Fast, Quantization Aware DNN Training for Efficient HW Implementation

Daniel Schnöll, Matthias Wess, Matthias Bittner, Maximilian Götzinger and Axel Jantsch
Christian Doppler Laboratory for Embedded Machine Learning, Institute of Computer Technology
TU Wien, 1040 Vienna, Austria
{firstname}.{lastname}@tuwien.ac.at

***Abstract*—Quantization of Deep Neural Networks is a central technique to reduce the computation load in embedded devices. Even in quantized Deep Neural Networks (DNNs), the scaler/rescaler following a convolution or dense layer often requires a high bit width multiplication and a shift. Previous work has proposed to remove the multiplier by restricting the quantization method. We propose a Quantisation Aware Training (QAT) approach, which explicitly models the rescaler during training, eliminating the limitations of quantization functions and achieving a 30 – 35% improvement in training time and a significant reduction in memory requirements compared to the state-of-the-art. GitHub:https://github.com/embedded-machine-learning/FastQATforPOTRescaler**

***Index Terms*—Convolution, Training, Neural networks, Quantization (signal), Quantization Aware Training, hardware-friendly**

I. Introduction

Due to resource constraints, inference of deep neural networks on edge devices remains challenging. Various optimization approaches have been proposed, including approximate matrix multiplications, pruning, and quantization. While pruning focuses on reducing the model size by removing unnecessary connections and weights, approximate matrix multiplications aim to perform computationally expensive operations more efficiently. On the other hand, quantization offers a dual benefit by reducing the model size while enabling more efficient hardware utilization. For example, NVIDIA claims up to 4× acceleration when transitioning from FP32 to INT8[1]. Using a Field programmable Gate Array (FPGA) or Application-Specific Integrated Circuits (ASIC) could further improve the performance, particularly because the size of a standard multiplier scales with the squared number of bits.

A. Operations of an Accelerator

Dense/Convolutional Layers require matrix multiplications, which in turn require Multiply Accumulate (MAC) operations (Fig. 1). A MAC operation is part of a Digital Signal Processor (DSP). The MAC is characterized by the number of bits for the weights and the feature maps. For instance, a common notation is A8W4, which indicates that the activation function produces an 8-bit value (typically the feature map for the following input), and the weights are represented in a 4-bit format. The Accumulation register is typically implemented using a 32-bit data type or is assumed to be large enough to avoid overflow.

FPGAs often have DSPs. These units have predefined sizes. For example, the Xilinx ultra-96 has 350 28x18-bit multiplier with a 48-bit accumulation register[2]. Since these DSPs are explicitly implemented in hardware, they can not be changed. However, they can be used in a modified manner, e.g., more energy efficient or multiple multiplications at once.

The matrices in Deep Neural Networks (DNNs) are often quite large. Therefore, matrix multiplications require numerous multiplications, making them computationally demanding. Minimizing the number of multiplications in other parts of the network can significantly enhance the overall performance. Other layers within DNNs that rely on multiplication operations include the Batch Normalization (BN)-Activation functions, which result in rescaling operations in hardware. Hence, reducing multiplications in these layers can improve the overall inference speed in DNNs.

B. Rescaler

The role of the Rescaler in DNN accelerators is to reduce the accumulation register to match the bit-width of the feature

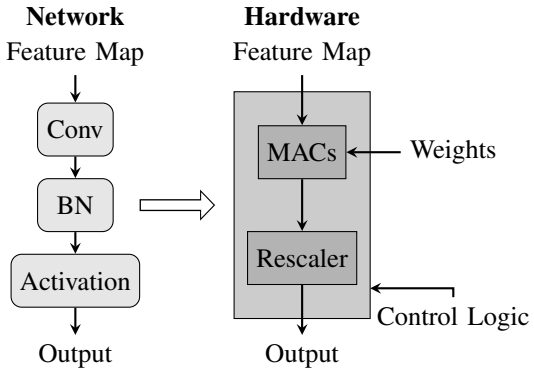

Fig. 1. Network operations and the HW Blocks for their implementation. (Conv=Convolution, BN=Batch Normalization, MAC=Multiply-Accumulate Unit)

This work was supported in part by the Austrian Federal Ministry for Digital and Economic Affairs, in part by the National Foundation for Research, Technology and Development, and in part by the Christian Doppler Research Association.

[1]Accessed: 10th of May 2023 https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/

[2]Accessed: 3rd of May 2023 https://docs.xilinx.com/v/u/en-US/ug579-ultrascale-dsp

2771-2508/23/$31.00 ©2023 IEEE
DOI 10.1109/DSD60849.2023.00100

TABLE I
APPROXIMATE CHARACTERISTICS FOR 10 nm TECHNOLOGY (BASED ON [3]).

Rescaler HW Costs	Resources		
	2-NAND # gates	*Area* μm^2	*Power* pJ
Components:			
NAND	1	0.042	0.0025
2:1 MUX	3	0.125	0.008
8x8 Multiplier	274	11.4	0.696
Rescalers:			
With 32x32 Mult.	4928	205.07	12.52
With 32x8 Mult.	1640	68.37	4.16
Power of 2	544	22.67	1.38

map and apply the non-linearity. Typically, this involves a 32-bit multiplication followed by a shift operation [1]. The resulting value is then clipped to a specific value domain, e.g., 0 – 255. However, performing a 32-bit multiplication is computationally expensive. Alternatively, a power of 2 Rescaler can be used to avoid multiplication and only perform the arithmetic shift operation [2].

