Weighted Quantization-Regularization in DNNs for Weight Memory Minimization Toward HW Implementation

Matthias Wess, Sai Manoj Pudukotai Dinakarrao, *Member, IEEE*, and Axel Jantsch, *Member, IEEE*

***Abstract*—Deployment of deep neural networks on hardware platforms is often constrained by limited on-chip memory and computational power. The proposed weight quantization offers the possibility of optimizing weight memory alongside transforming the weights to hardware friendly data types. We apply dynamic fixed point (DFP) and power-of-two (Po2) quantization in conjunction with *layer-wise precision scaling* to minimize the weight memory. To alleviate accuracy degradation due to precision scaling, we employ quantization-aware fine-tuning. For fine-tuning, quantization-regularization (QR) and weighted QR are introduced to force the trained quantization by adding the distance of the weights to the desired quantization levels as a regularization term to the loss-function. While DFP quantization performs better when allowing different bit-widths for each layer, Po2 quantization in combination with retraining allows higher compression rates for equal bit-width quantization. The techniques are verified on an all-convolutional network. With accuracy degradation of 0.10% points, for DFP with layer-wise precision scaling we achieve compression ratios of 7.34 for CIFAR-10, 4.7 for CIFAR-100, and 9.33 for SVHN dataset.**

***Index Terms*—Convolutional neural networks, memory minimization, quantization, regularization.**

I. Introduction

Starting with AlexNet [1] deep convolutional neural networks (DCNNs) have been gaining attention by delivering impressive results on challenging problems, such as object recognition on ImageNet dataset [2] or facial recognition [3]. The adaptation of such DCNNs and deep neural networks (DNNs) in various applications including autonomous driving, medical diagnosis [4], [5], and machine translation [6] led to an ever increasing amounts of data to process under high performance requirements.

Most of these applications can be described as supervised learning tasks, split into training phase and inference. In the training phase, the algorithm is optimized to solve a certain task for the training data. The architecture of a DCNN or DNN is defined by the number of layers and their functionality (e.g., convolutional, fully connected, pool, and batch-normalization) and the layer-specific parameters which define the dimensions and behavior of the layer in forward and backward-propagation. To train the defined architecture on a given training data, the labeled data is fed through the network and in back-propagation, the layer-specific weights are adjusted to decrease the error between the output and original label. For inference, DNN employs the model derived during training phase on the test or unknown data. The ability to correctly process the new data based on training data is called generalization ability.

Despite the state-of-the-art DNNs taking one or several high-end graphics processing units (GPUs) and up to several days to train, inference can be performed on a broad spectrum of platforms including CPUs, GPUs, field-programmable gate arrays (FPGAs), and application specific integrated circuits. With the increasing size of DNNs (e.g., ResNet [7] up to 152 layers), even the complexity of inference is also exacerbating due to more critical requirements and constraints such as limited power consumption, high throughput or hard real-time processing. There are several challenges that hinder the efficient deployment and inference of the state-of-the-art DNNs on embedded resource constrained platforms. The two biggest challenges are the large size of the networks and the total number of necessary operations in feed-forward computation, since a hardware accelerator design can be bound either by the limit of parallel operations, or by the memory interface transmission rate [8], [9]. As a consequence, model compression and increasing the efficiency of computations, are two legitimate ways to reach hardware requirements.

Recent works [10], [11] have proven the robustness of DNNs to compression of weights and simplification of activation functions with high number of parameters and the resulting redundancy [5], [10], [12]–[14]. This enables several techniques including weight sharing [10], [15], pruning [16], and Huffman encoding [10] to reduce external memory access. Pruning not only reduces the memory footprint of a DNN model, but also allows skipping of multiplications with 0,

Manuscript received April 3, 2018; revised June 8, 2018; accepted July 2, 2018. Date of publication July 18, 2018; date of current version October 18, 2018. This article was presented in the International Conference on Hardware/Software Codesign and System Synthesis 2018 and appears as part of the ESWEEK-TCAD special issue. *(Corresponding author: Matthias Wess.)*

M. Wess is with the Institute of Computer Technology, TU Wien, 1040 Vienna, Austria, and also with the Department of Corporate Technology, Siemens AG, 1210 Vienna, Austria (e-mail: matthias.wess@student.tuwien.ac.at).

