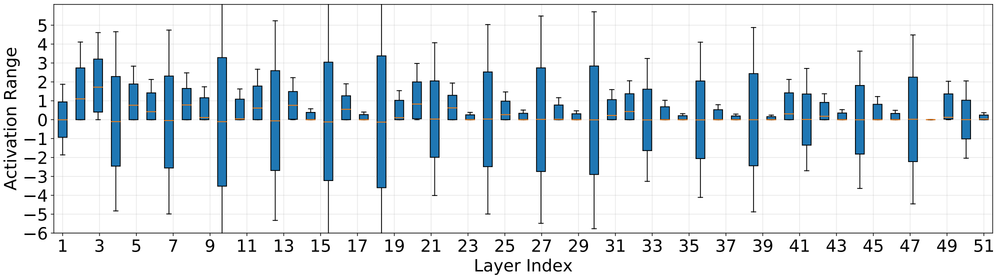
(a) ResNet-18 activation



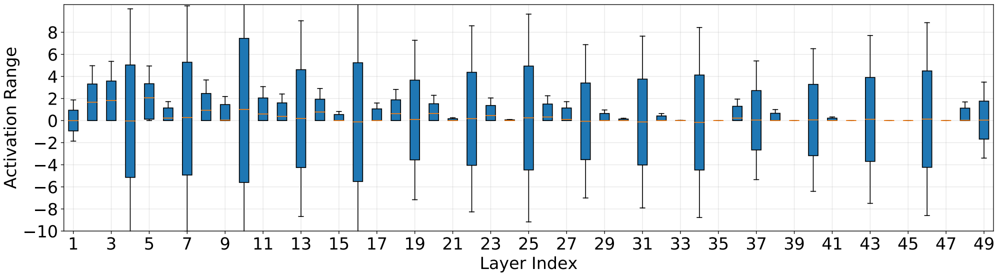
(b) ResNet-50 activation



(c) RegNetX-600MF activation



(d) MobileNetV2 activation



(e) EfficientNet-Lite0 activation

Figure 11: The activation distribution of LSQ in academic setting.

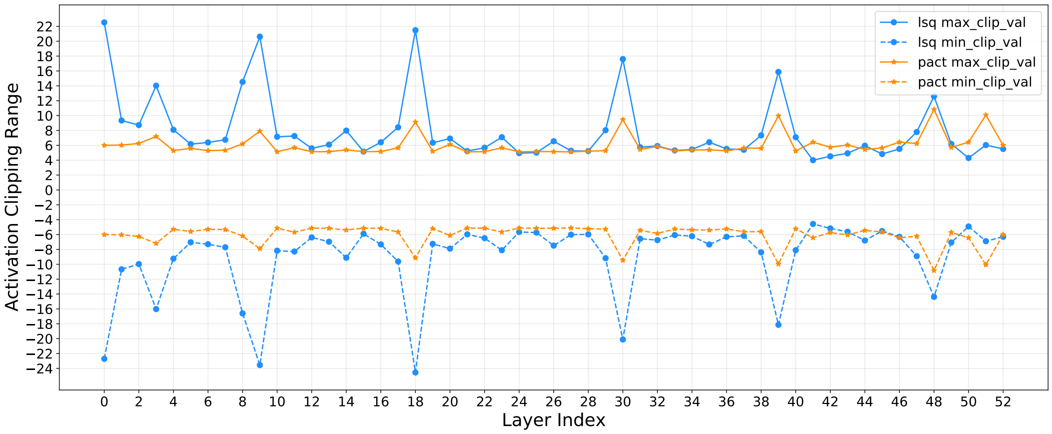


Figure 12: Activation clipping range on MobileNetV2 with TensorRT setting.

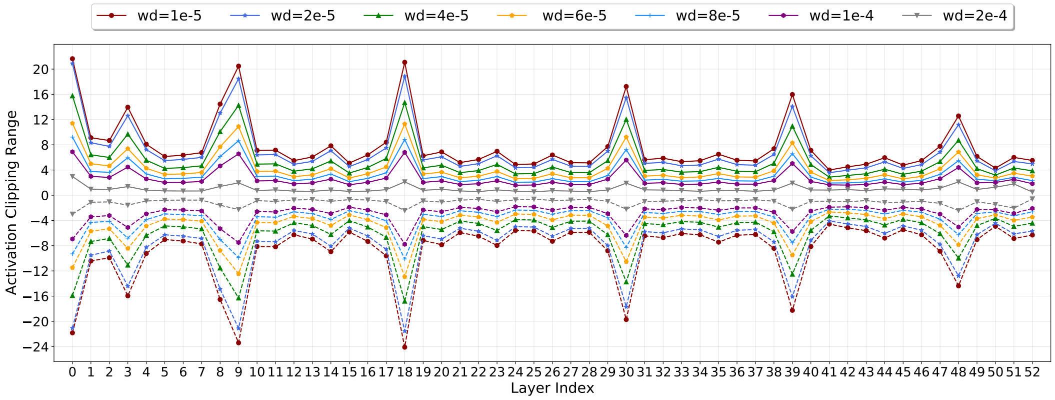


Figure 13: Activation clipping range learned by LSQ, given different $L2$ regularization of the scale parameter.