Table I contains ballpark estimations for different Rescalers to illustrate the potential impact of about 70 – 90% reduction in hardware complexity.

If the power of 2 Rescaler offers significant benefits, it raises the question of why it is not the default approach. We have identified two major challenges associated with the power of 2 Rescaler:

1) Restrictions on quantization method: The current approach to achieving a power of 2 Rescaler is to limit each quantization method to a power of 2. This is not always straightforward, as exemplified by the amount of work done in [4] to implement PArameterized Clipping acTivation (PACT).
2) Long training times: Effective Quantisation Aware Training (QAT) often requires two passes through a convolution/dense layer. This process, including quantization, nearly doubles the training time and requires approximately 40 – 50% more GPU RAM.

C. Contribution

In this paper, we address both of these challenges. We dynamically modify the linear weight quantization method during training to enforce a desired Rescaler (Fig. 2). This modification is applicable to any linear quantization method. Through the well-defined behavior, we can incorporate an approximation that eliminates the need for a second forward pass during training.

Furthermore, we showcase our approach with a power-of-2 Rescaler, but it is important to emphasize that our method is not limited to it. Our approach offers extensive flexibility, for instance:

- One can use channel-wise quantization with a layer-wise Rescaler,

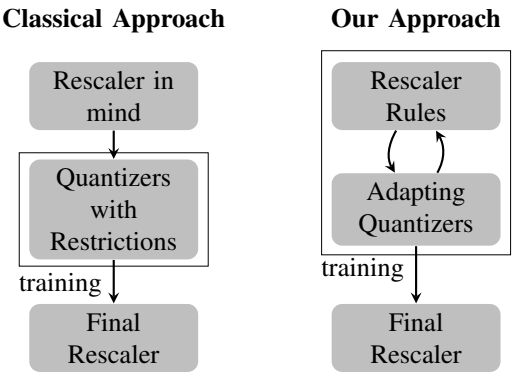

Fig. 2. Classical approach compared to our approach and which information we require for training.

- or pair any weight and output quantization method, without additional implementation effort, while still enforcing a specific Rescaler.
- One is free to tailor the Rescaler to the needs or capabilities of the hardware. For example, one may choose to only use prime numbers as rescaling factors or limit the network to only two rescaling factors.

II. RELATED WORK

Jacob *et al.* introduced two significant aspects in [1], "fake Quantization" and "Batch-norm folded Quantization Aware training," also sometimes called double forward for BN fusion.

Fake quantization is a widely used technique during training in which a value is converted to the quantized domain and then restored to full precision, thereby including all the quantization errors of the value. This approach has become standard in QAT. In this paper, we define it as

$$F(x, S, b) := S \left\lfloor \text{clamp}\left(\frac{x}{S}, -2^{b-1}, 2^{b-1}-1\right)\right\rceil \quad (1)$$

for signed values and

$$F(x, S, b) := S \left\lfloor \text{clamp}\left(\frac{x}{S}, 0, 2^{b}-1\right)\right\rceil \quad (2)$$

for unsigned values, where $\text{clamp}(x, m_1, m_2) = \max(m_1, \min(x, m_2))$. In both cases, x is the value to quantize, S is the scaling value from the quantization method, and b is the number of bits.

How to handle a BN is quite an important question in the quantized domain. There are multiple approaches, depending on which parts of the BN should be updated during training. On the one hand, Yao *et al.* suggest in [5] fully freezing and fusing the BN into the weights of the preceding linear or convolutional layer. This approach is often called fine-tuning and requires significant modifications to hyper-parameters. On the other hand, Jacob *et al.* in [1] and later refined by Krishnamoorthi *et al.* in [6] keep the BN unfrozen and fully capable of adapting. They essentially use two passes through the preceding layer, one in full precision to update the BN and a second one in a fused, quantized manner. Fig. 3 displays the data paths through such a double forward for

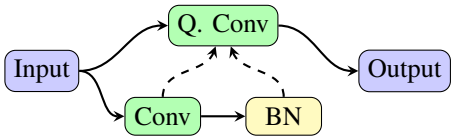

Fig. 3. Double forward and the paths the data takes.

BN fusion approach. Li *et al.* goes further into detail in [7], beautifully summarizing multiple approaches and their minute differences. Inside the PyTorch repository there exists a QAT implementation [3] of a modified [6], which does not require two passes while keeping the BN unfrozen. To our knowledge, no paper is associated with this implementation, reaffirmed by [7].

Contrary to the approaches above, Nagel *et al.* suggests in [8] to keep the BN in full precision and fuse it into the activation function, thereby also requiring only one pass.

Power of 2 Rescaler is desirable as it eliminates the need for multiplication and renders a simple arithmetic shift sufficient. Previous research, such as in [2], achieved this with

$$S_x, S_w, S_o \in \{2^x | x \in \mathbb{Z}\}, \tag{3}$$

$$2^n = \frac{S_x S_w}{S_o}, \tag{4}$$

where S_x, S_w, and S_o are the scaling factors for the input, weights, and output, respectively. All scaling factors are a power of 2 by restricting the quantization methods, which results in a power-of-2 Rescaler.

