Data-Free Quantization Through Weight Equalization and Bias Correction

Markus Nagel* Mart van Baalen* Tijmen Blankevoort Max Welling

Qualcomm AI Research†
Qualcomm Technologies Netherlands B.V.
{markusn, mart, tijmen, mwelling}@qti.qualcomm.com

Abstract

We introduce a data-free quantization method for deep neural networks that does not require fine-tuning or hyper-parameter selection. It achieves near-original model performance on common computer vision architectures and tasks. 8-bit fixed-point quantization is essential for efficient inference on modern deep learning hardware. However, quantizing models to run in 8-bit is a non-trivial task, frequently leading to either significant performance reduction or engineering time spent on training a network to be amenable to quantization. Our approach relies on equalizing the weight ranges in the network by making use of a scale-equivariance property of activation functions. In addition the method corrects biases in the error that are introduced during quantization. This improves quantization accuracy performance, and can be applied to many common computer vision architectures with a straight forward API call. For common architectures, such as the MobileNet family, we achieve state-of-the-art quantized model performance. We further show that the method also extends to other computer vision architectures and tasks such as semantic segmentation and object detection.

1. Introduction

In recent years, deep learning based computer vision models have moved from research labs into the cloud and onto edge devices. As a result, power consumption and latency of deep learning inference have become an important concern. For this reason fixed-point quantization is often employed to make inference more efficient. By quantizing floating point values onto a regularly spaced grid, the original floating point values can be approximated by a set of integers, a scaling factor, and an optional zero point offset [16]. This allows for the use of faster and more power-efficient integer operations in matrix multiplication and convolution computations, at the expense of lower representational power. We refer the reader to [18] for details on commonly used, hardware-friendly quantization methods for deep learning models.

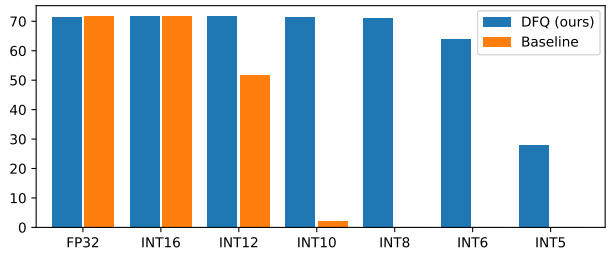

Figure 1. Fixed point inference for MobileNetV2 on ImageNet. The original model has significant drop in performance at 12-bit quantization whereas our model maintains close to FP32 performance even at 6-bit quantization.

Quantization of 32-bit full precision (FP32) models into 8-bit fixed point (INT8) introduces quantization noise on the weights and activations, which often leads to reduced model performance. This performance degradation ranges from very minor to catastrophic. To minimize the quantization noise, a wide range of different methods have been introduced in the literature (see Section 2). A major drawback of these quantization methods is their reliance on data and fine-tuning. As an example, consider real-world actors that manage hardware for quantized models, such as cloud-based deep learning inference providers or cellphone manufacturers. To provide a general use quantization service they would have to receive data from the customers to fine-tune the models, or rely on their customers to do the quantization. In either case, this can add a difficult step to the process. For such stakeholders it would be preferable if FP32 models could be converted directly to INT8, without needing the know-how, data or compute necessary for running traditional quantization methods. Even for model develop-

*Equal Contribution
†Qualcomm AI Research is an initiative of Qualcomm Technologies, Inc.

2380-7504/19/$31.00 ©2019 IEEE
DOI 10.1109/ICCV.2019.00141

ers that have the capability to quantize their own models, automation would save significant time.

In this paper, we introduce a quantization approach that does not require data, fine-tuning or hyperparameter tuning, resulting in accuracy improvement with a simple API call. Despite these restrictions we achieve near-original model performance when quantizing FP32 models to INT8. This is achieved by adapting the weight tensors of pre-trained models such that they are more amenable to quantization, and by correcting for the bias of the error that is introduced when quantizing models. We show significant improvements in quantization performance on a wide range of computer vision models previously thought to be difficult to quantize without fine-tuning.

Levels of quantization solutions In literature the practical application of proposed quantization methods is rarely discussed. To distinguish between the differences in applicability of quantization methods, we introduce four levels of quantization solutions, in decreasing order of practical applicability. Our hope is that this will enable other authors to explore solutions for each level, and makes the comparison between methods more fair. The axes for comparison are whether or not a method requires data, whether or not a method requires error backpropagation on the quantized model, and whether or not a method is generally applicable for any architecture or requires significant model reworking. We use the following definitions throughout the paper:

Level 1 No data and no backpropagation required. Method works for any model. As simple as an API call that only looks at the model definition and weights.

Level 2 Requires data but no backpropagation. Works for any model. The data is used e.g. to re-calibrate batch normalization statistics [27] or to compute layer-wise loss functions to improve quantization performance. However, no fine-tuning pipeline is required.

Level 3 Requires data and backpropagation. Works for any model. Models can be quantized but need fine-tuning to reach acceptable performance. Often requires hyperparameter tuning for optimal performance. These methods require a full training pipeline (e.g. [16, 35]).

Level 4 Requires data and backpropagation. Only works for specific models. In this case, the network architecture needs non-trivial reworking, and/or the architecture needs to be trained from scratch with quantization in mind (e.g. [4, 31, 21]). Takes significant extra training-time and hyperparameter tuning to work.