B.2.2 Algorithm

In this subsection we visualize the activation range on MobileNetV2 with PACT and LSQ. As shown in Fig. 12, the LSQ learns a significant larger clipping range than PACT. For PACT experiments, the maximum clipping range is $[-10, 10]$, however in LSQ the range could be up to $[-20, 20]$. We conjecture this is due to the weight decay. For LSQ optimization, the learnable parameter is the scale, while for PACT optimization the parameter is the clipping threshold. Note that $\alpha = N_{max} \times s$, therefore the regularization effect of α is much more evident than s. This phenomenon also proves that LSQ gradients cannot prevent the scale from growing too large.

B.2.3 Ablation Study

Having observed the large clipping range learned by LSQ, we conduct an ablation study: *imposing different weight decay on the scale parameter.* For other weights and bias parameters we remain the original choice. We run our ablation study on MobileNetV2 with TensorRT setting. The weight decay are chose from $\{1e-5, 2e-5, 4e-5, 6e-5, 8e-5, 1e-4, 2e-4\}$. The results are presented in Table 8 and the clipping range of activation are visualized in Fig. 13. It is easy to observe that the clipping range continues to narrow along with the increasing weight decay. Moreover, we find changing the regularization of scale leads to proportional change, a very similar phenomenon with rule-based clipping range. In terms of final accuracy, weight decay $8e-5$ reaches the highest results and is 2.3% higher than our baseline. We think this result indicates that learning-based range tends to clip less activation, and $L2$ regularization plays an important role to balance the total error.

B.3 Latency

The final objective of quantization is to accelerate the inference of neural networks. To show the practical efficiency of quantization, we profile six different hardware, including Tesla T4 and Tesla P4 (TensorRT), Atlas 300 (ACL), Snapdragon 845 Hexagon DSP (SNPE), Raspberry 3B (TVM) and Intel i7-6700 (FBGEMM). We

Table 8: Accuracy comparison of LSQ on MobileNetV2, given different $L2$ regularization of the scale parameter.

Weight Decay	$1e-5$	$2e-5$	$4e-5$	$6e-5$	$8e-5$	$1e-4$	$2e-4$
Accuracy	66.10	67.04	67.86	68.30	**68.41**	68.03	66.80

Table 9: Latency Benchmark with different architectures and different hardware under 8 bits. All results are testd with 5 runs. (Numbers in the bracket represent the speed up ratio compared with the former row. Green means faster and Red means slower.)

Model			ResNet-18	ResNet-50	MobileNetV2	EfficientNet-Lite0	RegNetX-600MF
FLOPS			1813M	4087M	299M	385M	600M
Batch 1 Latency (ms)	Tesla T4	FP32	1.27	3.33	0.94	1.39	1.69
		FP16	0.54(2.35)	1.11(3.0)	0.42(2.24)	0.49(2.84)	1.24(1.36)
		INT8	0.43(1.26)	0.89(1.25)	0.43(0.98)	0.49(1.0)	1.39(0.89)
	Tesla P4	FP32	1.98	4.67	1.82	2.19	4.33
		INT8	0.99(2.0)	2.08(2.25)	1.03(1.77)	1.53(1.43)	3.7(1.17)
	Atlas300	FP16	1.25	2.52	103.2	97.31	1.54
		INT8	1.0(1.25)	1.96(1.29)	102.2(1.01)	95.92(1.01)	1.39(1.11)
	Snapdragon 845	FP32	15.13	36.04	9.27	11.09	20.84
		INT8	12.09(1.25)	25.64(1.41)	9.02(1.03)	10.51(1.06)	16.69(1.25)
	Raspberry 3B	FP32	689.50	-	200.30	-	252.09
		INT8	567.33(1.22)	-	168.73(1.19)	-	230.77(1.09)
	Intel i7-6700	FP32	30.75	86.63	184.85	238.33	22.39
		INT8	16.42(1.87)	20.22(4.28)	9.16(20.18)	10.79(22.09)	9.06(2.47)
Batch 8 Latency (ms)	Tesla T4	FP32	6.21	15.85	3.29	4.59	4.34
		FP16	1.68(3.7)	4.17(3.8)	1.35(2.44)	2.26(2.03)	2.59(1.68)
		INT8	0.79(2.13)	2.28(1.83)	0.88(1.53)	1.03(2.19)	2.43(1.07)
	Tesla P4	FP32	6.69	18.42	5.54	7.21	6.52
		INT8	3.02(2.22)	6.25(2.95)	2.24(2.47)	4.69(1.54)	4.59(1.42)
	Atlas300	FP16	5.43	13.59	745.34	750.73	4.47
		INT8	4.37(1.24)	10.82(1.26)	850.91(0.88)	667.41(1.12)	5.15(0.87)
	Snapdragon 845	FP32	143.55	326.19	46.56	54.89	121.80
		INT8	77.18(1.86)	168.28(1.94)	37.97(1.23)	44.6(1.23)	76.67(1.59)
	Intel i7-6700	FP32	137.94	350.68	409.48	522.09	75.79
		INT8	58.33(2.36)	124.14(2.82)	33.0(12.41)	44.25(11.8)	38.35(1.98)

choose five prevalent network architectures: ResNet-18, ResNet50, MobileNet-V2, Efficient-Lite0, RegNetX-600MF. Batch size is set as 1 and 8. Among the diverse hardware, Raspberry 3B is an edge device with limited computational resource and requires a long compilation process with TVM for 8-bit. So we only test ResNet-18, MobileNet-V2 and RegNetX-600MF with batch size of 1.