F8NET is presented by Lin *et al.* in [4]. It is a current implementation of a power of 2 Rescaler. They use double forward for BN fusion. Their quantization functions are all a power of 2. They enabled PACT [9] by fusing its parameter into the weights. However, this procedure leads to an overhead, which is one of the problems we want to tackle.

Our approach differs from the previous work in two main aspects.

A. The Rescaler is a combination of multiple values. We do not restrict all of them; instead, we modify one so that the combination produces a power of 2.
B. We approximate the first pass-through in double forward for BN fusion, thereby removing it.

III. Enforcing a Rescaler

We start with a regular Convolution BN Activation path. Let X be the input for the convolution layer, W its weights, and O the output of the activation function A. We define our BN as

$$\mathrm{BN}(u) := \frac{u-\mu}{\sqrt{\sigma^2+\epsilon}}\gamma + \beta, \tag{5}$$

with the mean μ, the variance σ^2, the weights γ, and the bias β. ϵ exists to avoid division by 0 and is usually $1\mathrm{e}{-5}$.
The Activation function is a simple non-linearity such as Rectified Linear Uni (ReLU), ReLU6, and PACT. We use fake quantization with the scaling factors S_x for the input, S_w for the weights, and S_o for the output. We assume a full channel-wise quantization. Therefore they are vectors. We split the input quantization into a channel-wise part and an average part. We fuse the channel-wise aspect into the weights,

$$\begin{aligned} \overline{S}_x &:= 2^{\mathrm{mean}(\log_2(S_x))}, \\ \hat{S}_x &:= \frac{S_x}{\overline{S}_x}, \\ W' &:= \hat{S}_x W. \end{aligned} \tag{6}$$

S_w is only dependable on these weights,

$$S_w := \mathrm{LinearQuantizationMethod}(W'). \tag{7}$$

Considering these aspects, the Conv-BN-Act path can be defined as

$$O_{\mathbb{Z}} = \mathrm{A}_{\mathbb{Z}}\left(\frac{\gamma \overline{S}_x S_w (X_{\mathbb{Z}} * W'_{\mathbb{Z}})}{S_o\sqrt{\sigma^2+\epsilon}} + \beta'\right), \tag{8}$$

$$\beta' = \frac{\gamma\mu}{S_o\sqrt{\sigma^2+\epsilon}} + \frac{\beta}{S_o}, \tag{9}$$

where subscript $\mathbb{Z}$ represents the integer domain.

The Rescaler is the multiplicative aspect, which we want to define as a power of 2. However, our approach aims to calculate the multiplicative difference to the power of 2 rather than forcing all components to a power of 2. To represent this multiplicative difference and the right-shift value n, we can use the factor ϕ as

$$\frac{\gamma \overline{S}_x S_w}{S_o\sqrt{\sigma^2+\epsilon}} = \phi 2^{-n}. \tag{10}$$

The next steps require HW information. For example, if the accumulator is a 32-bit register, we can define the range of n as $[0, 31]$. Because a right shift is a flooring division, a left-sided 0 in (10) is comparable to a right shift of 31. Now we can include the rules for the Rescaler. We provide three examples of how these rules can look like:

- Channel-wise Rescaler:

$$\begin{aligned} n'' &:= -\log_2\left(\frac{\gamma \overline{S}_x S_w}{S_o\sqrt{\sigma^2+\epsilon}}\right) \\ n' &:= \mathrm{clamp}(n'', 0, 31) \\ n &:= \lfloor n' \rfloor \\ \phi &:= 2^{n-n'} \end{aligned}$$

  This means that ϕ is always in the range of $[0.5, 1]$.
- Layer-wise Rescaler with channel-wise quantization:

$$\begin{aligned} n'' &:= -\log_2\left(\frac{\gamma \overline{S}_x S_w}{S_o\sqrt{\sigma^2+\epsilon}}\right) \\ n' &:= \mathrm{clip}(n'', 0, 31) \\ n &:= \mathrm{median}(\lfloor n' \rfloor) \\ \phi &:= 2^{n-n'} \end{aligned}$$

- A fixed number of Rescalers per network or one fixed Rescaler: A fixed number of Rescalers or a single fixed Rescaler would be the extremest form. n would need to

[3]Last Accessed 14th of May 2023: https://github.com/pytorch/pytorch/blob/main/torch/ao/nn/intrinsic/qat/modules/conv_fused.py

be defined across Layers and could closely follow the Layer-wise approach.

ϕ is now everything which can not be expressed by the Rescaler. This would usually be fused into the weights. However, we suggest fusing it into the weight quantization factor. The reason for this is twofold:

1) It is an automatic "BN un-fusion", which is expected to improve accuracy [8].
2) The number of operations is significantly lower than fusing them into the weights and un-fusing them from the feature map. E.g., A Convolution layer, traditionally[4]: $C_{\mathbf{in}} \times C_{\mathbf{out}} \times K^2$ multiplications and $C_{\mathbf{in}} \times C_{\mathbf{out}} \times W \times H$ divisions compared to our approach $C_{\mathbf{out}}$ multiplications (the fake quantization function does the rest, which is present in both).

Applying this approach results in a modified weight quantization factor,

$$S'_w := \frac{S_w}{\phi(\gamma, \sigma, \overline{S}_x, S_w, S_o)}. \tag{11}$$

The state of the art would require S_x, S_o to be a power of 2, thereby restricting the allowed values. Our approach does not limit them and enables them to use their desired values.