S. M. P. Dinakarrao is with the Electrical Engineering Department, George Mason University, Fairfax, VA 22030 USA (e-mail: saimanoj.p.2013@ieee.org).

A. Jantsch is with the Institute of Computer Technology, TU Wien, 1040 Vienna, Austria (e-mail: axel.jantsch@tuwien.ac.at).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2018.2857080

thus reducing the amount of total multiplications [17], [18]. To reduce also power consumption within operations, the model parameters have to be quantized in specific formats a dedicated hardware can make use of. Dynamic fixed point (DFP) [14], [19] and power of two quantization [11] are two hardware friendly formats that enable performing multiplications either as low-precision multiplications or simple shift operations.

There are several approaches on how to best prepare a DNN for inference with low precision data types. On one side when employing the state-of-the-art DNNs it is desirable to directly make use of pretrained models without architectural adjustments. Lin *et al.* [13] and Zhou *et al.* [20] proposed methods for layer-wise bit-width optimization without retraining but not for bit-width optimization followed by retraining. Furthermore, to fully leverage optimized hardware accelerators for efficient inference (e.g., [21]) it can be desirable to force certain quantization [11], compression or pruning schemes [17] in an additional fine-tuning step. Zhou *et al.* [11] proposed incremental weight quantization while incorporating a power-of-two (Po2) data type and achieve almost lossless quantization for several DNNs. Other works such as [19] and [22] employ stochastic quantization methods to during training. In stochastic training the algorithm stores a floating point value and the quantized value at the same time and for each feed forward computation the quantized weights are newly computed on a stochastic basis.

This paper makes the following contributions.

1) We propose weighted quantization-regularization (WQR), a method for trained low precision quantization of weights in neural networks to any given quantization scheme.
2) We combine layer-wise precision scaling [20] with WQR to reduce the loss in classification performance while increasing the compression rate.
3) We analyze the benefits of Po2 and DFP-based quantization in our setting and in combination with WQR and layer-wise bit-width optimization.

Aiming at highly efficient implementation in FPGAs, we perform evaluation for quantization-regularization (QR) for DFP [22] and power of two quantization [11] schemes on convolutional neural networks (CNNs). We apply the proposed algorithm on SVHN CIFAR-10 and CIFAR-100 dataset for two different quantization schemes and show that WQR decreases loss in classification performance in comparison to direct weight quantization for all-convolutional network on CIFAR-10 from 1.5% to 0%. The results suggest that the proposed algorithm reduces accuracy loss due to quantization.

II. Motivational Case Study

Fig. 1 explains with a simple example the two main parts of this paper. Assuming a two-layer neural network with two layers with 600 and 900 weights, respectively, we want to achieve model compression by reducing the number of bits stored per weight and specific quantization of the weights to enhance the computational energy efficiency.

First note, that layer 2 has a stronger impact on the size of the weight memory, as it contains more weights. Thus, it is beneficial in terms of memory footprint to reduce the bit-width of its weights more than those of layer 1. However, quantization also negatively affects the accuracy of the algorithm, due to weight quantization errors. Therefore, we apply layer-wise precision scaling [Fig. 1(a)] to find the best trade-off between compression due to quantization and accuracy degradation. While for the example in Fig. 1 uniform 3-bit quantization leads to 4.5-kB weight memory, with layer-wise precision scaling applied according to Fig. 1(a) only 3.6-kB weights need to be stored, allowing us to increase compression ratio by a factor 1.25.