The overall results are listed in Table 9. The number in bracket displays the speed up compared with the former row (Green represents faster and Red means slower). It can be seen that most cases can enjoy a satisfactory acceleration. However, there still exists cases with little gains and some are even slower with quantization. This indicates that some hardware drops behind for novel network architectures. For GPU such as Tesla T4 and P4, there is a consistent improvement with INT8 compared with FP32, especially for the ResNet-family. However, because Tesla T4 introduces Tensor Core supporting FP16, INT8 shows little advantage compared with FP16. Atlas of HUAWEI is the representative of ASIC. Its INT8 optimization seems very preliminary and only have a 1.03%∼1.29% speed up. For the searched advanced model EfficientNet-Lite0 and RegNetX-600MF, it is even slower using 8-bit for batch 8. For the Mobile DSP on Snapdragon 845, we find that it can ensure a more efficient inference with 8-bit no matter what network we choose. For batch 8, the speed up ratio is up to around 2. Beside, we also evaluate CPU including X86 and ARM. For X86 CPU, we directly utilize the official implementation from Facebook and the FBGEMM's INT8 inference can save up to 20 times of latency compared with the default FP32 counterpart of PyTorch. For ARM CPU, we only compile the integer convolutional kernel with 200 iterations and they can already achieve a 9%∼22% improvement.

With the comprehensive evaluation of latency, researchers can acquire the knowledge of practical acceleration brought by quantization instead of remain the level of theory. Meanwhile, hardware vendors can identify the under-optimized network architecture and put in more efforts. We believe both communities will benefit from it.

C Post-Training Quantization.

C.1 Definition of different calibration algorithm

Calibration is the offline process to calculate scale and zero points of activations and weights by simple forward of pretrained floating-point models on a sampled dataset. Usually, different calibration algorithm employs different statistics like maximum or mean for the calculation. No training is involved in this process which is simple and practical. With t bit-width, the floating-point $\boldsymbol{x}$ is quantized into $\bar{\boldsymbol{x}}$, the definition of calibration algorithm to calculate scale s is as follows:

MinMax calibration. The minimum value and maximum value of the original floating-point $\boldsymbol{x}$ are recorded to calculate scale and zero points. For symmetric quantizer, the equation is as follows:

$$s = \frac{max(|\boldsymbol{x}|)}{(2^t - 1)/2} \tag{6}$$

Quantile calibration. The quantile α is used to determine the maximum value and minimum value. Unless specified noted, the quantile is set to 0.9999 in default. Quantile is a hyper-parameter which means $(1 - \alpha)\%$ largest values are clipped. In asymmetric quantizer, the equation is as follows:

$$s = \frac{quantile(\boldsymbol{x}, \alpha) - quantile(\boldsymbol{x}, 1 - \alpha)}{2^t - 1} \tag{7}$$

MSE calibration. The maximum value is searched to minimize mean squared quantization to find a better clip range for the calculation of scale and zero point. For symmetric or asymmetric quantizer, the equation is as follows:

$$\min_{s} \|\boldsymbol{x} - \bar{\boldsymbol{x}}\|^2 \tag{8}$$

KLDivergence calibration. The maximum value is also searched to minimize the KL divergence of two distribution between the original FP $\boldsymbol{x}$ and the quantized $\bar{\boldsymbol{x}}$. To get the distribution of $\boldsymbol{x}$ and $\bar{\boldsymbol{x}}$, we use the histogram of data and split it into 2048 bins denoted as $hist(*)$. For symmetric or asymmetric quantizer, the equation is as follows:

$$\min_{s} D_{kl}(hist(\boldsymbol{x}), hist(\bar{\boldsymbol{x}})) \tag{9}$$

Norm calibration. The p-norm of absolute value $|x|$ scaled by the maximum quantization number is N_{max} is used to compute scale. Norm calibration is introduced in LSQ [15] with $p = 2$. Unless specified noted, l2 norm is used in default. For symmetric quantizer, the equation is as follows:

$$s = \frac{\||x|\|^p}{\sqrt[2]{N_{max}}} \tag{10}$$

MeanStd calibration. The mean and standard deviation denoted as μ and σ of the original floating point $\boldsymbol{x}$ are recorded to calculate quantization scale. α is a hyper-parameter, LSQ+ [42] uses 3 and DSQ [28] uses 2.6. Unless specified noted, 3 is used in default. For symmetric quantizer, the equation is as follows:

$$s = \frac{max(|\mu - \alpha \times \sigma|, |\mu + \alpha \times \sigma|)}{(2^t - 1)/2} \tag{11}$$

C.2 PTQ experiments

We conduct extensive experiments on different calibration algorithms. For brevity, only MinMax, MSE, KLD, and Quantile are reported in Table 10. 5 models are evaluated to compare the quantization sensitivity among different architectures. Among the results presented below, we find no calibration algorithm always works the best. However, as the calibration requires very little time cost, evaluation with several different calibration algorithms helps us to find some useful insights for algorithm and hardware designers.

