AdaQAT: Adaptive Bit-Width Quantization-Aware Training

Cédric Gernigon, Silviu-Ioan Filip, Olivier Sentieys, Clément Coggiola, Mickael Bruno

AdaQAT: Adaptive Bit-Width Quantization-Aware Training

Cédric Gernigon
Univ. Rennes, Inria, CNRS, IRISA
F-35000 Rennes, France
Email: cedric.gernigon@inria.fr

Silviu-Ioan Filip
Univ. Rennes, Inria, CNRS, IRISA
F-35000 Rennes, France
Email: silviu.filip@inria.fr

Olivier Sentieys
Univ. Rennes, Inria, CNRS, IRISA
F-35000 Rennes, France
Email: olivier.sentieys@inria.fr

Clément Coggiola
Spacecraft techniques, on-board data handling
CNES, Toulouse, France
Email: clement.coggiola@cnes.fr

Mickaël Bruno
Spacecraft techniques, on-board data handling
CNES, Toulouse, France
Email: mickael.bruno@cnes.fr

***Abstract*—Large-scale deep neural networks (DNNs) have achieved remarkable success in many application scenarios. However, high computational complexity and energy costs of modern DNNs make their deployment on edge devices challenging. Model quantization is a common approach to deal with deployment constraints, but searching for optimized bit-widths can be challenging. In this work, we present Adaptive Bit-Width Quantization Aware Training (AdaQAT), a learning-based method that automatically optimizes weight and activation signal bit-widths during training for more efficient DNN inference. We use relaxed real-valued bit-widths that are updated using a gradient descent rule, but are otherwise discretized for all quantization operations. The result is a simple and flexible QAT approach for mixed-precision uniform quantization problems. Compared to other methods that are generally designed to be run on a pretrained network, AdaQAT works well in both training from scratch and fine-tuning scenarios. Initial results on the CIFAR-10 and ImageNet datasets using ResNet20 and ResNet18 models, respectively, indicate that our method is competitive with other state-of-the-art mixed-precision quantization approaches.**

***Index Terms*—Neural Network Compression, Quantization Aware Training, Adaptive Bit-Width Optimization**

I. Introduction

Deep Neural Networks (DNN) have achieved remarkable results in recent years in a wide range of domains. While many inference computations are done in the cloud, it is increasingly desirable to deploy trained DNNs to edge devices, such as mobile phones and wearable devices, due to privacy, security, and latency concerns or limitations in communication bandwidth. However, modern DNNs contain at least millions of parameters and require billions of arithmetic operations. Memory and computational costs make deployment on embedded devices difficult, if not infeasible in many cases. To mitigate these issues, various compression techniques have been proposed, such as pruning [1], weight sharing [2], knowledge distillation [3], and quantization [4]. With the emergence of hardware platforms offering better support for low (*e.g.* recent Nvidia GPUs and Google TPUs) and custom precision (*e.g.* FPGA and ASIC solutions) compute, quantization is at the forefront of methods used to increase the efficiency of DNN model inference.

Many models can be uniformly quantized to 8 bits [5] and in some cases to even binary [6] or ternary [7] representations. Various methods [8]–[11] push the compression limit even further, using different bit-widths at the (sub)layer level. However, it is challenging to find efficient mixed-precision configurations that compress a model with minimal impact on accuracy. There are three main families of methods that attempt to optimize bit-width allocation for model compression: *search*, *metric* and *optimization*-based. Search-based methods iteratively explore the bit-width assignment space and are generally costly to use. Metric-based approaches are much faster, but tend to give sub-optimal results. Optimization methods offer good performance at a reasonable cost, but most results of this type tend to suffer from instabilities during the optimization process (cf. [9]), especially if starting far (*e.g.* training from scratch) from an optimized configuration.

In the following, we introduce **Ada**ptive Bit-Width **Q**uantization **A**ware **T**raining (**AdaQAT**), an optimization-based method for mixed-precision uniform quantization of both weights and activations. Its defining characteristic is the use of relaxed fractional bit-widths that are updated using a gradient descent rule, but are otherwise discretized for all operations (in forward and backward passes). Compared to previous approaches, our initial tests show that AdaQAT is able to produce efficient quantized DNNs that are comparable to the state of the art, in both training from scratch and fine-tuning settings.