The final combined Conv-BN-Act path during training is:

$$S_w = \mathrm{LinearQuantizationMethod}(\hat{S}_x W), \tag{12}$$
$$W'' = F(W, S'_w/\hat{S}_x, \text{Weight bits}), \tag{13}$$
$$O = F\left(\mathrm{A}\left(\mathrm{BN}\left(X * W''\right)\right), S_o, \text{Act. bits}\right), \tag{14}$$

and during inference:

$$\beta''_{\mathbb{Z}} = \left\lfloor 2^n \left(\frac{\gamma\mu}{S_o\sqrt{\sigma^2+\epsilon}} + \frac{\beta}{S_o} \right) \right\rceil, \tag{15}$$
$$O_{\mathbb{Z}} = \mathrm{A}_{\mathbb{Z}}\left(\left\lfloor 2^{-n}(X_{\mathbb{Z}} * W''_{\mathbb{Z}} + \beta''_{\mathbb{Z}}) \right\rfloor\right). \tag{16}$$

Note that all scaling factors are assumed as vectors/tensors and need to be used in their corresponding axes, e.g. $S'_w/\hat{S}_x$ would be a $C_{\mathbf{in}} \times C_{\mathbf{out}} \times 1 \times 1$ tensor.

IV. Accelerating training

Accelerating training is the second primary task. One of the drawbacks of double forward for BN fusion is the necessity to calculate the convolution twice, which also means storing data twice. This overhead reduces training speed and increases memory overhead significantly. In [8], Nagel *et al.* argues that folding the BN is unnecessary for the particular case of channel-wise quantization with a full precision Rescaler. We fully agree, and our equations come to the same conclusion (ϕ would always be 1, meaning the neutral element for multiplication and division[5]). In our case, S'_w is already a partial fusion. If, for a moment, we ignore the dependencies of ϕ, the multiplicative components,

$$\frac{\gamma\overline{S}_x S'_w}{S_o\sqrt{\sigma^2+\epsilon}}, \tag{17}$$

could be seen as full precision per-channel quantization, and the Rescaler "happens" to always fall on a power of 2. That, in turn, would mean no double forward would be required. Nonetheless, the dependencies do exist, namely

$$\phi(\gamma, \sigma, \overline{S}_x, S_w, S_o). \tag{18}$$

The question is whether they change during a forward pass. γ updates per step and thus can be considered fixed. Moreover, S_x, the scaling factor of the input, can be seen as semi-fixed as the point of updating the value is before requiring it. S_w depends on the convolution weights and S_x, which means it is semi-fixed too. This leaves σ and S_o, which get updated after they are required. However, they are approximable as described in the following.

σ, the running variance is used for fusion. It should represent the average variance of a layer for the whole dataset. The stability of the running values is assumed by regular full-precision training. The only time it does not hold is at the early stages of training, with high learning rates and rapidly changing values. However, we argue that it is unnecessary to accurately quantize early during the training because a perfect representation of values is not required if their ballpark is incorrect. Once the training converges, the running parameters of the BN should stabilize. Therefore, it appears as approximately constant from one step to the next, which means that the last step approximates the current step well.

The output quantization factor S_o should stabilize as well. The quantization is desired to be unchanging during device inference. Therefore a value optimal for the whole dataset is desired. A few approaches exist, such as low pass filters over the extracted information (e.g., Exponential Moving Average (EMA)), using BN information, or a trainable parameter (e.g., PACT), which all fall into the previous assumptions.

It is important to note that there is a difference between the running values of the BN and the values of the current batch. This difference can still cause problems, as described in [6]. We are only arguing for the running values.

To our knowledge, no paper exists using such an approach, but we found an implementation inside the PyTorch repository as described in the related work. Compared to our approach, PyTorch is using [6] and accepts the inaccuracies of using the last step. There is a problem when the BN weights reach 0 (division by 0). A full fusion into the weights is made before passing them into the quantization function, so if the scales change quickly, the quantization function needs to adapt quickly, especially with a sign change in the BN weights. In other words, stabilized or trained quantization is questionable for such an implementation.

The existence of this implementation further reassures us that the made assumptions likely hold, even if we find it strange that no paper is associated with it.

[4]Following the common names of C for channels, K for Kernel, and W/H for feature-map width/height.

[5]The modification would be in (10), 2^{-n} is replaced by a full precision number so ϕ would result in the question: "Which parts of full precision can not be represented by full precision?"

V. Experiments

Unless otherwise mentioned, all trainings are done from scratch to display the quantization error dominantly. Since this paper focuses on the speed and efficiency of the training, we must track the resources used and the time spent. This requires a highly reliable and controlled environment. We use our development server, which has two NVIDIA V100 with 32GB RAM each. By staying on one server, we exclude network interference, which can occur during regular cluster training. Consequently, big datasets and extensive networks have enormous training costs. To remove data transfer limitations, we load all datasets into RAM. CIFAR10 [10] and CIFAR100 [10] do this automatically, as they are tiny. For Imagenet1k [11], we created a directory on a RAM drive, copied train/eval, and symbolically linked the compressed files.

We first define the quantization functions in the following (Section V-A). Then, we show accuracy, memory requirements, and training time for small networks (Section V-B) as well as larger networks (Section V-C). Section V-D demonstrates our method's significant speedup and reduction in memory usage. Finally, Section V-E illustrates the possibilities and effects of flexible rules imposed on the Rescaler.