Efficient models tend to be more sensitive to quantization. While ResNet-18 and ResNet-50 are almost lossless with simple calibration in their best setting. MobileNetV2 and EfficientNet-Lite0 have 1.4% and 1.6% accuracy degradation even with their best setting. However, RegNetX-600MF is also lossless. We assume that group convolution in RegNetX-600M is more robust to quantization than depthwise convolution in MobileNetV2 and EfficientNet-Lite0.

Different networks may favor different calibration algorithms. In inference library SNPE, MinMax for both activation and weights is the best among four different settings for ResNet-18, ResNet-50, and RegNetX-600MF.

Table 10: PTQ Post-training quantization benchmark on the ImageNet dataset with different calibration algorithm. The accuracy is reported with the mean and variance of accuracy after 3 runs. Two inference library named SNPE and FBGEMM are reported. SNPE has per-tensor quantizer for weights while FBGEMM has per-channel quantizer.

Model	Bit	A_calibration	W_calibration	SNPE Acc.	FBGEMM Acc.
ResNet-18 FP:71.0	8	MinMax	MinMax	$\mathbf{70.7 \pm 0.05}$	$\mathbf{70.9 \pm 0.01}$
	8	MSE	MinMax	$\mathbf{70.7 \pm 0.04}$	70.8 ± 0.04
	8	KLD	MinMax	69.5 ± 0.01	69.7 ± 0.05
	8	Quantile	MinMax	66.6 ± 0.16	66.8 ± 0.2
ResNet-50 FP:77.0	8	MinMax	MinMax	$\mathbf{76.5 \pm 0.02}$	$\mathbf{76.6 \pm 0.03}$
	8	MSE	MinMax	76.4 ± 0.001	$\mathbf{76.6 \pm 0.03}$
	8	KLD	MinMax	75.7 ± 0.02	75.9 ± 0.04
	8	Quantile	MinMax	75.8 ± 0.06	76.0 ± 0.04
MobileNetV2 FP: 72.6	8	MSE	MSE	$\mathbf{71.1 \pm 0.2}$	$\mathbf{71.2 \pm 0.04}$
	8	MinMax	MinMax	70.4 ± 0.04	70.9 ± 0.1
	8	MSE	MinMax	70.9 ± 0.09	$\mathbf{71.2 \pm 0.06}$
	8	KLD	MinMax	69.3 ± 0.04	70.0 ± 0.15
	8	Quantile	MinMax	70.0 ± 0.03	70.6 ± 0.1
EfficientNet-Lite0 FP: 75.3	8	MSE	MSE	$\mathbf{72.6 \pm 0.03}$	$\mathbf{73.7 \pm 0.06}$
	8	MinMax	MinMax	71.0 ± 0.05	73.4 ± 0.01
	8	MSE	MinMax	71.5 ± 0.09	$\mathbf{73.7 \pm 0.05}$
	8	KLD	MinMax	48.1 ± 0.82	72.2 ± 0.02
	8	Quantile	MinMax	67.0 ± 0.9	71.4 ± 0.2
RegNetX-600MF FP: 73.7	8	MSE	MSE	73.3 ± 0.02	73.4 ± 0.01
	8	MinMax	MinMax	$\mathbf{73.4 \pm 0.05}$	$\mathbf{73.5 \pm 0.04}$
	8	MSE	MinMax	73.3 ± 0.04	$\mathbf{73.5 \pm 0.03}$
	8	KLD	MinMax	72.1 ± 0.02	72.2 ± 0.02
	8	Quantile	MinMax	71.4 ± 0.2	71.4 ± 0.2

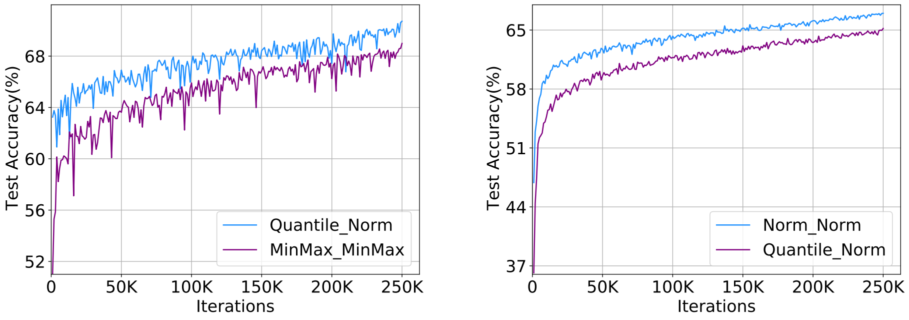
Table 11: Initialization PTQ accuracy is reported with simple calibration algorithm. The accuracy after BatchNorm calibration is denoted as BN Calib Acc. QAT Acc is reported with the same quantization-aware training settings.