II. Related Work

Depending on when quantization is performed (after or during training), there are two main families of methods used in practice. The first, Post-Training Quantization (PTQ), is fast, but can lead to non-negligible loss in accuracy for very small formats [12]. Quantization Aware Training (QAT), while slower, generally leads to better results and should be preferred for extreme quantization problems.

A. Quantization-Aware Training

QAT methods stem from pioneering work on binary neural networks [6], [13]. At its core, a QAT method consists of

using a quantized version of the network during training in both forward and backward passes, while performing updates on full-precision copies of the network parameters. These full precision parameters are then quantized to be used in the next iteration. A crucial aspect is how to perform backpropagation through quantized variables (parameters and activations). In the binary case, this was done using a so-called Straight-Through Estimator (STE) [14] and this approach was later [5] extended to cover larger bit-widths, while also applying quantization to gradient signals.

To further improve the accuracy of quantized DNNs, the STE idea can also be used to learn the parameters of uniform quantizers, such as scaling factors and bias terms for weight quantization [15], [16], and in the case of ReLU-based activations, clipping parameters [17].

B. Bit-Width Search Strategies

Finding bit-width allocations that improve inference efficiency has been addressed using various approaches.

Search-based methods like HAQ [8] rely on reinforcement learning with hardware (latency & energy) feedback in the agent, whereas neural architecture search work like DNAS [18] uses gradient-based information. The major downside in using them is that they require significant time and computational resources.

Much faster results can be obtained using metric-based methods. For example, HAWQ [11] uses Hessian spectrum information at each layer to assign precisions. Methods like [19], [20] rely on linear programming models, while [21] encourages quantization that leads to reduced sharpness in the task loss function. A potential downside of these methods might be the fact that they can lead to sub-optimal results compared to other approaches (cf. [9]).

Optimization-based approaches formulate the bit-width assignment as an optimization problem, with the main challenge being how to handle the fact that the loss is non-differentiable w.r.t. the bit-widths. Methods like FracBits [10] and BitPruning [22] use fractional bit-widths and linear interpolation during the forward path, whereas SDQ [9] is based on stochastic quantization, but seems limited to weight quantization. These methods work well in fine-tuning scenarios, but are unstable or do not work when training from scratch.

AdaQAT falls into this third category. It is an optimization-based mixed-precision QAT method that shows good flexibility when compared to other approaches in the same vein.

III. Method

We start by presenting the necessary background on DNN quantization before to describe in detail the proposed method.

A. Quantization background

We adopt the DoReFa [5] scheme for weight quantization and PACT [17] for activation quantization with the improvements suggested in SAT [23]. The same quantization function is applied to both weight and activation:

$$\mathrm{q}(x) = \frac{1}{s} \lfloor xs \rceil \,, \tag{1}$$

where $x \in [0, 1]$, $\lfloor \cdot \rceil$ indicates rounding to the nearest integer, $s = 2^k - 1$ is the scaling factor and k is the quantization bit-width.

The weight tensors are first brought into $[0, 1]$ using the transformation $f(\mathbf{w}) = \dfrac{\tanh(\mathbf{w})}{2\max(|\tanh(\mathbf{w})|)}$ and then rescaled and shifted to $[-1, 1]$. Backpropagation through (1) is done using STE, leading to the following rule for $\mathbf{w}$:

$$\textbf{Forward: } \mathbf{w}_q = 2\,\mathrm{q}\left(f(\mathbf{w}) + \frac{1}{2}\right) - 1$$
$$\textbf{Backward: } \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{\partial \mathcal{L}}{\partial \mathbf{w}_q}\frac{\partial \mathbf{w}_q}{\partial \mathbf{w}}$$

where $\mathbf{w}$ is an unquantized weight tensor, $\mathcal{L}$ is the loss function, and $\mathbf{w}_q$ is the quantized version of $\mathbf{w}$.