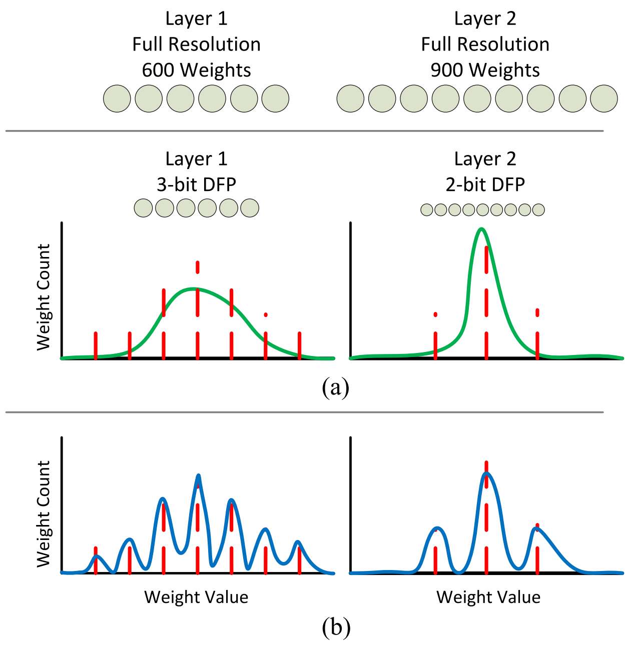

Fig. 1. Example network with two layers demonstrating (a) layer-wise precision scaling and (b) retraining with QR. Starting with two layer network at first (a) bit-widths of both layers are adjusted by defining a lower precision format with the quantization levels marked as dotted lines in the histogram charts. To reduce the quantization error, (b) retraining with additional regularization, decreasing the average distance of the weights to the quantization levels, is performed.

To alleviate the accuracy degradation, performing trained quantization by applying additional regularization with the goal of reducing the weight quantization error results in an increase the accuracy. For state-of-the-art DNNs layer-wise precision scaling shows even higher efficiency due to the higher variation of numbers of weights per layer (Table III).

III. Proposed Method

Fig. 2 illustrates the entire quantization flow for learned model compression which can be separated into three steps.

1) *Quantization Scheme Evaluation:* We define and analyze two quantization strategies in terms of their effectiveness for hardware-friendly execution their advantages and disadvantages during fine-tuning and the resulting performance.
2) *Layer-Wise Precision Scaling:* To increase the model compression ratio we apply layer-wise precision scaling, meaning that for each layer different bit-widths are used for weights. Thereby, we study the influence of

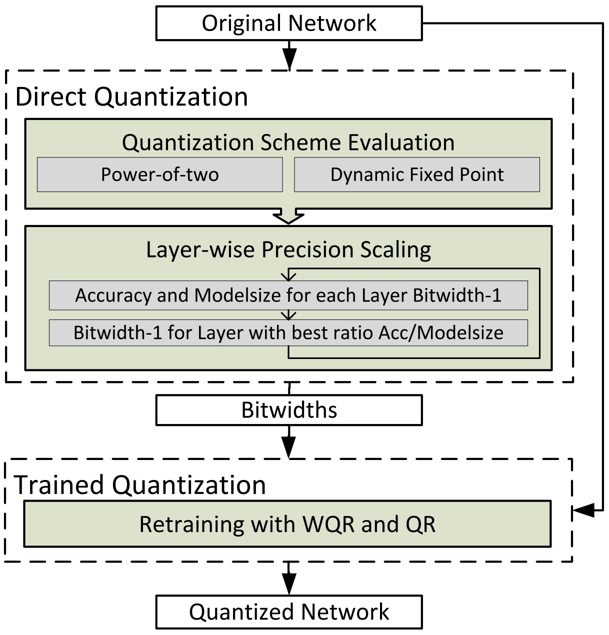
Original Network
Direct Quantization
Quantization Scheme Evaluation
Power-of-two
Dynamic Fixed Point
Layer-wise Precision Scaling
Accuracy and Modelsize for each Layer Bitwidth-1
Bitwidth-1 for Layer with best ratio Acc/Modelsize
Bitwidths
Trained Quantization
Retraining with WQR and QR
Quantized Network

Fig. 2. Learned weight quantization step by step.

selecting different bit-widths per layer on the resulting classification accuracy.

3) *Retraining With WQR and QR:* The last task focuses on reducing accuracy degradation occurring due to quantization. As loss of accuracy is induced due to the change of weight magnitudes when approximating them by rounding to the nearest quantization level, we aim to force weights to reduce their distance to such quantization levels in retraining, thus increasing classification accuracy of the quantized network.