2. Background and related work

There are several works that describe quantization and improving networks for lower bit inference and deployment [9, 10, 16, 35]. These methods all rely on fine-tuning, making them level 3 methods, whereas data-free quantization improves performance similarly without that requirement. Our method is complementary to these and can be applied as a pre-processing before quantization aware fine-tuning.

In a whitepaper, Krishnamoorthi [18], introduces a level 1 'per-channel' quantization scheme, in which the weights of a convolutional weight tensor are quantized per output channel. A major drawback of this method is that it is not supported on all hardware, and that it creates unnecessary overhead in the computation due to the necessity of scale and offset values for each channel individually. We show that our method improves on per-channel quantization, while keeping a single set of scale and offset values for the whole weight tensor instead.

Other methods to improve quantization need architecture changes or training with quantization in mind from the start [1, 21, 31, 33, 36]. These methods are even more involved than doing quantization and fine-tuning. They also incur a relatively large overhead during training because of sampling and noisy optimization, and introduce extra hyperparameters to optimize. This makes them level 4 methods.

Methods that binarize [5, 15, 27, 28] or ternarize [19] networks result in models with great inference efficiency as expensive multiplications and additions are replaced by bit-shift operations. However, quantizing models to binary often leads to strong performance degradation. Generally they need to be trained from scratch, making them level 4 methods.

Other approaches use low-bit floating point operations instead of integer operations, or other custom quantization implementations [8, 17, 24, 35]. We do not consider such approaches as the hardware implementation is less efficient.

In concurrent work, Meller et al. [22] also exploits the scale equivariance of the ReLU function to rescale weight channels and notice the biased error introduced by weight quantization [7], leading to a method that resembles our data-free quantization approach. Stock et al. [32] also use the scale equivariance property of the ReLU function, but use it for network optimization instead.

3. Motivation

While many trained FP32 models can be quantized to INT8 without much loss in performance, some models exhibit a significant drop in performance after quantization ([18, 31]). For example, when quantizing a trained MobileNetV2 [30] model, Krishnamoorthi [18] reports a drop in top-1 accuracy from 70.9% to 0.1% on the ImageNet [29] validation set. The author restores near original model performance by either applying per-channel quantization, fine-tuning or both.

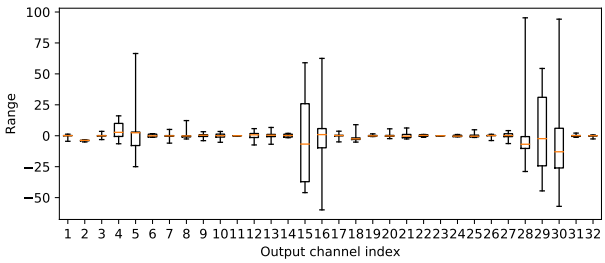

Figure 2. Per (output) channel weight ranges of the first depthwise-separable layer in MobileNetV2. In the boxplot the min and max value, the 2nd and 3rd quartile and the median are plotted for each channel. This layer exhibits strong differences between channel weight ranges.

3.1. Weight tensor channel ranges

The fact that per-channel quantization yields much better performance on MobileNetV2 than per-tensor quantization suggests that, in some layers, the weight distributions differ so strongly between output channels that the same set of quantization parameters cannot be used to quantize the full weight tensor effectively. For example, in the case where one channel has weights in the range $[-128, 128]$ and another channel has weights in the range $(-0.5, 0.5)$, the weights in the latter channel will all be quantized to 0 when quantizing to 8-bits.

Figure 2 shows that large differences in output channel weight ranges do indeed occur in a (trained) MobileNetV2 model. This figure shows the weight distribution of the output channel weights of the depthwise-separable layer in the model's first inverted residual block. Due to the strong differences between channel weight ranges that this layer exhibits, it cannot be quantized with reasonable accuracy for each channel. Several layers in the network suffer from this problem, making the overall model difficult to quantize.

We conjecture that performance of trained models after quantization can be improved by adjusting the weights for each output channel such that their ranges are more similar. We provide a level 1 method to achieve this without changing the FP32 model output in section 4.1.

3.2. Biased quantization error

A common assumption in literature (e.g. [2]) is that quantization error is unbiased and thus cancels out in a layer's output, ensuring that the mean of a layer's output does not change as a result of quantization. However, as we will show in this section, the quantization error on the weights might introduce biased error on the corresponding outputs. This shifts the input distribution of the next layer, which may cause unpredictable effects.

The biased error in a quantized layer's output unit j can

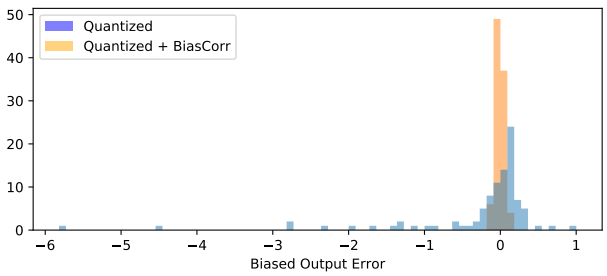

Figure 3. Per-channel biased output error introduced by weight quantization of the second depthwise-separable layer in MobileNetV2, before (blue) and after (orange) bias correction.

be computed empirically using N input data points as:

$$\mathbb{E}[\tilde{\mathbf{y}}_j - \mathbf{y}_j] \approx \frac{1}{N} \sum_n (\widetilde{\mathbf{W}}\mathbf{x}_n)_j - (\mathbf{W}\mathbf{x}_n)_j \tag{1}$$

where $\mathbf{y}_j$ and $\tilde{\mathbf{y}}_j$ are the original outputs and the outputs generated using the quantized weight matrix, respectively.