A. Quantisation Functions Used

1) F8: From F8NET [4], we use the quantization method but remove the limiting rounding in the logarithmic domain. We define it as

$$S_{\text{F8}} := \frac{\sigma}{40} \qquad \text{(signed)}, \tag{19}$$

$$S_{\text{F8}} := \frac{\sigma}{70} \qquad \text{(unsigned)}. \tag{20}$$

2) MinMSE: As F8NET is only defined for 8-bit, we follow a similar approach with our example quantization function MinMSE:

$$S_{\text{MinMSE}} := \sigma 3.347 e^{-0.5739\text{bits}} \qquad \text{(signed)}, \tag{21}$$

$$S_{\text{MinMSE}} := \sigma 1.688 e^{-0.5813\text{bits}} \qquad \text{(unsigned)}. \tag{22}$$

We fitted this function by a brute force search on a Gaussian distribution to minimize the mean squared error.

3) OCTAV: To also represent a different approach, we implemented OCTAV [12]. It defines the quantization level by an iterative approach.

$$s_{n+1} := \frac{\sum_x \left[|x| \cdot \mathbb{1}_{\{|x|>s_n\}}\right]}{\frac{4^{-\text{bits}}}{3} \sum_x \left[\mathbb{1}_{\{0<|x|\leq s_n\}}\right] + \sum_x \left[\mathbb{1}_{\{|x|>s_n\}}\right]} \tag{23}$$

$$S_{\text{OCTAV}} := \frac{s}{2^{\text{bits}-1}} \tag{24}$$

We found that with a few values to quantize, it can start to oscillate between values. We added an EMA filter over the generated values to suppress this behavior and then increased the maximum iterations to 1000. We allowed an early exit if the relative change was less than $1\text{e}{-3}$. These modifications stabilize the algorithm but, at the same time, slow it down. If the stabilized version is used, it is marked as such.

4) Activation functions: We are using PACT [9] for most activation functions,

$$\text{PACT}(x, \alpha) := \begin{cases} 0 & : x < 0 \\ x & : 0 \leq x \leq \alpha \\ \alpha & : x > \alpha \end{cases}. \tag{25}$$

We use F8 signed for down-scaling if no activation function is directly behind the BN. This is the case inside a ResNet block [13], where the non-linearity follows the addition of two 'unrestricted' feature maps. Those feature maps must have the same or by a power of 2 separated scaling factors. We follow a similar approach as F8NET and scale one by the other. Except for PACT, we use exclusively straight-through estimation.

B. Small Network Experiments

We tested VGG-small [14], ResNet8 [15] structure but without the feature embedding, and the baseline model of the modified Mobilenet-V2 [16], on the datasets CIFAR10 [10] and CIFAR100 [10], with the results shown in figures 4, 5 and 6 for accuracy, memory, and training time, respectively. Please note that these networks are designed for CIFAR10 and not for CIFAR100. As these networks train quickly, we can make a broader test with quantization functions and hyper-parameters. Using cosine annealing, we tested learning rates of 0.1; 0.02; 0.01; 0.001. For VGG-small and Mobilenet-v2 modified, the number of bits for weights and activations are W8A8; W4A8; W4A4, and for ResNet8, they are W8A8; W4A8. The number of epochs is 100, the weight decay is $5\text{e}{-4}$, and the batch size is 300. Everything is trained from scratch on one GPU, and two tests run in parallel. In contrast to the usual approach of preserving the first and last layer in full precision, we quantize the layers and the input/output to INT8. We only display training and not validation. The memory is measured by *torch.cuda.mem_get_info()*, the average of the whole epoch is taken, and Python's time package measures time.

The accuracy is fairly constant across the networks and hyper-parameters, Fig. 4. Unsurprisingly, the accuracy is slightly lower than full precision, as all are trained from scratch. VGG-small suffers the least as it is explicitly designed for quantization. ResNet8 quantizes nicely. Mobilenet-V2 modified behaves as expected since Mobilenet is known to quantize poorly without extra care [6], [8], [17], [18]. We attribute the improvement of accuracy for CIFAR100 to the regulative effect of quantization [19].

The memory overhead is quite surprising, Fig. 5, as quantized Mobilenet-V2 modified requires far less memory than full precision. This is highly unlikely, as we store all quantized values in full precision during training. Full precision Mobilenet requires about 17GiB of memory during training. As later displayed in Table III, a memory-optimized training state exists, which takes longer to train, but with less memory. However, it would be surprising if it would trigger this early, roughly 20 out of 32 GiB. It might also be possible that the inverse is happening. It could be that full precision is sped up at the cost of memory consumption. That a form of optimization is active is further supported by the training