Model	Bit	Setting	A_calibration	W_calibration	PTQ Acc.	BN Calib Acc.	QAT Acc.
MobileNetV2 FP: 72.6	4	Academic	Quantile	Norm	1.1	51.3	70.7
	4	Academic	Quantile	MeanStd	0.76	46.1	70.7
	4	Academic	Norm	Norm	0.88	45.0	70.4
	4	Academic	MSE	MeanStd	0.65	39.8	70.3
	4	Academic	MinMax	MinMax	0.43	28.1	69.0
	4	SNPE	Quantile	Norm	0.13	0.14	65.2
	4	SNPE	Norm	Norm	0.18	0.18	67.0

In efficient models, using MSE for both activations and weights is the best MobileNetV2 and EfficientNet-Lite0. MSE also works fines for RegNetX-600MF. It indicates that compared with using the maximum value, searching for a better clipping value is essentially important for an efficient model which may contain some outliers.

Different calibration algorithms bring a large accuracy gap. For ResNet-18, MSE calibration for activation is better than Quantile with over 4.1% with the same MinMax calibration for weights. Similarly for EfficientNet-Lite0, using MSE calibration for weights is better than using MinMax with over 1.1% accuracy gain with the same MSE calibration for activation.

Per-channel quantizer tends to have higher accuracy up bounds. We also compare a per-channel quantizer in FBGEMM with a per-tensor quantizer in SNPE. Although the advantage is not obvious in heavy models, the accuracy of per-channel is higher with 1.1% than per-channel for EfficientNet-Lite0. The results reveal that a per-channel quantizer for weights may come as a better design for increasing the PTQ accuracy.



(a) Academic **(b)** SNPE

Figure 14: Comparison of different initialization. The accuracy curve is plotted and shows that better initialization results in fast convergence and better final accuracy. Quantile_Norm denotes Quantile calibration for activation and Norm calibration for weights.

C.3 The impact of BatchNorm calibration.

In the PTQ experiments under 4 bit, we find that simple calibration fails with accuracy under 1%. The results are listed below in Table 11. Therefore, as suggested by recent papers like [49], BatchNorm statistics changes after quantization especially for low-bit models. Therefore, we propose a simple BatchNorm statistics calibration to see the impact of BatchNorm calibration.

Similar to the calibration of quantization parameters, We sample a calibration dataset to perform the calibration of BatchNorm statistics. The calibration dataset is composed of 4096 training images which are randomly sampled. In the initialization of quantization parameters, the calibration dataset is forward to calculate scale and zero points, which follows the definition of the different calibration algorithms. And then the quantization parameters are kept unchanged, we forward the calibration dataset again to recalculate the BatchNorm statistics as the method proposed in [50]. The running mean and variance of BatchNorm are reset with the average mean and variance in the calibration dataset. In a 4bit academic setting, BatchNorm statistics calibration brings large accuracy improvement. With Quantile calibration for activation and Norm calibration for weights, the accuracy is 51.3% after simple BatchNorm calibration. The Norm calibration for both activation and weights is the same as the LSQ [15] proposes with 45.0% after BN. Therefore, we find a better initialization method for LSQ with over 6.3% accuracy improvement. However, BatchNorm calibration fails under hardware settings. Th BatchNorm Calibration accuracy is still below 1%. It indicates that a more advanced PTQ algorithm is needed for this case.

C.4 The impact of initialization for QAT

While using different calibration algorithms is a common practice for post-training quantization to improve accuracy, it is rarely discussed in quantization aware training. The recent approach named LSQ+ [42] proposes a different calibration named MeanStd for weights and MSE for activations to initialize scale and offset, it shows that different initialization of quantization parameters affects the final QAT accuracy on EfficientNetB0. We extend their experiments with more calibration algorithms with different inference libraries as shown in Table 11.

With the same training set and different calibration algorithms for initialization, the initialization accuracy varies with large gaps. In 4 bit settings, the initialization accuracy using Norm is better than MinMax by 16.9%, and after a full training pipeline with 100 epochs, the QAT accuracy of Norm still surpasses MinMax by 1.4%. However, the final accuracy of Quantile for activation and Norm for weights is only slightly better with 0.3% than Norm for activation and Norm for weights. In hardware settings, the initialization accuracy of different calibration algorithms is below 1%. Therefore, we choose the top1 setting and the original LSQ initialization from the academic setting to perform this comparison. The Quantile is lower than Norm initialization by 1.8%. The training curve is shown in Fig. 14, the right calibration algorithm is chosen to initialize the quantization parameters, it can bring accuracy improvements and bring fast convergence. The calibration algorithm acts as the bridge between PTQ and QAT. Our experiments verify that the combination of the PTQ and QAT process is crucial and affects the final QAT accuracy. How to combine PTQ and QAT still needs further discussion.