Figure 3 shows the biased error per channel of a depthwise-separable convolution layer in a trained MobileNetV2 model. From this plot it is clear that for many channels in the layer's output, the error introduced by weight quantization is biased, and influences the output statistics. Depthwise-separable layers are especially susceptible to this biased error effect as each output channel has only 9 corresponding weights.

Such a biased error on the outputs can be introduced in many settings, e.g. when weights or activations are clipped [23], or in non-quantization approaches, such as weight tensor factorization or channel pruning [13, 34].

In section 4.2 we introduce a method to correct for this bias. Furthermore, we show that a model's batch normalization parameters can be used to compute the expected biased error on the output, yielding a level 1 method to fix the biased error introduced by quantization.

4. Method

Our proposed data-free quantization method (DFQ) consists of three steps, on top of the normal quantization. The overall flow of the algorithm is shown in Figure 4.

4.1. Cross-layer range equalization

Positive scaling equivariance We observe that for a ReLU [25] activation function $f(\cdot)$ the following scaling equivariance property holds:

$$f(sx) = sf(x) \tag{2}$$

for any non-negative real number s. This follows from the definition of the ReLU:

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0. \end{cases} \tag{3}$$

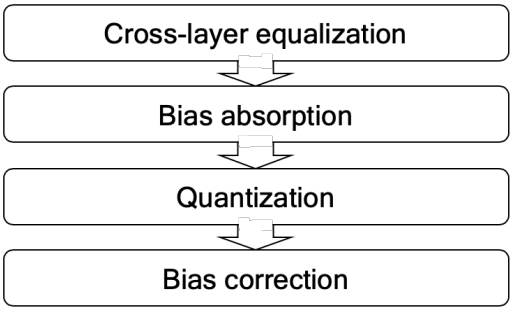

Figure 4. Flow diagram of the proposed DFQ algorithm.

This equivariance also holds for the PreLU [11] activation function. More generally, the positive scaling equivariance can be relaxed to $f(sx) = s\hat{f}(x)$ for any piece-wise linear activation functions:

$$f(x) = \begin{cases} a_1 x + b_1 & \text{if } x \leq c_1 \\ a_2 x + b_2 & \text{if } c_1 < x \leq c_2 \\ \vdots & \\ a_n x + b_n & \text{if } c_{n-1} < x \end{cases} \tag{4}$$

where $\hat{f}(\cdot)$ is parameterized as $\hat{a}_i = a_i$, $\hat{b}_i = b_i/s$ and $\hat{c}_i = c_i/s$. Note that contrary to equivariance defined in eq. 2 we now also change the function $f(\cdot)$ into $\hat{f}(\cdot)$.

4.1.1. Scaling equivariance in neural networks

The positive scaling equivariance can be exploited in consecutive layers in neural networks. Given two layers, $\mathbf{h} = f(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$ and $\mathbf{y} = f(\mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)})$, through scaling invariance we have that:

$$\mathbf{y} = f(\mathbf{W}^{(2)} f(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) \tag{5}$$

$$= f(\mathbf{W}^{(2)}\mathbf{S}\hat{f}(\mathbf{S^{-1}}\mathbf{W}^{(1)}\mathbf{x} + \mathbf{S^{-1}}\mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) \tag{6}$$

$$= f(\widehat{\mathbf{W}}^{(2)}\hat{f}(\widehat{\mathbf{W}}^{(1)}\mathbf{x} + \widehat{\mathbf{b}}^{(1)}) + \mathbf{b}^{(2)}) \tag{7}$$

where $\mathbf{S} = diag(\mathbf{s})$ is a diagonal matrix with value S_{ii} denoting the scaling factor s_i for neuron i. This allows us to reparameterize our model with $\widehat{\mathbf{W}}^{(2)} = \mathbf{W}^{(2)}\mathbf{S}$, $\widehat{\mathbf{W}}^{(1)} = \mathbf{S^{-1}}\mathbf{W}^{(1)}$ and $\widehat{\mathbf{b}}^{(1)} = \mathbf{S^{-1}}\mathbf{b}^{(1)}$. In case of CNNs the scaling will be per channel and broadcast accordingly over the spatial dimensions. The rescaling procedure is illustrated in Figure 5.