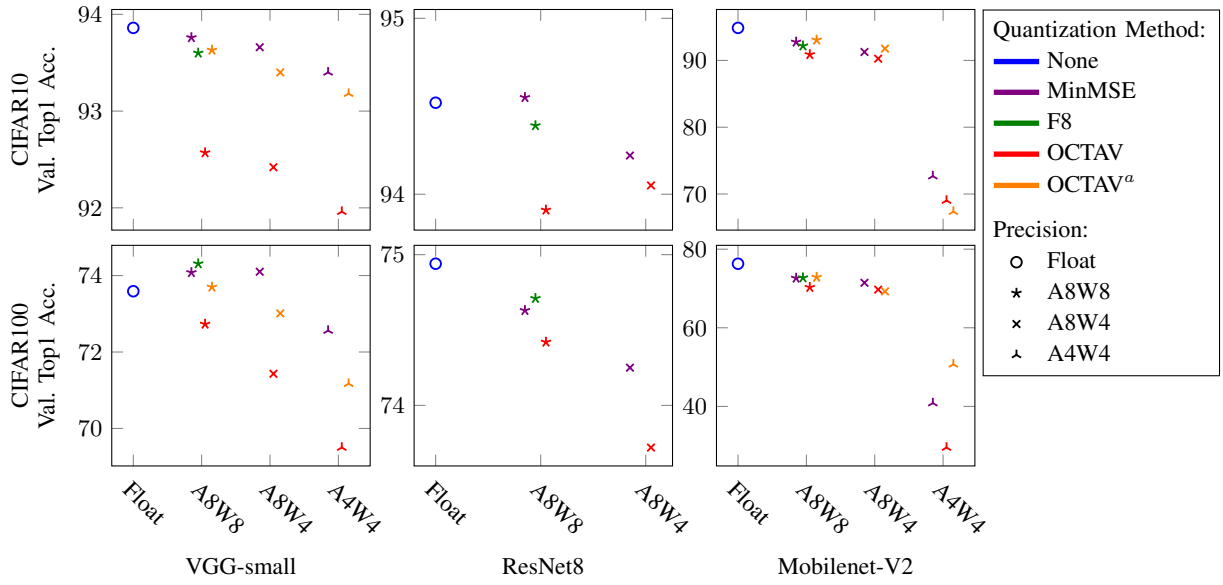

Fig. 4. The Validation Accuracy of the tested models, quantization methods, and precisions. Values are slightly displaced on the horizontal axis to increase visibility. Only the highest value per learning rate is displayed. The accuracies are as expected for from-scratch training. ^{a}Stabilized OCTAV.

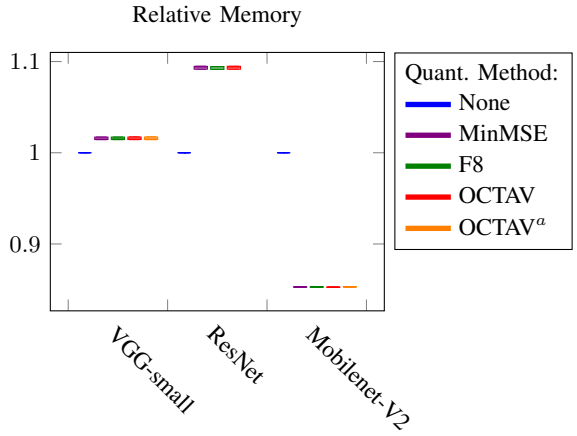

Fig. 5. The training memory requirement compared to Float, depending on the networks and quantization methods. The absolute memory for full precision VGG-small, ResNet8, and Mobilenet-V2 modified is about 7GiB, 7.5GiB, and 16.5GiB, respectively. ^{a}Stabilized OCTAV.

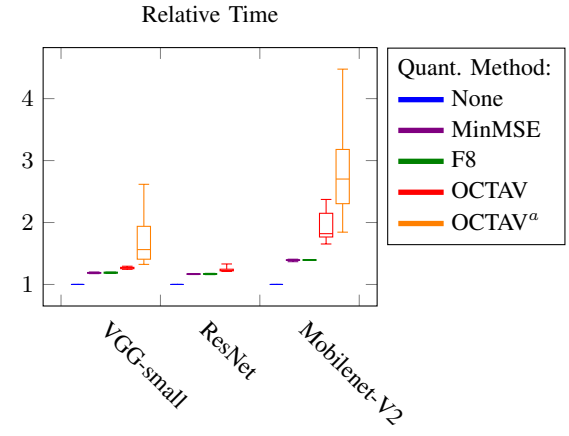

Fig. 6. The training time compared to Float, depending on the networks and quantization methods. The absolute amount of time per epoch for full precision VGG-small, ResNet8, and Mobilenet-V2 modified is about 12.5 seconds, 20.5 seconds, and 36 seconds, respectively. ^{a}Stabilized OCTAV.

time, Fig. 6 and Table II, which display a reasonably constant time overhead. Realistically speaking, we observed a memory overhead of about 2 – 10%.

Training time overhead is about 20% for VGG-small and ResNet8, Fig. 6. As expected iterative quantization approach takes longer than the other methods. Mobilenet-V2 modified takes relatively longer than all other tests, which, as already discussed, we attribute to an optimization. OCTAV, especially Stabilized OCTAV, requires many iterations for Mobilenet-V2 modified, as the weights of the dep-wise separated convolution are channel-wise quantized, meaning 9 values get one quantization factor. However, Stabilized OCTAV did achieve the highest accuracy for quantized Mobilenet-V2 modified.

C. ResNet18 on Imagenet1k

To study larger networks on a large data set, we tested ResNet18 [13] on Imagenet1k [11]. The training strategy is close to Torchvision[6]. We use 90 epochs of training, a step learning rate scheduler that reduces the learning rate by 0.1 every 30 epochs, and ten warm-up epochs of linear increasing learning rate to the initial learning rate of 0.1. The augmentations used are RandomResizeCrop[7] to 224 with otherwise default arguments and a random horizontal flip.