While the first two steps serve for finding the best quantization method and bit-width for each layer when applying quantization without retraining, in the third step we perform retraining aiming to reduce the accuracy loss caused by quantization. In our flow, the weights are not first quantized and then retrained, but we always start from the high accuracy model, fine-tune weights with modified loss-functions and then perform quantization. This approach has the advantage that the network parameters are trained in full precision but with the additional regularization terms which cause the weights to reduce their distance to the desired quantization levels before performing the actual quantization step. To better distinguish we use the term direct quantization for quantization without any fine-tuning. Trained quantization on the other hand consists of fine-tuning, followed by the actual quantization step. Input for the quantization process is a DNN with N convolutional and/or fully connected layers and weight-tensors $W_n, 0 < n < N$ of arbitrary resolution. Details on the network used for evaluation can be found in Table III. Table I lists the variables used in this paper.

A. Direct Quantization

In direct quantization, the original network M is expressed as M_q where the weights W_n of each layer are represented as W_{q_n}. The values of W_{q_n} are determined by rounding each element of W_n to the quantization level with the smallest absolute distance of a defined quantization scheme Q.

TABLE I
VARIABLES USED IN THIS PAPER

Variable	Comment
M	Original network model
M_q	Quantized network model
N	Number of layers
W_n	Original Weights
W_{q_n}	Quantized weights
b_n	Bit-widths for layers $n = 0...N$
Q_n	Quantization schemes for layers $n = 0...N$
Q_{p2}	Power-of-2 quantization scheme
n_1	Maximum exponent for Power-of-2 quantization
n_2	Minimum exponent for Power-of-2 quantization
s	Maximum absolute weight within the selected layer
Q_{dfp}	Dynamic Fixed Point quantization scheme
B	Unscaled Dynamic Fixed Point quantization scheme
acc_M	Classification accuracy of the original network
acc_{M_q}	Classification accuracy of the quantized network
Δacc	Accuracy degradation due to quantization
W_{mem}	Weight memory bits
λ_1	Quantization-Regularization scale factor
QR	Quantization-Regularization Term
λ_2	Weighted Quantization-Regularization scale factor
WQR	Weighted Quantization-Regularization term

1) Quantization Scheme Evaluation: Here, we present Po2 [11] and DFP [19], [22], two different quantization schemes and compare their properties for direct and trained quantization.

a) Power-of-Two Quantization: We implement Po2 quantization similar as in [11]. Q_{p2} is given as

$$Q_{p2} = \left\{\pm 2^{n_1}, \ldots, \pm 2^{n_2}, 0\right\}. \tag{1}$$

n_1 and n_2 are integers with

$$n_1 = \lfloor \log_2 \frac{4s}{3} \rfloor \tag{2}$$

$$s = \max(abs(W)). \tag{3}$$

For a given bit-width b and n_2 are defined by

$$n_2 = n_1 - \left(2^{b-1} - 1\right). \tag{4}$$

Thus, the quantization levels depend on the distribution of weights, especially on the weight with the highest absolute value. By adding "0" as a quantization level, we enable Po2 quantization to also serve as a pruning mechanism when applied to weight matrices, as small weights are rounded to zero. In experiment symmetrical quantization schemes lead to higher classification accuracies for the quantized networks, therefore we only use $2^b - 1$ of 2^b possible quantization levels.

b) Dynamic Fixed Point: DFP data type is successfully used in several works for either direct quantization or retrained model compression [19], [22]. For DFP quantization, we first define a set of $2^b - 1$ equidistant quantization levels

$$B = \left\{\pm 2^{b-1} - 1, \pm 2^{b-1} - 2, \ldots, 0\right\}. \tag{5}$$

Similar to Po2 quantization, we prefer a symmetric quantization scheme. Next B is normalized and scaled, depending on the distribution of weights