4.1.2. Equalizing ranges over multiple layers

We can exploit the rescaling and reparameterization of the model to make the model more robust to quantization. Ideally the ranges of each channel i are equal to the total range of the weight tensor, meaning we use the best possible representative power per channel. We define the precision of a channel as:

$$p_i^{(1)} = \frac{r_i^{(1)}}{R^{(1)}} \tag{8}$$

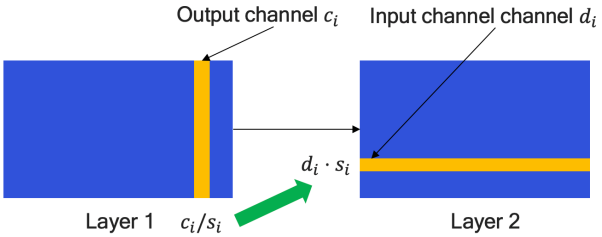

Figure 5. Illustration of the rescaling for a single channel. If scaling factor s_i scales c_i in layer 1; we can instead factor it out and multiply d_i in layer 2.

where $r_i^{(1)}$ is the quantization range of channel i in $\widehat{\mathbf{W}}^{(1)}$ and $R^{(1)}$ is the total range of $\widehat{\mathbf{W}}^{(1)}$. We want to find $\mathbf{S}$ such that the total precision per channel is maximized:

$$\max_{\mathbf{S}} \sum_i p_i^{(1)} p_i^{(2)} \tag{9}$$

In the case of symmetric quantization we have $r_i^{(1)} = 2 \cdot \max_j |\widehat{\mathbf{W}}_{\mathbf{ij}}^{(\mathbf{1})}|$ and $R^{(1)} = 2 \cdot \max_{ij} |\widehat{\mathbf{W}}_{\mathbf{ij}}^{(\mathbf{1})}|$. Solving eq. 9 (see appendix A) leads to the necessary condition:

$$\arg\max_j \frac{1}{s_j} r_j^{(1)} = \arg\max_k s_k r_k^{(2)} \tag{10}$$

meaning the limiting channel defining the quantization range is given by $\arg\max_i r_i^{(1)} r_i^{(2)}$. We can satisfy this condition by setting $\mathbf{S}$ such that:

$$s_i = \frac{1}{r_i^{(1)}} \sqrt{r_i^{(1)} r_i^{(2)}} \tag{11}$$

which results in $\forall i : r_i^{(1)} = r_i^{(2)}$. Thus the channel's ranges between both tensors are matched as closely as possible.

When equalizing multiple layers at the same time, we iterate this process for pairs of layers that are connected to each other without input or output splits in between, until convergence.

4.1.3. Absorbing high biases

In case $s_i < 1$ the equalization procedure increases bias $b_i^{(1)}$. This could in turn increase the range of the activation quantization. In order to avoid big differences between per-channel ranges in the activations we introduce a procedure that absorbs high biases into the subsequent layer.

For a layer with ReLU function r, there is a non-negative vector $\mathbf{c}$ such that $r(\mathbf{W}\mathbf{x} + \mathbf{b} - \mathbf{c}) = r(\mathbf{W}\mathbf{x} + \mathbf{b}) - \mathbf{c}$. The trivial solution $\mathbf{c} = \mathbf{0}$ holds for all $\mathbf{x}$. However, depending on the distribution of $\mathbf{x}$ and the values of $\mathbf{W}$ and $\mathbf{b}$, there can be a some values $\mathbf{c}_i > \mathbf{0}$ for which this equality holds for (almost) all $\mathbf{x}$. Following the previous two layer example,

these $\mathbf{c}_i$ can be absorbed from layer 1 into layer 2 as:

$$\mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)} \tag{12}$$
$$= \mathbf{W}^{(2)}(r(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{c} - \mathbf{c}) + \mathbf{b}^{(2)} \tag{13}$$
$$= \mathbf{W}^{(2)}(r(\mathbf{W}^{(1)}\mathbf{x} + \hat{\mathbf{b}}^{(1)}) + \mathbf{c}) + \mathbf{b}^{(2)} \tag{14}$$
$$= \mathbf{W}^{(2)}\hat{\mathbf{h}} + \hat{\mathbf{b}}^{(2)} \tag{15}$$

where $\hat{\mathbf{b}}^{(2)} = \mathbf{W}^{(2)}\mathbf{c} + \mathbf{b}^{(2)}$, $\hat{\mathbf{h}} = \mathbf{h} - \mathbf{c}$, and $\hat{\mathbf{b}}^{(1)} = \mathbf{b}^{(1)} - \mathbf{c}$.

To find $\mathbf{c}$ without violating our data-free assumption we assume that the pre-bias activations are distributed normally with the batch normalization shift and scale parameters $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ as its mean and standard deviation. We set $\mathbf{c} = \max(\mathbf{0}, \boldsymbol{\beta} - 3\boldsymbol{\gamma})$. If $\mathbf{c} > 0$, the equality introduced above will hold for the 99.865% of values of $\mathbf{x}$ (those greater than $> \mathbf{c}$) under the Gaussian assumption. As we will show in section 5.1.1, this approximation does not harm the full precision performance significantly but helps for activation quantization. Note that, in case data is available, the pre-bias distribution of $\mathbf{x}$ can be found empirically and used to set $\mathbf{c}$.

4.2. Quantization bias correction

As shown empirically in the motivation, quantization can introduce a biased error in the activations. In this section we show how to correct for the bias in the error on the layer's output, and how we can use the network's batch normalization parameters to compute this bias without using data.

For a fully connected layer with weight tensor $\mathbf{W}$, quantized weights $\widetilde{\mathbf{W}}$, and input activations $\mathbf{x}$, we have $\widetilde{\mathbf{y}} = \widetilde{\mathbf{W}}\mathbf{x}$ and therefore $\widetilde{\mathbf{y}} = \mathbf{y} + \boldsymbol{\epsilon}\mathbf{x}$, where we define the quantization error $\boldsymbol{\epsilon} = \widetilde{\mathbf{W}} - \mathbf{W}$, $\mathbf{y}$ as the layer pre-activations of the FP32 model, and $\widetilde{\mathbf{y}}$ that layer with quantization error added.