PACT [17] proposes to learn the upper bound of a ReLU activation function in order to compute an appropriate scaling factor s. The vanilla ReLU is thus replaced with:

$$\mathrm{PACT}(x) = \begin{cases} 0 & \text{if } x < 0 \\ \alpha & \text{if } x > \alpha \\ x & \text{otherwise} \end{cases}$$

The scaling factor in (1) is now $s = (2^k - 1)/\alpha$. The complete activation quantization procedure is:

$$\textbf{Forward: } \mathbf{y}_q = \mathrm{q}(\mathbf{y})$$
$$\textbf{Backward: } \frac{\partial \mathcal{L}}{\partial \mathbf{y}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}_q}\mathbb{I}_{\mathbf{x} \leqslant \alpha} \text{ and } \frac{\partial \mathbf{y}_q}{\partial \alpha} = \frac{\partial \mathbf{y}}{\partial \alpha}\mathbb{I}_{\mathbf{x} \leqslant \alpha}$$

where $\mathbf{y}$ is an unquantized activation, $\mathcal{L}$ is the loss function, $\mathbf{y}_q$ is the quantized activation, and $\mathbb{I}_{\mathcal{C}(\mathbf{x})}$ is an indicator function that returns 1 if $\mathbf{x}$ satisfies condition $\mathcal{C}$ and 0 otherwise.

B. Objective Function

In order to learn the bit-widths of the uniform quantizers for both weights and activations, we use two real-valued variables $N_{\mathbf{w}}$ and $N_{\mathbf{a}}$, respectively. The actual integer bit-widths of the quantized network are $\lceil N_{\mathbf{w}} \rceil$ and $\lceil N_{\mathbf{a}} \rceil$.

We model the loss function to minimize that takes into account the cost of a particular bit-width configuration as:

$$\mathcal{L}_{\text{Total}} = \mathcal{L}_{\text{Task}}\left(\lceil N_{\mathbf{w}} \rceil, \lceil N_{\mathbf{a}} \rceil\right) + \lambda \mathcal{L}_{\text{Hard}}\left(\lceil N_{\mathbf{w}} \rceil, \lceil N_{\mathbf{a}} \rceil\right) \tag{2}$$

where $\lambda > 0$ is a balancing hyper-parameter between the task and hardware losses.

FracBits [10] has reviewed various methods used to model the hardware cost of arithmetic precision choices for weights and activations. They argue in favor of memory size if only targeting weight quantization, and BitOPs (see [10, eqs. (4) and (5)]) for joint weight and activation quantization. For a convolutional filter f, the BitOPs metric corresponds to

$$\mathrm{BitOPs}(f) = \lceil N_{\mathbf{w}} \rceil \lceil N_{\mathbf{a}} \rceil |f| w_f h_f / s_f^2,$$

where $|f|$ denotes the cardinality of the filter, w_f, h_f, s_f are the spatial width, height, and stride of the filter, respectively.

In our particular case, since we are using one bit-width per weights and one per activations, the overall BitOPs hardware cost will be linear in $\lceil N_{\mathbf{w}} \rceil \lceil N_{\mathbf{a}} \rceil$, namely

$$\mathcal{L}_{\text{Hard}}\left(\lceil N_{\mathbf{w}} \rceil, \lceil N_{\mathbf{a}} \rceil\right) = \lceil N_{\mathbf{w}} \rceil \lceil N_{\mathbf{a}} \rceil .$$

C. Bit-Width Gradients & Parameter Updates

Since the task loss is not directly differentiable with respect to the bit-width parameters, we use finite difference approximations as follows:

$$\frac{\partial \mathcal{L}_{\text{Task}}}{\partial N_{\mathbf{w}}} \approx \mathcal{L}_{\text{Task}}(\lceil N_{\mathbf{w}} \rceil, \lceil N_{\mathbf{a}} \rceil) - \mathcal{L}_{\text{Task}}(\lfloor N_{\mathbf{w}} \rfloor, \lceil N_{\mathbf{a}} \rceil)$$
$$\frac{\partial \mathcal{L}_{\text{Task}}}{\partial N_{\mathbf{a}}} \approx \mathcal{L}_{\text{Task}}(\lceil N_{\mathbf{w}} \rceil, \lceil N_{\mathbf{a}} \rceil) - \mathcal{L}_{\text{Task}}(\lceil N_{\mathbf{w}} \rceil, \lfloor N_{\mathbf{a}} \rfloor)$$