$$Q_{\mathrm{dfp}} = \frac{B}{2^{b-1}} * 2^{n_1}. \tag{6}$$

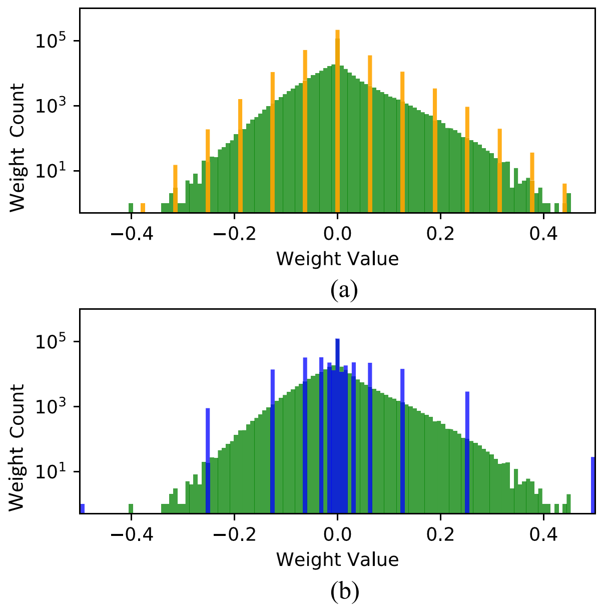

Fig. 3. Distributions before and after direct weight quantization for (a) DFP and (b) Po2 quantization.

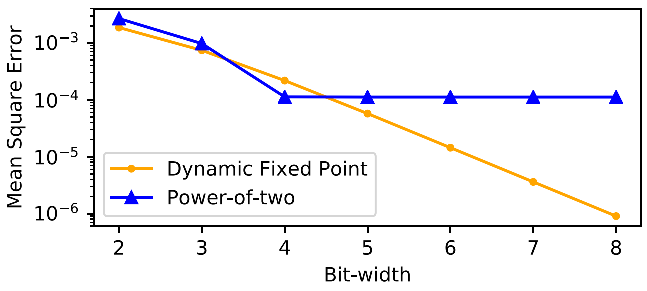

Fig. 4. MSE for an example layer when applying Po2 and DFP quantization with different bit-widths. While DFP quantization decreases the quantization error exponentially with increasing bit-width, with Po2 quantization the quantization error reaches the minimum already at bit-width 4.

Fig. 3 depicts the distribution of weights for an example layer of a CNN, before and after quantization. While Fig. 3(a) shows the distribution for Po2 quantization, Fig. 3(b) illustrates the distribution for DFP quantization. As can be seen that po2 has much irregular quantization values compared to DFP, and also considers the values close to 0 which might help to retain the information with lower weights and aid in improving the accuracy.

As accuracy degradation of the quantized model M_q in comparison to the original model M is a result of the quantization error, it is necessary to understand the relation between bit-width and quantization error for both data types.

With DFP quantization, the mean square error (MSE) can be reduced with increasing bit-width, since every additional bit divides the intervals in half [see Fig. 3(a)]. Meanwhile when increasing bit-width in Po2 quantization, the new quantization levels are always added close to 0 [see Fig. 3(b)]. As a consequence with Po2 quantization, the quantization error can only be reduced to a certain extent. Fig. 4 shows the resulting MSEs for one weight-tensor of an example layer when applying different bit-widths.

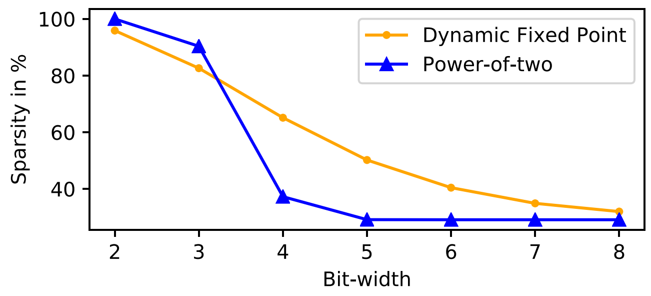

Fig. 5. With increasing bit-widths the sparsity due to quantization decreases for DFP and Po2 quantization. Sparsity denotes the amount of weights that are 0 relative to the total amount of weights.

In addition, we consider the amount of pruned weights as an important factor for model compression. In comparison to DFP, Po2 quantization decreases sparsity within the weight matrices as a result of quantization, due to the higher density of levels close to 0. Therefore to fully benefit from the advantages of sparsity, an additional pruning step before retraining is recommended. In Fig. 5, the number of pruned weights depending on the selected bit-width is shown for Po2 and DFP quantization.

Figs. 4 and 5 suggest that for direct Po2 quantization bit-widths higher than 4 bits do not further decrease the Δacc but 4 bits in comparison to 5 bits slightly increase sparsity. On the other hand, by scaling the bit-width of DFP the resulting MSE can be reduced exponentially (Fig. 4) meaning that even bit-widths higher than 8 bits deliver more accurate results. In terms of sparsity, DFP prunes more weights than Po2 for bit-widths of four and higher.