D Related Work

Quantization Algorithms. As an appealing strategy for model compression and acceleration, quantization algorithm has driven much attention ever since 90s in last century. In (Holi & Hwang, 1993) [51], an empirical analysis on simple neural networks show 8-bit is sufficient for lossless quantization, Hoehfeld & Fahlman (1992) [52] developed a stochastic rounding scheme to further quantize artificial neural network below 8-bits. In modern deep learning, low-bit quantization has driven lots of interests in academical research. DoReFa-Net [31] first proposes to quantize neural networks into multi-bit. [15, 18, 30] leverage the straight-through estimator (STE) to learn an appropriate clipping range for quantization. DSQ [28] studies the variant of STE and approximate the gradient of step functions. Works like [29, 53, 54] use non-uniform quantization to better adapt the distribution of the weights and activations. Recently, mixed-precision algorithm also made significant progress. For example, [44] uses reinforcement learning to study the bit-width allocation. The eigenvalue of Hessian matrix is employed in [55] to determine the bit for each layer. Differentiable methods like [56,57,58,59] directly learns the bit-width through gradient descent.

Along with quantization-aware training (QAT), a much easier way to perform quantization is called post-training quantization (PTQ). PTQ only requires 10-1000 training images and tiny computational resources to finish the quantization. [60, 61, 62] seek to find the optimal clipping range for 4-bit PTQ. OCS [63] uses channel-splitting to deal with outliers. [5, 45] even do quantization without accessing any real data. [64, 65] adopt intermediate feature-map reconstruction to optimize the rounding policy.

Quantization Frameworks and Hardware. Generally speaking, hardware for quantization can be naturally divided to specialized hardware and general hardware. In 1990, Hammerstrom [66] already designed specialized VLSI hardware for 8-bit and 16-bit training. More recently, specialized hardware like BISMO [67] and BitFusion [68] are designed to handle mixed-precision inference. On general hardware, NVIDIA Volta Tensor Cores [69] can support FP16 & FP32 mixed training and achieve at least 2x speed-up. As for low-bit inference, there are several hardware libraries: such as NVIDIA's TensorRT [22], Qualcomm's SNPE [24], and so on. Hardware providers also build some framework for model quantization (or other compression techniques). For example, the Neural Network Compression Framework (NNCF) [70] developed by Intel supports quantization and pruning. However, their implementation can only be shared on OpenVINO. AIMET, developed by Qualcomm, also just focuses on their own hardware. To our best knowledge, no existing framework can handle multi-platform deployment of quantization neural networks with diverse advanced algorithms.

Benchmarks. In many sub-fields of deep learning or computer vision, a thorough analysis and evaluation is necessary [71]. In neural architecture search (NAS) [72], the search space is notoriously large and the search process requires tremendous computation resources, making researchers hard to reproduce results of prior works. For this reason, NAS-Bench-101 [19] and -201 [48] are proposed to solve the reproducibility issues and make a fair evaluation for NAS research. Recently, Hardware-NAS-Bench [73] was proposed to benchmark the inference latency and energy on several hardware devices. Despite the progress of benchmarks in NAS and other areas, model quantization lacks such standard to foster the reproducibility and deployability.

E Detailed Inference Library Setup

Academic Setup. In academical research, most existing work chooses the per-tensor, symmetric quantization. This quantizer design could be challenging. However, academical setting only quantizes the input and the weight of a convolutional or linear layer. Thus the computational graph is not aligned to any hardware implementations. Note that people tend to use *unsigned* quantization to quantize input activation and *signed* quantization to quantize weights. For unsigned quantization, the target integer range is $[N_{min}, N_{max}] = [0, 2^t - 1]$, while for signed quantization, the range becomes $[N_{min}, N_{max}] = [-2^{t-1}, 2^{t-1} - 1]$. The intuition for adopting unsigned quantization is ReLU activation are non-negative, and symmetric signed quantization will waste one bit for negative parts. In our implementation, we add a switch variable called *adaptive signness*, which can turn the integer range to $[-2^{t-1}, 2^{t-1} - 1]$ based on data statistics. It should be noted that *Adaptive signness* is only designed for academic setting, while symmetric quantization must waste one bit for non-negative activation in real-world hardware.

TensorRT Setup. TensorRT [22] is a high-performance inference library developed by NVIDIA. The quantization scheme in TensorRT is symmetric per-channel for weights, and symmetric per-tensor for activations. The integer range is $[-128, 127]$. TensorRT will quantize every layers in the network including Add and Pooling besides those layers which have weights. However, per-channel quantization scheme can reduce the error for weight quantization to a certain extent. TensorRT model will be deployed on GPUs, and Int8 computation will be achieved by Tensor Cores or DP4A instructions, which are highly efficient. Typically, one GTX1080TI GPU have 45.2 peak Tops in INT8 mode.