[6]Accessed 14th of May: https://github.com/pytorch/vision/tree/main/references/classification

[7]https://pytorch.org/vision/main/generated/torchvision.transforms.RandomResizedCrop.html

TABLE II
RESNET18 ON IMAGENET1K. THE VALIDATION ACCURACY IS AS ONE WOULD EXPECT FROM QAT. RELATIVE TRAINING RESOURCES ARE CLOSE TO FIG. 6 AND 5.

ResNet18 Imagenet1k	**Pre-Train**	**Train**	**Val.**		**Training Resources**	
	Val-Top1 %	***Top1*** %	***Top1*** %	***Top5*** %	***Time p.E.*** seconds	***Memory*** MiB
Float	–	68.9	68.6	88.5	535	18 292
F8 from Scratch	–	68.0	67.9	88.0	650	19 632
F8 from Full-Pr.	68.6	71.1	68.8	88.9	650	19 632
F8 from Torchvision	69.8	72.1	69.8	89.3	650	19 632

Moreover, we normalize it with a mean of 0.5 and a standard deviation of 0.225. The normalization is defined in this manner to quantize the input easily. We use a batch size of 512 per GPU and train on 2 GPUs. We use a weight decay of 4e-5, as suggested by Jin *et al.* [4]. In contrast to the usual approach of keeping the first and last layer in full precision, we quantize the layers and the input to INT8 and accept an INT32 output.

We are using ResNet-V1 [13], not V2 [20]. The essential difference is that in V2, the activation function is before the convolution rather than after the convolution in the basic Blocks. Reviewing F8NETs GitHub repository [8] [4], we conclude that they use ResNet-V2 variation d) BasicBlocks, with an additional INT32 ReLU after the addition. The drawback is that INT32 feature maps must be stored and processed during inference. Our approach enforces INT8/UINT8 feature maps in the BasicBlocks.

As expected, from-scratch training results in worse accuracy than full precision, while using full precision as initialization improves the accuracy (Table II). We also tested Torchvisions [9] weights and kept the accuracy. We noticed instability, as described by [6], which leads us to conclude that a similar training strategy would benefit us. The training time overhead is 21.5% which is highly similar to the previous tests, even though it is a multi-GPU test. The memory overhead is 7.3%, which is close to the ResNet8 tests.

D. Accelerating Training

We compare the training times of our approach with the F8Nets GitHub repository[10]. We used a single GPU for 1 minute of training and *tqdm*[11] for the estimation of the whole epoch. To validate the estimation, we ran one epoch at batch size 512. The error is less than 15 seconds.

We use *nvidia-smi* for memory usage, which attributes 32768MiB of memory to the GPUs. The used version of PyTorch is 1.13.1+cu116.

Removing the second forward pass significantly reduces the training time, namely by about 30 – 35%

We tested full precision as well. For QATs seemingly optimal batch size of 512, full precision would require about 18 minutes and 18264MiB. This means double forward requires nearly twice the amount of time, while our approach only requires about 21% more time. The memory overhead for double forward is about 45%, while our method requires only about 10% more.

TABLE III
COMPARISON DOUBLE FORWARD AND OUR APPROACH, ROUNDED TO THE NEAREST HALF MINUTE. THE ESTIMATED TIME IS REASONABLY ACCURATE AS WE FULLY RAN ONE EPOCH AT THE BATCH SIZE OF 512 AND THE ERROR IS LESS THAN 15 SECONDS. THE FULL RUNTIMES ARE 34 MINUTES AND 42 SECONDS, AND 21 MINUTES AND 32 SECONDS, FOR DOUBLE FORWARD AND OUR APPROACH RESPECTIVELY.

Time & Memory usage	**Double Forward** F8Net Repository		**Our Approach**	
BS	***est. Time*** minutes	***Memory*** MiB	***est. Time*** minutes	***Memory*** MiB
256	35.5	13292	22.0	11782
512	34.5	26496	21.5	19992
640	35.0	30276	21.5	24354
768	35.0	31916	22.0	27950
896	OF	OF	23.0	32052
960	–	–	23.0	31322
1024	–	–	23.0	29364
1152	–	–	23.0	31940
1216	–	–	26.0	32208
1280	–	–	OF	OF

OF: Memory overflow.
– Not tested due to previous memory overflow.

Table III also displays some form of memory-optimization algorithm. The memory goes down as the batch size increases. It is unknown if PyTorch's or NVIDIA's optimization is strictly on the GPU or if allows it to store some tensors on the system memory. It is also quite surprising that this can not be seen in double forward. If system memory is used, it could explain why our approach could go further, as each layer requires its feature map only once, potentially making it easier to free up that memory on GPU and save a copy to the system memory for backpropagation.

E. Enforcing Rescalers

Table IV illustrates the possibility and effects of flexible Rescaler rules that allow enforcement of specific behavior. The network is a VGG-small channel-wise quantized network, and we enforce the Rescaler to take the following forms: layer-wise Rescaler, one Rescaler for the first layer and one for the rest, and one uniform Rescaler for the whole network. Unsurprisingly, the accuracies are lower than in the previous tests (Table IV and Fig. 4), as the restrictiveness of enforcing

[8]Accessed 12th of May 2023: https://github.com/snap-research/F8Net/tree/main

[9]Last Accessed 12th of May: https://pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html

[10]See footnote 8.