Based on these observations we expect that for direct DFP quantization Δacc can be reduced to almost 0, based on layer-wise precision scaling. For direct Po2 quantization we expect a higher Δacc and no significant increase of accuracy for bit-widths higher than five. It can be seen in Figs. 6 and 10 that these expectations are confirmed.

2) Layer-Wise Precision Scaling: Second, to further reduce the model-size, we apply different quantization schemes per layer by optimizing bit-widths. In comparison to choosing equal bit-width for each layer, due to the varying amount of parameters and varying distribution of weights between layers, selecting fitting quantization schemes for each layer can enable lower bit-widths per layer without reducing the resulting accuracy. For the experiments we assume either DFP or Po2 quantization. For an arbitrary network M with accuracy acc_M applying weight quantization with the set of bit-widths b_n leads to accuracy acc_{Mq} and weight memory bits

$$W_{\text{mem}} = \sum_{n}^{N} \text{card}(W_n) * b_n \tag{7}$$

where $\text{card}(A)$ denotes the cardinality of set A. For each b_n, we compute the resulting accuracy degradation

$$\Delta\text{acc} = \text{acc}_M - \text{acc}_{Mq} \tag{8}$$

and iteratively decrease the bit-width of the layer where a lower bit-width leads to the smallest product of $\Delta\text{acc} * W_{\text{mem}}$ (see Algorithm 1).

Algorithm 1 Layer-Wise Precision Scaling

```
procedure LAYER-WISE PRECISION SCALING($M$)
    initialize $b_n$
    while $\Delta acc < \epsilon$ do
        for all $n$ in layers do
            bitwidth of $layer_n$ - 1
            Compute $Acc_{M_q}$, $\Delta acc$ and $W_{mem}$
            bitwidth of $layer_n$ + 1
        end for
        Decrease bitwidth of layer with min($\Delta acc * W_{mem}$)
    end while
end procedure
```

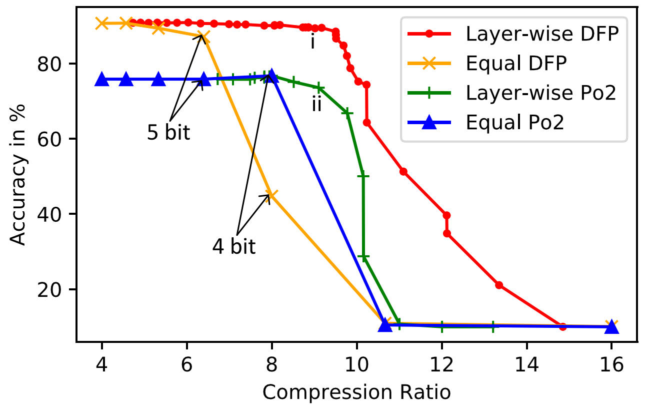

Fig. 6. Layer-wise precision scaling compared with equal bit-width quantization for DFP and Po2 quantization. Compression ratio is the ratio between 32-bit weight memory and the weight memory for the quantized network. Point (i) indicates DFP with $b_n = [7\ 7\ 7\ 4\ 4\ 3\ 3\ 7\ 7]$ and point (ii) indicates Po2 with $b_n = [4\ 4\ 4\ 4\ 3\ 3\ 4\ 4\ 4]$.

Fig. 6 shows the results for layer-wise precision scaling performed by Algorithm 1 on all-convolutional network [23] for CIFAR-10. We can deduce that while DFP quantization also allows direct quantization, whereas for Po2 quantization almost always an additional fine-tuning step is necessary to achieve high accuracy results.

B. Trained Quantization

The used datasets (CIFAR-10, CIFAR-100, and SVHN) are already divided into *test data* and *training data*. While with layer-wise precision scaling as described in Section III-A2 focuses on decreasing $\Delta\text{acc} * W_{\text{mem}}$ without retraining, we can additionally reduce Δacc by retraining the original network on the training data with the goal of increasing accuracy of the classifier on test data. As a consequence we use the performance metrics in Table II.