SNPE Setup. SNPE is a neural processing engine SDK developed by Qualcomm Snapdragon for model inference on Snapdragon CPU, Adreno GPU and the Hexagon DSP. SNPE supports 8bit fixed-point quantization for Hexagon DSP. Hexagon DSP is an advanced, variable instruction lenghth, Very Long Instruction Word(VLIW) processor architecture with hardware multi-threading. The quantization scheme of SPNE is asymmetric and per-tensor for weights and activations.

TVM Setup. TVM [25] is a deep learning compiler and can compile neural networks to a 8-bit implementation. For now, TVM supports running the whole network in symmetric per-tensor quantization scheme. One different point is, in order to accelerate the quantization affine operation, they represent the scale as power-of-two and thus can utilize the efficient shift operation to enjoy further speed up. The quantized INT8 model compiled by TVM can be deployed on GPUs, CPUs or DSPs.

ACL Setup. ACL is a neural network inference software developed by HUAWEI for the hardware named Atlas. Atlas supports INT8 convolution and linear kernel so ACL quantizes the layer which has weights, such as convolution, fully-connected layer to int8 fixed point, but remains the rest part of network in FP32. The quantization scheme is symmetric per-channel for weight, and asymmetric per-tensor for activation to avoid the waste of one bit. Typically an Atlas 300 inference card have 88 Tops in INT8 mode.

FBGEMM Setup. FBGEMM is a inference library developed by Facebook and can deploy torch model easily. The quantization scheme is asymmetric per-channel and we quantize the whole network into int8 fixed point.

F Quantization Algorithms Implementation

Learned Step Size Quantization. LSQ leverages the Straight-Through Estimator [74] to learn the quantization scale for each layer. For initialization, we use the method proposed in original paper: the scale is determined by $s = {}^{2||\boldsymbol{w}||_1}/\sqrt{N_{max}}$. For symmetric quantization, the zero point is initialized to 0, and kept fixed. For asymmetric quantization, zero point is initialized to N_{min} if the activation is non-negative. Inspired by LSQ+ [42], the zero point can also be updated through backpropagation with the help of STE. Therefore we make it learnable in asymmetric quantization. LSQ uses gradient scale to stabilize the scale learning. The gradient scale is determined by ${}^1/\sqrt{MN_{max}}$ where M is the number of elements in that tensor. We extend this gradient scale to per-channel weight learning, where the M is the number of weights in each filter.

Differentiable Soft Quantization. DSQ uses the hyperbolic tangent function to approximate the conventionally adopted STE. In our implementation, we use $\alpha = 0.4$ (for definition please refer to the original paper [28]) which controls the shape and smoothness of the $\tanh$ function. For weight quantization, we use the min-max range as

$$Clip_{min} = \mu(\boldsymbol{w}) - 2.6\sigma(\boldsymbol{w}), \tag{12}$$

$$Clip_{max} = \mu(\boldsymbol{w}) + 2.6\sigma(\boldsymbol{w}), \tag{13}$$

where $\mu(\cdot)$ and $\sigma(\cdot)$ computes the mean and standard deviation of the tensor. Then, the scale is determined by $s = \frac{\max(-Clip_{min}, Clip_{max})}{N_{max}-N_{min}}$ for symmetric quantization, and $s = \frac{Clip_{max}-Clip_{min}}{N_{max}-N_{min}}$ for asymmetric quantization. The zero point is set to 0 for symmetric and $N_{min} - \lfloor \frac{Clip_{min}}{s} \rceil$ for asymmetric quantization. For activation, we use the BatchMinMax as the clipping range, i.e. the averaged min-max range across the batch dimension. This is further updated with exponential moving average across different batches with momentum 0.9, similar to BN.

Parameterized Clipping Activation. PACT is introduced to quantized activation by learning the clipping threshold through STE. Its activation is clipped by a parameter α first. Then, the clipped activation is quantized and re-quantized. Although PACT and LSQ both learns the scale, they have three differences. First, the clipping range in PACT is handcrafted initialized to 6 while LSQ initialization is based on the tensor $L1$ norm. Second, PACT has no gradient in the range of clipping. While LSQ can compute the gradient. Third, PACT does not scale the gradient of α, while LSQ does. Note that PACT only has non-negative, unsigned quantization in the first. To extend it to our hardware settings, we clip the activation to $(-\alpha, \alpha)$ in symmetric case and (β, α) for asymmetric case, (where β is initialized to -6). For weight quantization of PACT, it is the same with DoReFa-Net.