[11]https://pypi.org/project/tqdm/4.64.1/

TABLE IV
RESTRICTING THE RESCALER. THE TESTED NETWORK IS VGG-SMALL. THE COLUMN RESCALERS REPRESENTS THE RESULTING RIGHT SHIFT. HEAVIER RESTRICTIONS ON THE RESCALER MEAN HEAVIER RESTRICTIONS DURING TRAINING.

Rescricted Rescaler CIFAR10	**Train** ***Top1*** %	**Val.** ***Top1*** %	**Rescalers** ***Individual Layers*** right shift (2^{-n})
A8W8:			
Layerwise	100	92.7	7 , 11, 10, 10, 9 , 10, 11
First and Rest	100	92.7	7 , 10, 10, 10, 10, 10, 10
All	100	91.7	10, 10, 10, 10, 10, 10, 10
A4W4:			
Layerwise	100	92.6	7 , 7 , 7 , 6 , 6 , 6 , 9
First and Rest	100	92.3	7 , 7 , 7 , 7 , 7 , 7 , 7
All	99.9	92.1	7 , 7 , 7 , 7 , 7 , 7 , 7
Layer numbers:			1 , 2 , 3 , 4 , 5 , 6 , 7

Rescalers from the very start is noticeable. It is especially noticeable in 4-bit, as the last two tests result in identical Rescalers, but the accuracy differs. We conclude that if a restrictive Rescaler is desired, it should be introduced gradually and not from the very start.

VI. CONCLUSION

We propose a novel approach to QAT that allows using any linear quantization method for any linear Rescaler. We test it with unconstrained quantization methods and a power of 2 Rescaler. The tests are successful, as accuracy behaves as one would expect from regular QAT. This indicates that our approach can be treated similarly to regular QAT, with the benefit of additional HW restrictions. These restrictions affect accuracy, Table IV.

We see high potential for including existing optimizations, such as freezing the BN at convergence [6] and better gradient approximations than straight-through estimation. Furthermore, there are opportunities for developing novel optimizations regarding the Rescaler. These optimizations should address questions about when, how, and to which extent the Rescaler should constrain scaling factors. Table IV shows that easing into a Rescaler might be better than enforcing it from the start. Finally, more extensive tests are required to investigate the interaction between our approach and more general optimization strategies, such as a hyper-parameter search.

Through the well-defined behavior of our approach, approximations to remove the second forward pass through a convolution can be made without risking undefined behavior. Comparing our approach to full precision results in a reasonably consistent 20 – 22% training time overhead. The memory overhead ranges from 2 – 10%.

Compared to the state-of-the-art QAT method of double forward for BN fusion, our approach accelerates training by about 30 – 35% and significantly reduces the memory overhead, enabling bigger networks and or bigger batch sizes.

REFERENCES

[1] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," 2017.

[2] S. Jain, A. Gural, M. Wu, and C. Dick, "Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks," in *Proceedings of Machine Learning and Systems* (I. Dhillon, D. Papailiopoulos, and V. Sze, eds.), vol. 2, pp. 112–128, 2020.

[3] IRDS, "International roadmap for devices and systems - more Moore," tech. rep., IEEE, 2017.

[4] Q. Jin, J. Ren, R. Zhuang, S. Hanumante, Z. Li, Z. Chen, Y. Wang, K. Yang, and S. Tulyakov, "F8net: Fixed-point 8-bit only multiplication for network quantization," *arXiv preprint arXiv:2202.05239*, 2022.

[5] Z. Yao, Z. Dong, Z. Zheng, A. Gholami, J. Yu, E. Tan, L. Wang, Q. Huang, Y. Wang, M. Mahoney, *et al.*, "Hawq-v3: Dyadic neural network quantization," in *International Conference on Machine Learning*, pp. 11875–11886, PMLR, 2021.

[6] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018.

[7] Y. Li, M. Shen, J. Ma, Y. Ren, M. Zhao, Q. Zhang, R. Gong, F. Yu, and J. Yan, "Mqbench: Towards reproducible and deployable model quantization benchmark," 2022.

[8] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. Van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021.

[9] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "Pact: Parameterized clipping activation for quantized neural networks," 2018.

[10] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Tech. Rep. 0, University of Toronto, Toronto, Ontario, 2009.

[11] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[12] C. Sakr, S. Dai, R. Venkatesan, B. Zimmer, W. J. Dally, and B. Khailany, "Optimal clipping and magnitude-aware differentiation for improved quantization-aware training," 2022.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[14] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *European Conference on Computer Vision (ECCV)*, 2018.

[15] X. Sun, B. Wei, X. Ren, and S. Ma, "Label embedding network: Learning label representation for soft training of deep networks," 2017.

[16] M. Ayi and M. El-Sharkawy, "Rmnv2: Reduced mobilenet v2 for cifar10," in *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*, pp. 0287–0292, 2020.

[17] S. Yun and A. Wong, "Do all mobilenets quantize poorly? gaining insights into the effect of quantization on depthwise separable convolutional networks through the eyes of multi-scale distributional dynamics," 2021.

[18] R. Patel and A. Chaware, "Quantizing mobilenet models for classification problem," in *2021 8th International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 348–351, 2021.

[19] M. AskariHemmat, R. A. Hemmat, A. Hoffman, I. Lazarevich, E. Saboori, O. Mastropietro, S. Sah, Y. Savaria, and J.-P. David, "Qreg: On regularization effects of quantization," 2022.

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," 2016.