The gradient of the total loss w.r.t. the bit-widths is then approximated as:

$$\frac{\partial \mathcal{L}_{\text{Total}}}{\partial N_{\mathbf{x}}} \approx \frac{\partial \mathcal{L}_{\text{Task}}}{\partial N_{\mathbf{x}}} + \lambda \frac{\partial \mathcal{L}_{\text{Hard}}}{\partial \lceil N_{\mathbf{x}} \rceil} \tag{3}$$

which are then used to update the fractional bit-width parameters. The gradient descent rule that does this takes the form

$$N_{\mathbf{x}}^{+} = N_{\mathbf{x}} - \eta_{\mathbf{x}} \frac{\partial \mathcal{L}_{\text{Total}}}{\partial N_{\mathbf{x}}}, \tag{4}$$

with $\mathbf{x} \in \{\mathbf{w}, \mathbf{a}\}$, $N_{\mathbf{x}}^{+}$ the new bit-width values at the next iteration, and $\eta_{\mathbf{x}} > 0$ corresponding learning rates.

The rest of the network and quantizer parameters are updated using the SGD-like or accelerated algorithms that train the network normally, with their own hyperparameters.

We have noticed that too rapid changes in the values of the learned bit-widths tend to degrade accuracy considerably, slowing down the optimization process. To avoid this, the learning rates need to be reasonably small. Unless otherwise stated, default values of $\eta_{\mathbf{w}} = 0.001$ and $\eta_{\mathbf{a}} = 0.0005$ are considered in our testing. A smaller learning rate $\eta_{\mathbf{a}}$ is picked since it appears that the progressive quantization of activations is more sensitive to changes in $N_{\mathbf{a}}$ than weight quantization is to changes in $N_{\mathbf{w}}$.

When $N_{\mathbf{w}}$ and $N_{\mathbf{a}}$ reach their optimized values, continuing to decrease them will lead to a (steep) increase of the task loss $\mathcal{L}_{\text{Task}}$ and consequently of $\mathcal{L}_{\text{Total}}$. This means that their gradient estimates (3) will become negative and (4) will start increasing $N_{\mathbf{w}}$ and $N_{\mathbf{a}}$. An oscillatory pattern forms. For an example, see Figure 1. When this happens, we monitor the number of oscillations and as soon as it passes a certain threshold (which we empirically set to 10) we fix the bit-widths to $\lceil N_{\mathbf{w}} \rceil$ and $\lceil N_{\mathbf{a}} \rceil$, respectively, and continue the rest of the quantization process in standard QAT fashion.

IV. Experiments

To evaluate the effectiveness of AdaQAT, we conduct several mixed-precision quantization experiments on the CIFAR-10 [24] and the ImageNet [25] datasets and compare the results with those obtained with other mixed-precision quantization methods from the state-of-the-art.

A. Experimental Setup

Datasets We use the CIFAR-10 and ImageNet datasets for our experiments. We only perform basic data augmentation on the training dataset [26], which includes (in PyTorch parlance) *RandomResizedCrop* and *RandomHorizontalFlip* during training, and a single-crop operation during evaluation for ImageNet.

Networks We use a ResNet20 [27] model on CIFAR-10 and a ResNet18 [27] one on ImageNet. Following the practice adopted by prior work regarding greater sensitivity to quantization at the input and output of a network (see for instance [10]), we fix the bit-width to 8 bits in the first and last layers.

Training Settings We use an SGD optimizer with a batch size of 256, weight decay set to 10^{-4}, and momentum to 0.9. In the training from scratch scenario weights are initialized using the Kaiming method [28]. We use a cosine annealing learning rate scheduler with initial learning rate set to 0.1 for the from scratch scenario and 0.01 for the fine-tuning scenario. Training is run for 150 epochs in the fine-tuning scenario and 300 epochs when starting from scratch. We use PyTorch 1.13 for all experiments. The ImageNet tests are run on a cluster of 8 NVIDIA V100 GPUs, whereas the CIFAR-10 ones use a single GPU configuration.

B. Comparison with State-of-the-Art Methods

Table I shows the results of applying AdaQAT on CIFAR-10 compared to other methods from the literature. The first line shows the floating-point baseline result, whereas the second group of lines showcases *static* methods, where activations are not quantized and weights are quantized uniformly to 2 bits. The third group of lines corresponds to mixed-precision methods where the weight bit-width is learned and the activations are not quantized. AdaQAT with learned weight bit-width (2 bits) and unquantized activations is on par with the best of these, both when starting from a pretrained full-precision model as well as from scratch. We should nevertheless note that the FracBits results were obtained without fine-tuning its hyperparameters as much as possible.