If the expectation of the error for output i, $\mathbb{E}[\boldsymbol{\epsilon}\mathbf{x}]_i \neq 0$, then the mean of the output i will change. This shift in distribution may lead to detrimental behavior in the following layers. We can correct for this change by seeing that:

$$\mathbb{E}[\mathbf{y}] = \mathbb{E}[\mathbf{y}] + \mathbb{E}[\boldsymbol{\epsilon}\mathbf{x}] - \mathbb{E}[\boldsymbol{\epsilon}\mathbf{x}] \tag{16}$$
$$= \mathbb{E}[\widetilde{\mathbf{y}}] - \mathbb{E}[\boldsymbol{\epsilon}\mathbf{x}]. \tag{17}$$

Thus, subtracting the expected error on the output $\mathbb{E}[\boldsymbol{\epsilon}\mathbf{x}] = \boldsymbol{\epsilon}\mathbb{E}[\mathbf{x}]$ from the biased output $\widetilde{\mathbf{y}}$ ensures that the mean for each output unit is preserved.

For implementation, the expected error can be subtracted from the layer's bias parameter, since the expected error vector has the same shape as the layer's output. This method easily extends to convolutional layers as described in Appendix B.

4.2.1. Computing the expected input

To compute the expected error of the output of a layer, the expected input to the layer $\mathbb{E}[\mathbf{x}]$ is required. If a model does not use batch normalization, or there are no data-usage restrictions, $\mathbb{E}[\boldsymbol{\epsilon}\mathbf{x}]$ can be computed by comparing the activations before and after quantization. Appendix D explains this procedure in more detail.

Clipped normal distribution When the network includes batch normalization before a layer, we can use it to calculate $\mathbb{E}[\mathbf{x}]$ for that layer without using data. We assume the pre-activation outputs of a layer are normally distributed, that batch normalization is applied before the activation function, and that the activation function is some form of the class of clipped linear activation functions (e.g. ReLU, ReLU6), which clips its input range to the range $[a, b]$ where $a < b$, and b can be ∞.

Due to the centralization and normalization applied by batch normalization, the mean and standard deviation of the pre-activations are known: these are the batch normalization scale and shift parameters (henceforth referred to as $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ respectively).

To compute $\mathbb{E}[\mathbf{x}]$ from the previous layer's batch normalization parameters, the mean and variance need to be adjusted to account for the activation function that follows the batch normalization layer. For this purpose we introduce the clipped normal distribution. A clipped-normally distributed random variable X is a normally distributed random variable with mean μ and variance σ^2, whose values are clipped to the range $[a, b]$ The mean and variance of the clipped normal distribution can be computed in closed form from μ, σ, a and b. We present the mean of the clipped normal distribution for the ReLU activation function, i.e. $a = 0$ and $b = \infty$ in this section, and refer the reader to Appendix C for the closed form solution for the general clipped normal distribution.

The expected value for channel c in $\mathbf{x}$, $\mathbb{E}[\mathbf{x}_c]$, which is the output of a layer with batch normalization parameters $\boldsymbol{\beta}_c$ and $\boldsymbol{\gamma}_c$, followed by a ReLU activation function is:

$$\mathbb{E}[\mathbf{x}_c] = \mathbb{E}\left[\text{ReLU}\left(\mathbf{x}_c^{pre}\right)\right] \tag{18}$$
$$= \boldsymbol{\gamma}_c \mathcal{N}\left(\frac{-\boldsymbol{\beta}_c}{\boldsymbol{\gamma}_c}\right) + \boldsymbol{\beta}_c\left[1 - \Phi\left(\frac{-\boldsymbol{\beta}_c}{\boldsymbol{\gamma}_c}\right)\right] \tag{19}$$

where $\mathbf{x}_c^{pre}$ is the pre-activation output for channel c, which is assumed to be normally distributed with mean $\boldsymbol{\beta}_c$ and variance $\boldsymbol{\gamma}_c^2$, $\Phi(\cdot)$ is the normal CDF, and the notation $\mathcal{N}(x)$ is used to denote the normal $\mathcal{N}(x|0, 1)$ PDF.

5. Experiments

In this section we present two sets of experiments to validate the performance of data-free quantization (DFQ). We first show in section 5.1 the effect of the different aspects of DFQ and how they solve the problems observed earlier. Then we show in section 5.2 how DFQ generalizes to other

models and tasks, and sets a new state-of-the-art for level 1 quantization.

To allow comparison to previously published results, we use both weights and activations are quantized using 8-bit asymmetric, per-tensor quantization in all experiments. Batch normalization is folded in the adjacent layer before quantization. Weight quantization ranges are the min and max of the weight tensor. Activation quantization ranges are set without data, by using the learned batch normalization shift and scale parameter vectors $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ as follows: We compute the activation range for channel i as $\boldsymbol{\beta}_i \pm n \cdot \boldsymbol{\gamma}_i$ (with $n = 6$), with the minimum clipped to 0 in case of ReLU activation. We observed a wide range of n can be used without significant performance difference. All experiments are done in Pytorch [26]. In appendix E we show additional experiments using short-term fine-tuning, symmetric quantization and per-channel quantization.