DoReFa-Network. DoReFa-Net simply clips the activation to $[0, 1]$ and then quantizes it. This is based on the intuition that most activation will fall into this range in old network architectures, e.g. AlexNet [75] and ResNet [17]. In hardware settings, we modify the activation range to $[-1, 1]$ for both symmetric and asymmetric quantization. As for weight quantization, it can be described as:

$$\tilde{\boldsymbol{w}} = \tanh(\boldsymbol{w}) \frac{1}{\max(|\tanh(\boldsymbol{w})|)}, \tag{14}$$

$$\hat{\boldsymbol{w}} = \text{dequantize}(\text{quantize}(\tilde{\text{w}})), \tag{15}$$

where the first step is a non-linear transformation and the second step is the same with Eq. (1). The scale is simply calculated by $\frac{2}{N_{max}-N_{min}}$ for symmetric quantization and $\frac{\max(\tilde{\boldsymbol{w}})-\min(\tilde{\boldsymbol{w}})}{N_{max}-N_{min}}$ for asymmetric quantization.

Additive Powers-of-Two Quantization. APoT quantization uses multiple PoT's (Powers-of-Two) combination to composes a set of non-uniform quantization levels. Since the quantization are non-uniform in most cases (except the case of 2-bit the APoT becomes uniform quantization), we do not benchmark it on real hardware. Additionally, APoT introduces weight normalization (similar to standardization [76] technique) to smooth the learning process of clipping range in weight. However, it is unclear how to incoporate this technique with

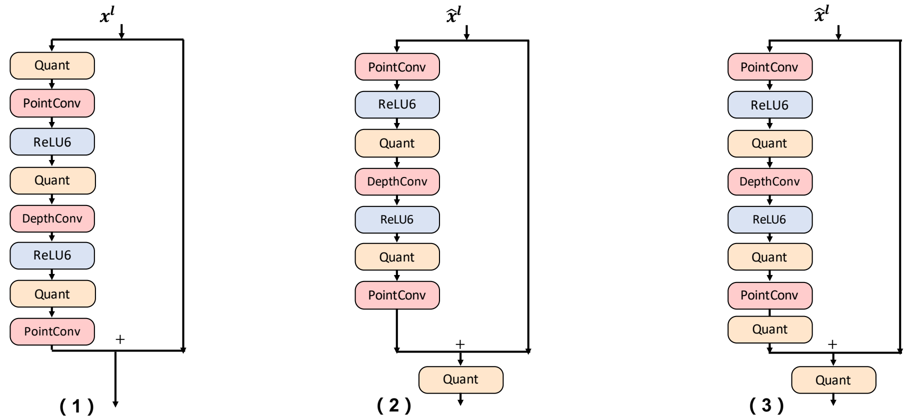


Figure 16: Comparison of different quantization implementations for inverted bottleneck block in MobileNetV2 [13].

BN folding. Therefore, we only reproduce it in our academic setting. The implementation are based on the open-source codes.

Quantization Interval Learning. QIL composes of two unit to quantization: (1) the first one is called transformer, which transform the weights or activation to $[-1, 1]$ ($[0, 1]$ as for non-negative activation). This transformer also has two functionalities: pruning and non-linearity. (2) The second one is called quantizer, given by

$$\tilde{\boldsymbol{w}} = \text{clip}\left((\alpha|\boldsymbol{w}| + \beta)^{\gamma}, 0, 1\right) * \text{sign}(\boldsymbol{w}), \tag{16}$$

$$\hat{\boldsymbol{w}} = \text{dequantize}(\text{quantize}(\tilde{\text{w}})), \tag{17}$$

where $\alpha = \frac{1}{2*D}$ and $\beta = -\frac{C}{2D} + \frac{1}{2}$. This transformation maps the weight from $[C - D, C + D]$ to $[0, 1]$ and $[-C - D, -C + D]$ to $[-1, 0]$. As a result, the weights between $[-C + D, C - D]$ are pruned. The non-linearity of the transformation function is introduced by γ. This parameter can control the linearity and thus control the quantization interval. However, we find this technique is extremely unstable. In our experimental reproduction, learning γ will not converge. In the original paper, the gradient scale of C and D is set to 0.01. We find this gradient scale also leads to frequent crashes. Thus we use the gradient scale introduced in LSQ, i.e. $\frac{1}{\sqrt{MN_{max}}}$.

G Block Graph

In this section we provide visualization of the graph implementation. First, we visualize concatenation operation. In academic setting, the concatenation is not considered to be quantized and is operated at FP32. However, in real-world hardware we must quantize the input of the concatenation. See Fig. 15 aside. It is worthwhile to note that there is only one implementation for concatenation in all hardware deployment environments. Moreover, the quantization of two branches must share one set of scale and zero point. Other-wise the fused branch will be represented with higher precision. This parameter-sharing mechanism in concatenation may cause severe accuracy degradation in practice, although we do not testify any architectures containing concatenation operations.

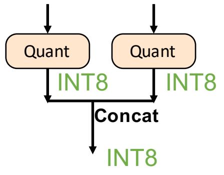


Figure 15: Concatenation graph.

Then, we visualize the *inverted bottleneck block* implementation adopted in MobileNetV2 and EfficientNet-Lite0. As shown in Fig. 16, graph 1 stands for academic setting where only the input of a convolutional layer is quantized. Graph 2, 3 describes the graph implementation in real-world hardware. The difference is similar to Basic Block in Fig. 4, where graph 2 omits the quantization of the main branch in shortcut-add.