The last two groups of lines in Table I illustrate the behaviour of our method when the activations are also quantized. The accuracy results are still competitive, either when starting from a pretrained model or from scratch. Even though the WCR metric is not as good as that of SDQ, it is more than compensated by the reduction in activation bit-width. It directly impacts how much memory (the BitOps column) gets transferred from one layer of the network to the next, going from 2.61 down to 0.51, more than a $5\times$ improvement.

Table II shows similar results on ImageNet compression. The quality of the obtained quantization is comparable to other methods from the state of the art. SDQ uses knowledge distillation with a ResNet-101 model as teacher, coupled with color jitter data augmentation, leading to better accuracy.

C. Balancing Parameter Impact

The hyperparameter λ dictates how the task loss $\mathcal{L}_{\text{Task}}$ and the hardware complexity $\mathcal{L}_{\text{Hard}}$ are balanced out in the total

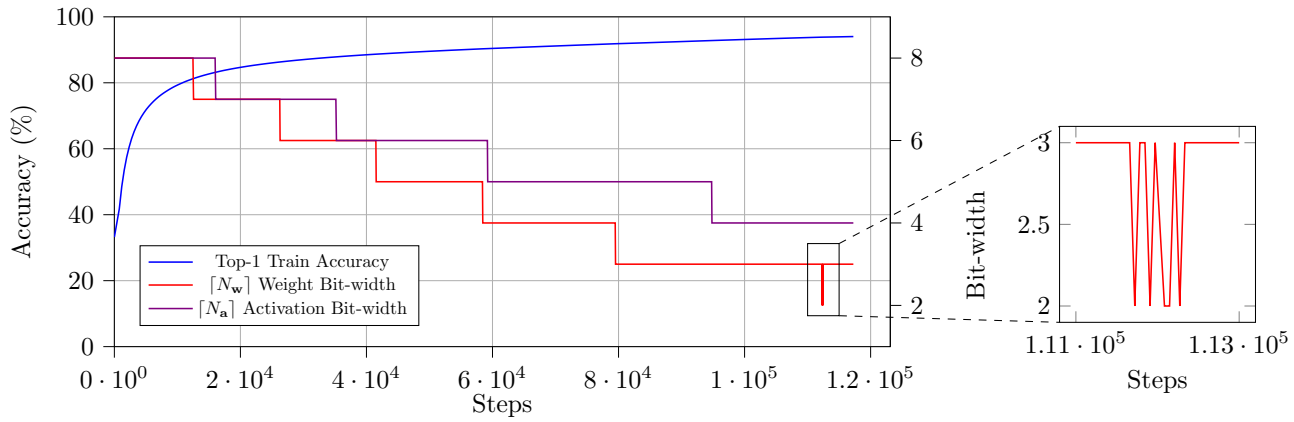

Fig. 1. Example of applying our approach with a ResNet20 network on the CIFAR-10 dataset. It showcases the evolution of the train accuracy with respect to updating the bit-width parameters $\lceil N_{\mathbf{w}} \rceil$ and $\lceil N_{\mathbf{a}} \rceil$ and how an oscillatory pattern can form (here, for the weight bit-width $\lceil N_{\mathbf{w}} \rceil$). When oscillations appear, we fix the value of the corresponding bit-width to the largest of the two oscillation points for the rest of the QAT process, considering that it has converged.

TABLE I
Comparison with state-of-the-art quantization methods (ResNet20 on CIFAR-10). Bit-width (W/A) denotes the average bit-width for weights and activation signals, whereas WCR represents the weight compression rate w.r.t. baseline. BitOPs denotes the bit operations metric (see Sec. III-B). The 4-bit activation result is learned using our method ($\lambda = 0.15$), whereas the activation bit-widths are fixed in the 8-bit and 32-bit settings, with only the weight bit-widths being learned.