5.1. Ablation study

In this section we investigate the effect of our methods on a pre-trained MobileNetV2 [30] model[1]. We validate the performance of the model on the ImageNet [29] validation set. We first investigate the effects of different parts of our approach through a set of ablation studies.

5.1.1. Cross-layer equalization

In this section we investigate the effects of cross-layer equalization and high-bias folding. We compare these methods to two baselines: the original quantized model and the less hardware friendly per-channel quantization scheme.

The models considered in this section employ residual connections [12]. For these networks we apply cross-layer equalization only to the layers within each residual block. MobileNetV2 uses ReLU6 activation functions, which clips activation ranges to $[0, 6]$. To avoid ReLU6 requiring a different cut off per channel after applying the equalization procedure, we replace ReLU6 with regular ReLU.

The results of the equalization experiments are shown in Table 1. Similar to [18], we observe that the model performance is close to random when quantizing the original model to INT8. Further we note that replacing ReLU6 by ReLU does not significantly degrade the model performance. Applying equalization brings us to within 2% of FP32 performance, close to the performance of per-channel quantization. We note that absorbing high biases results in a small drop in FP32 performance, but it boosts quantized performance by 1% due to more precise activation quantization. Combining both methods improves performance over per-channel quantization, indicating the more efficient per-tensor quantization could be used instead.

[1]We use the Pytorch implementation of MobileNetV2 provided by `https://github.com/tonylins/pytorch-mobilenet-v2`.

Model	FP32	INT8
Original model	71.72%	0.12%
Replace ReLU6	71.70%	0.11%
+ equalization	71.70%	69.91%
+ absorbing bias	71.57%	**70.92%**
Per channel quantization	71.72%	70.65%

Table 1. Top1 ImageNet validation results for MobileNetV2, evaluated at full precision and 8-bit integer quantized. Per-channel quantization is our own implementation of [16] applied post-training.

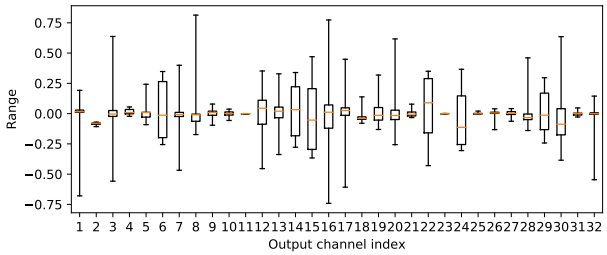

Figure 6. Per (output) channel weight ranges of the first depthwise-separable layer in MobileNetV2 after equalization. In the boxplot the min and max value, the 2nd and 3rd quartile and the median are plotted for each channel. Most channels in this layer are now within similar ranges.

To illustrate the effect of cross-layer equalization, we show the weight distributions per output channel of the depthwise-separable layer in the models first inverted residual block after applying the equalization in Figure 6. We observe that most channels ranges are now similar and that the strong outliers from Figure 2 have been equalized. Note, there are still several channels which have all weight values close to zero. These channels convey little information and can be pruned from the network with hardly any loss in accuracy.

5.1.2. Bias correction

In this section we present results on bias correction for a quantized MobileNetV2 model. We furthermore present results of bias correction in combination with a naive weight-clipping baseline, and combined with the cross-layer equalization approach.

The weight-clipping baseline serves two functions: 1) as a naive baseline to the cross-layer equalization approach, and 2) to show that bias correction can be employed in any setting where biased noise is introduced. Weight clipping solves the problem of large differences in ranges between channels by clipping large ranges to smaller ranges, but it introduces a strongly biased error. Weight clipping is applied by first folding the batch normalization parameters into a layer's weights, and then clipping all values to

Model	FP32	INT8
Original Model	71.72%	0.12%
Bias Corr	71.72%	**52.02**%
Clip @ 15	67.06%	2.55%
+ Bias Corr	**71.15**%	**70.43**%
Rescaling + Bias Absorption	71.57%	70.92%
+ Bias Corr	71.57%	**71.19**%

Table 2. Top1 ImageNet validation results for MobileNetV2, evaluated at full precision and 8-bit integer quantized. Bold results show the best result for each column in each cell.

a certain range, in this case $[-15, 15]$. We tried multiple symmetric ranges, all provided similar results. For residual connections we calculate $\mathbb{E}[\mathbf{x}]$ and $\text{Var}[\mathbf{x}]$ based on the sum and variance of all input expectations, taking the input to be zero mean and unit variance.

To illustrate the effect of bias correction, Figure 3 shows the per output channel biased error introduced by weight quantization. The per-channel biases are obtained as described in eq. 1. This figure shows that applying bias correction reduces the bias in the error on the output of a layer to very close to 0 for most output channels.

Results for the experiments described above for MobileNet V2 on the ImageNet validation set are shown in Table 2. Applying bias correction improves quantized model performance, indicating that a part of the problem of quantizing this model lies in the biased error that is introduced. However, bias correction on its own does not achieve near-floating point performance. The reason for this is most likely that the problem described in 3.1 is more severe for this model. The experiments on weight-clipping show that bias correction can mitigate performance degradation due to biased error in non-quantized models as well as quantized models. Clipping without correction in the FP32 model introduces a 4.66% loss in accuracy; bias correction reduces that loss to a mere 0.57%. Furthermore, it shows that weight clipping combined with bias correction is a fairly strong baseline for quantizing MobileNet V2. Lastly, we show that bias correction improves results when combined with the cross-layer equalization and bias folding procedures. The combination of all methods is our data-free quantization (DFQ) method. The full DFQ approach achieves near-floating point performance with a reduction of 0.53% top 1 accuracy relative to the FP32 baseline.

5.2. Comparison to other methods and models

In this section we show how DFQ generalizes to other popular computer vision tasks, namely semantic segmentation and object detection, and other model architectures such as MobileNetV1 [14] and Resnet18 [12]. Afterwards we compare DFQ to methods in the literature, including more complex level 3 and 4 approaches. This set of models was chosen as they are efficient and likely to be used in mobile applications where 8-bit quantization is frequently used for power efficiency.

Model	FP32	INT8
Original model	72.94	41.40
DFQ (ours)	72.45	**72.33**
Per-channel quantization	72.94	71.44

Table 3. DeeplabV3+ (MobileNetV2 backend) on Pascal VOC segmentation challenge. Mean intersection over union (mIOU) evaluated at full precision and 8-bit integer quantized. Per-channel quantization is our own implementation of [16] applied post-training.

Model	FP32	INT8
Original model	68.47	10.63
DFQ (ours)	68.56	**67.91**
Per-channel quantization	68.47	67.52

Table 4. MobileNetV2 SSD-lite on Pascal VOC object detection challange. Mean average precision (mAP) evaluated at full precision and 8-bit integer quantized. Per-channel quantization is our own implementation of [16] applied post-training.

5.2.1. Other tasks

Semantic segmentation To demonstrate the generalization of our method to semantic segmentation we apply DFQ for DeeplabV3+ with a MobileNetV2 backend [3, 30], performance is evaluated on the Pascal VOC segmentation challenge [6]. For our experiments we use the publicly available Pytorch implementation[2].

We show the results of this experiment in Table 3. As observed earlier for classification we notice a significant drop in performance when quantizing the original model which makes it almost unusable in practice. Applying DFQ recovers almost all performance degradation and achieves less than 1% drop in mIOU compared to the full precision model. DFQ also outperforms the less hardware friendly per-channel quantization. To the best of our knowledge we are the first to publish quantization results on DeeplabV3+ as well as for semantic segmentation.

Object detection To demonstrate the applicability of our method to object detection we apply DFQ for MobileNetV2 SSDLite [30, 20], evaluated on the Pascal VOC object detection challenge [6]. In our experiments we use the publicly available Pytorch implementation of SSD[3].

[2] https://github.com/jfzhang95/pytorch-deeplab-xception

[3] https://github.com/qfgaohao/pytorch-ssd

	∼D	∼BP	∼AC	MobileNetV2		MobileNetV1		ResNet18		
				FP32	INT8	FP32	INT8	FP32	INT8	INT6
DFQ (ours)	✓	✓	✓	71.7%	**71.2%**	70.8%	**70.5%**	69.7%	**69.7%**	66.3%
Per-layer [18]	✓	✓	✓	71.9%	0.1%	70.9%	0.1%	69.7%	69.2%*	63.8%*
Per-channel [18]	✓	✓	✓	71.9%	69.7%	70.9%	70.3%	69.7%	69.6%*	**67.5%***
QT [16] ˆ	✗	✗	✓	71.9%	70.9%	70.9%	70.0%	-	**70.3%**†	67.3%†
SR+DR†	✗	✗	✓	-	-	-	**71.3%**	-	68.2%	59.3%
QMN [31]	✗	✗	✗	-	-	70.8%	68.0%	-	-	-
RQ [21]	✗	✗	✗	-	-	-	70.4%	-	69.9%	**68.6%**

Table 5. Top1 ImageNet validation results for different models and quantization approaches. The top half compares level 1 approaches (∼D: data free, ∼BP: backpropagation-free, ∼AC: Architecture change free) whereas in the second half we also compare to higher level approaches in literature. Results with * indicates our own implementation since results are not provided, ˆ results provided by [18] and † results from table 2 in [21].

The results are listed in Table 4. Similar to semantic segmentation we observe a significant drop in performance when quantizing the SSDLite model. Applying DFQ recovers almost all performance drop and achieves less than 1% drop in mAP compared to the full precision model, again outperforming per-channel quantization.

5.2.2. Comparison to other approaches

In this section we compare DFQ to other approaches in literature. We compare our results to two other level 1 approaches, direct per-layer quantization as well as per-channel quantization [18]. In addition we also compare to multiple higher level approaches, namely quantization aware training [16] as well as stochastic rounding and dynamic ranges [9, 10], which are both level 3 approaches. We also compare to two level 4 approaches based on relaxed quantization [21], which involve training a model from scratch and to quantization friendly separable convolutions [31] that require a rework of the original MobileNet architecture. The results are summarized in Table 5.

For both MobileNetV1 and MobileNetV2 per-layer quantization results in an unusable model whereas DFQ stays close to full precision performance. DFQ also outperforms per-channel quantization as well as most level 3 and 4 approaches which require significant fine-tuning, training or even architecture changes.

On Resnet18 we maintain full precision performance for 8-bit fixed point quantization using DFQ. Some higher level approaches [16, 21] report slightly higher results than our baseline model, likely due to a better training procedure than used in the standard Pytorch Resnet18 model. Since 8-bit quantization is lossless we also compare 6-bit results. DFQ clearly outperforms traditional per-layer quantization but stays slightly below per-channel quantization and higher level approaches such as QT and RQ [16, 21].