Method	Bit-width (W/A)	top-1 (%)	Δ_{acc} (%)	WCR	BitOPs (Gb)
baseline	32/32	92.4	-	-	41.7
DoReFa [5]	2/32	88.2	-4.2	16×	2.7
PACT [17]	2/32	89.7	-2.7	16×	2.7
LQ-Net [29]	3/3	91.6	-0.5	10.7×	0.39
FracBits [10]	2.00/32	89.6	-2.8	16×	-
TTQ [30]	2.00/32	91.1	-1.2	16×	-
SDQ [9]	1.93/32	92.1	-0.3	16.6×	-
HAWQ-V1 [11]	3.89/4	92.2	-0.2	8.2×	0.67
Ours (fine-tuning)	2/32	92.0	-0.4	16×	2.7
	3/8	92.1	-0.3	10.7×	0.99
	3/4	**92.2**	**-0.2**	**10.7**×	**0.51**
Ours (from scratch)	2/32	91.8	-0.6	16×	2.7
	3/8	91.8	-0.6	10.7×	0.99
	3/4	**92.1**	**-0.3**	**10.7**×	**0.51**

TABLE II
Comparison with state-of-the-art quantization methods on the ImageNet dataset with ResNet18 in a fine-tuning setting. We set λ in our approach to 0.15.

Method	Bit-width (W/A)	Accuracy(%) top-1	Accuracy(%) FP top-1	WCR	BitOPs (Gb)
DoReFa [5]	4/4	68.1	70.5	8×	35.2
PACT [17]	4/4	69.2	70.5	8×	35.2
LQ-Net [29]	4/4	69.3	70.3	8×	35.2
FracBits [10]	4.00/4.00	70.6	70.2	8×	34.7
SDQ [9]	3.85/4	71.7	70.5	8.9×	33.4
HAWQ-V3 [19]	4.8/7.5	70.4	71.5	6.7×	72.0
Ours	**4/4**	**70.3**	**70.5**	**8**×	**35.2**

TABLE III
Evolution of AdaQAT mixed-precision quantization results on CIFAR-10 with respect to λ.

λ	W	A	top-1
0.2	2	4	91.7
0.15	3	4	92.1
0.1	4	5	92.3

loss $\mathcal{L}_{\text{Total}}$ (see eq. (2)). It controls how much accuracy loss is allowed in the final DNN model compared to the W/A quantization levels. As can be seen in Table III, a larger λ leads to more compression, but less accurate test results as well. Its value should be chosen carefully on a model-by-model basis, taking into account the application constraints (*i.e.,* how much accuracy degradation is allowed versus a certain level of attainable compression).

V. Conclusion & Future Work

We have introduced AdaQAT, an optimization-based method for mixed-precision quantization. Compared to previous approaches that are generally intended to be used in a fine-tuning setting, in early tests AdaQAT seems to be more flexible, being capable of operating in both fine-tuning and training from scratch scenarios, producing results that are on par with state-of-the-art mixed-precision quantization approaches on CIFAR10 with a ResNet20 network. It also performs well in mixed-precision fine-tuning of ResNet18 on ImageNet.

As future work, we will evaluate AdaQAT on other network types that are more sensitive to quantization (*e.g.* the MobileNet family of models). Currently, bit-width assignment is done on a per-network basis. Our goal is to generalize the approach to cover a much larger design space. One direction is to look at finer levels of mixed-precision quantization granularity, such as per-layer and per-channel. We also intend to explore finer hardware complexity and energy consumption metrics, tailored for a specific target architecture (*e.g.* FPGAs), in the $\mathcal{L}_{\text{Hard}}$ term.

REFERENCES

[1] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *arXiv:1510.00149*, 2015.

[2] E. Dupuis, D. Novo, I. O'Connor, and A. Bosio, "On the Automatic Exploration of Weight Sharing for Deep Neural Network Compression," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 1319–1322.

[3] F. Tung and G. Mori, "Similarity-Preserving Knowledge Distillation," *IEEE/CVF Int. Conf. on Computer Vision*, pp. 1365–1374, 2019.

[4] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[5] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients," *arXiv:1606.06160*, 2016.

[6] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to $+1$ or -1," *arXiv:1602.02830*, 2016.

[7] F. Li, B. Zhang, and B. Liu, "Ternary Weight Networks," *arXiv:1605.04711*, 2016.

[8] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "HAQ: Hardware-Aware Automated Quantization With Mixed Precision," *IEEE/CVF Int. Conf. on Computer Vision*, pp. 8604–8612, 2019.

[9] X. Huang, Z. Shen, S. Li, Z. Liu, H. Xianghong, J. Wicaksana, E. Xing, and K.-T. Heng, "SDQ: Stochastic Differentiable Quantization with Mixed Precision," *Int. Conf. on Machine Learning*, pp. 9295–9309, 2022.

[10] L. Yang and Q. Jin, "FracBits: Mixed Precision Quantization via Fractional Bit-Widths," *AAAI Conf. on Artificial Intelligence*, pp. 10 612–10 620, 2021.

[11] Z. Dong, Z. Yao, A. Gholami, M. Mahoney, and K. Keutzer, "HAWQ: Hessian Aware Quantization of Neural Networks with Mixed-Precision," *IEEE/CVF Int. Conf. on Computer Vision*, pp. 293–302, 2019.

[12] R. Banner, Y. Nahshan, and D. Soudry, "Post training 4-bit quantization of convolutional networks for rapid-deployment," *Neural Information Processing Systems*, vol. 32, 2019.

[13] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training Deep Neural Networks with binary weights during propagations," *Neural Information Processing Systems*, vol. 2, pp. 3123–3131, 2015.

[14] Y. Bengio, N. Léonard, and A. Courville, "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation," *arXiv:1308.3432*, 2013.

[15] S. Esser, J. McKinstry, D. Bablani, R. Appuswamy, and D. Modha, "Learned Step Size Quantization," *arXiv:1902.08153*, 2019.

[16] Y. Bhalgat, J. Lee, M. Nagel, T. Blankevoort, and N. Kwak, "LSQ+: Improving Low-Bit Quantization Through Learnable Offsets and Better Initialization," *IEEE/CVF Int. Conf. on Computer Vision*, pp. 696–697, 2020.

[17] J. Choi, Z. Wang, S. Venkataramani, P. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: Parameterized Clipping Activation for Quantized Neural Networks," *arXiv:1805.06085*, 2018.

[18] B. Wu, Y. Wang, P. Zhang, Y. Tian, P. Vajda, and K. Keutzer, "Mixed Precision Quantization of ConvNets via Differentiable Neural Architecture Search," *arXiv:1812.00090*, 2018.

[19] Z. Yao, Z. Dong, Z. Zheng, A. Gholami, J. Yu, E. Tan, L. Wang, Q. Huang, Y. Wang, and M. Mahoney, "HAWQ-V3: Dyadic Neural Network Quantization," *Int. Conf. on Machine Learning*, pp. 11 875–11 886, 2021.

[20] Y. Ma, T. Jin, X. Zheng, Y. Wang, H. Li, Y. Wu, G. Jiang, W. Zhang, and R. Ji, "OMPQ: Orthogonal Mixed Precision Quantization," *arXiv:2109.07865*, 2021.

[21] J. Liu, J. Cai, and B. Zhuang, "Sharpness-Aware Quantization for Deep Neural Networks," *arXiv:2111.12273*, 2021.

[22] M. Nikolić, G. Hacene, C. Bannon, A. Lascorz, M. Courbariaux, Y. Bengio, V. Gripon, and A. Moshovos, "BitPruning: Learning Bitlengths for Aggressive and Accurate Quantization," *arXiv:2002.03090*, 2020.

[23] Q. Jin, L. Yang, and Z. Liao, "Towards efficient training for neural network quantization," *arXiv preprint arXiv:1912.10207*, 2019.

[24] A. Krizhevsky and G. Hinton, "Learning Multiple Layers of Features from Tiny Images," University of Toronto, Tech. Rep., 2009.

[25] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," *IEEE/CVF Int. Conf. on Computer Vision*, pp. 248–255, 2009.

[26] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-Supervised Nets," *Int. Conf. on Artificial Intelligence and Statistics*, vol. 38, pp. 562–570, 2015.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE/CVPR Conf. on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[28] ——, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *IEEE Int. Conf. on Computer Vision*, 2015, pp. 1026–1034.

[29] D. Zhang, J. Yang, D. Ye, and G. Hua, "LQ-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 365–382.

[30] S. Jain, A. Gural, M. Wu, and C. Dick, "Trained Quantization Thresholds for Accurate and Efficient Fixed-Point Inference of Deep Neural Networks," *Machine Learning and Systems*, vol. 2, pp. 112–128, 2020.