Overall DFQ sets a new state-of-the-art for 8-bit fixed point quantization on several models and computer vision tasks. It is especially strong for mobile friendly architectures such as MobileNetV1 and MobileNetV2 which were previously hard to quantize. Even though DFQ is an easy to use level 1 approach, we generally show competitive performance when comparing to more complex level 2-4 approaches.

6. Conclusion

In this work, we introduced DFQ, a data-free quantization method that significantly helps quantized model performance without the need for data, fine-tuning or hyper-parameter optimization. The method can be applied to many common computer vision architectures with a straight-forward API call. This is crucial for many practical applications where engineers want to deploy deep learning models trained in FP32 to INT8 hardware without much effort. Results are presented for common computer vision tasks like image classification, semantic segmentation and object detection. We show that our method compares favorably to per-channel quantization [18], meaning that instead the more efficient per-tensor quantization can be employed in practice. DFQ achieves near original model accuracy for almost every model we tested, and even competes with more complicated training based methods.

Further we introduced a set of quantization levels to facilitate the discussion on the applicability of quantization methods. There is a difference in how easy a method is to use for generating a quantized model, which is a significant part of the impact potential of a quantization method in real world applications. We hope that the quantization levels and methods introduced in this paper will contribute to both future research and practical deployment of quantized deep learning models.

Acknowledgments

We would like to thank Christos Louizos, Harris Teague, Jakub Tomczak, Mihir Jain and Pim de Haan for their helpful discussions and valuable feedback.

References

[1] Jan Achterhold, Jan Mathias Koehler, Anke Schmeink, and Tim Genewein. Variational network quantization. In *International Conference on Learning Representations (ICLR)*, 2018.

[2] Raziel Alvarez, Rohit Prabhavalkar, and Anton Bakhtin. On the efficient representation and execution of deep acoustic models. In *The Annual Conference of the International Speech Communication Association (Interspeech)*, 2016.

[3] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[4] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. PACT: parameterized clipping activation for quantized neural networks. *arXiv preprint arxiv:805.06085*, 2018.

[5] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 3123–3131, Cambridge, MA, USA, 2015. MIT Press.

[6] Mark Everingham, S. M. Ali Eslami, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, 1 2015.

[7] Alexander Finkelstein, Uri Almog, and Mark Grobman. Fighting quantization bias with bias. *arXiv preprint arxiv:1906.03193*, 2019.

[8] Denis A. Gudovskiy and Luca Rigazio. Shiftcnn: Generalized low-precision architecture for inference of convolutional neural networks. *arXiv preprint arxiv:1706.02393*, 2017.

[9] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 1737–1746, 2015.

[10] Philipp Gysel, Jon J. Pimentel, Mohammad Motamedi, and Soheil Ghiasi. Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks. *IEEE Trans. Neural Netw. Learning Syst.*, 29(11):5784–5789, 2018.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034, 2015.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778, 2016.

[13] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 1398–1406, 2017.

[14] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[15] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18:187:1–187:30, 2017.

[16] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[17] Urs Köster, Tristan Webb, Xin Wang, Marcel Nassar, Arjun K. Bansal, William Constable, Oguz Elibol, Stewart Hall, Luke Hornof, Amir Khosrowshahi, Carey Kloss, Ruby J. Pai, and Naveen Rao. Flexpoint: An adaptive numerical format for efficient training of deep neural networks. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 1740–1750, 2017.

[18] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, Jun 2018.

[19] Fengfu Li and Bin Liu. Ternary weight networks. *arXiv preprint arxiv:1605.04711*, 2016.

[20] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part I*, pages 21–37, 2016.

[21] Christos Louizos, Matthias Reisser, Tijmen Blankevoort, Efstratios Gavves, and Max Welling. Relaxed quantization for discretized neural networks. In *International Conference on Learning Representations (ICLR)*, 2019.

[22] Eldad Meller, Alexander Finkelstein, Uri Almog, and Mark Grobman. Same, same but different: Recovering neural network quantization error through weight factorization. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 4486–4495, 2019.

[23] Asit K. Mishra, Jeffrey J. Cook, Eriko Nurvitadhi, and Debbie Marr. WRPN: training and inference using wide reduced-precision networks. *arXiv preprint arxiv 1704.03079*, 2017.

[24] Daisuke Miyashita, Edward H. Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *arXiv preprint arxiv:1603.01025*, 2016.

[25] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the*

27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel, pages 807–814, 2010.

[26] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[27] Jorn W. T. Peters and Max Welling. Probabilistic binary neural networks. *arXiv preprint arxiv:1809.03368*, 2018.

[28] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, pages 525–542, 2016.

[29] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[30] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[31] Tao Sheng, Chen Feng, Shaojie Zhuo, Xiaopeng Zhang, Liang Shen, and Mickey Aleksic. A quantization-friendly separable convolution for mobilenets. In *1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, 2018.

[32] Pierre Stock, Benjamin Graham, Rmi Gribonval, and Herv Jgou. Equi-normalization of neural networks. In *International Conference on Learning Representations*, 2019.

[33] Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. In *International Conference on Learning Representations (ICLR)*, 2017.

[34] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 38(10):1943–1955, 2016.

[35] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arxiv:1702.03044*, abs/1702.03044, 2017.

[36] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.