1) Quantization-Regularization: To decrease Δacc for a selected set of bit-widths b_n, we need to find the best set of W_n so that approximation with Wq_n achieves a maximum of acc_{Mq}. As shown in [13] and [24], the degradation of classification accuracy of a DNN due to quantization is directly related to the signal to quantization-noise ratio (SQNR) and the amount of weights per layer, as both influence the SQNR of the intermediate layer outputs and as a consequence the resulting network outputs. Therefore, retraining network weights

TABLE II
PERFORMANCE METRICS USED FOR FINE-TUNING

Variable	Metrics	Comment
acc_{tr}	Training Accuracy	Accuracy of the network on the training data
acc_M	Test Accuracy	Accuracy of the network on the test data
acc_{M_q}	Quantized Test Accuracy[1]	Accuracy of the network with quantized weights on the test data
acc_{init}	Init. Quantized Test Accuracy	Accuracy of the network with direct quantized initial weights on the test data
CR	Compression Ratio	Ratio W_{Mem} of original model to W_{Mem} of quantized model

to achieve lower SQNR without reducing acc_M, leads to an increased accuracy of the quantized network. To enforce weight quantization during the training phase we define the QR term as

$$\text{QR} = \sum_{n}^{N} \sum_{i}^{\text{card}(W_n)} \frac{|\, W_{n_i} - Wq_{n_i} \,|}{\max(Q_n) * \text{card}(W_n)} \tag{9}$$

which expresses the mean of the absolute weight distances of each weight to the corresponding quantized value.

By adding the QR-term to the loss function (10) weights are forced closer to the quantization levels during retraining

$$\text{Modified Loss} = \text{Loss} + \lambda_1 * \text{QR} \tag{10}$$

During fine-tuning with the parameter λ_1 the tradeoff between min(Loss) and min(QR) and, as a consequence, between min(Δaccuracy) and max(acc_M) can be adjusted. For the experiments we applied fixed λ_1 and linearly increasing λ_1 (e.g., $\lambda_1 = 10 * \text{epoch}$). Fig. 7 depicts the trained quantization process. With each epoch the weights are pulled closer to the quantization levels, thus decreasing QR and Δacc.

While choosing a high (>1000) λ_1 leads to fast quantization with strong accuracy degradation, a low λ_1 value (< 1) does not enforce quantization. Either way, much like weight decay low λ_1 values can help avoiding overfitting during training.

2) Weighted Quantization-Regularization: While in normal QR each weight within one layer is considered equally important for reaching high classification accuracy, the efficiency of pruning [15]–[17] shows that especially weights with small magnitudes can be changed without reducing the accuracy of the network. Similarly to [8], the weights can be divided into two disjoint subsets, where QR is applied on one of the subsets while the other weights are being retrained without QR. Going one step further we can multiply the QR value of each weight with the absolute magnitude of the weight.[2] This strategy forces quantization stronger on weights with higher magnitudes which can be especially useful for Po2 quantization, where density of quantization levels decreases with increasing

[1]For the computation of quantized test accuracy, the weights of the network are directly quantized after each epoch.

[2]Previously the sum of weights is normalized to 1 for each layer.

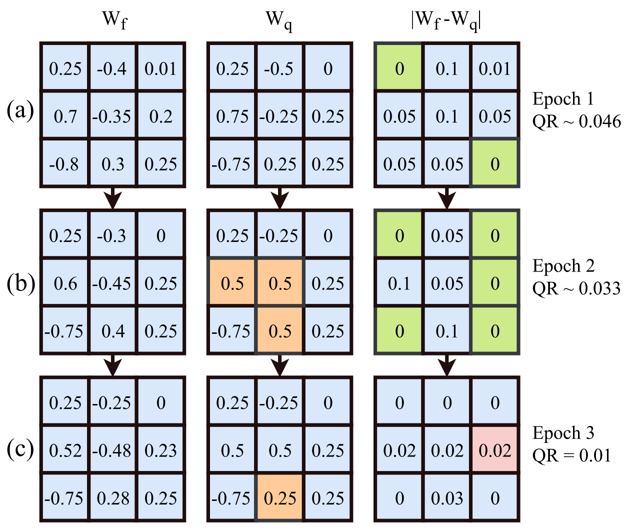

Fig. 7. Illustration of QR. The first column shows the floating point values of the weights on which the actual training is performed. The second column shows the quantized weights, and the third column the element-wise absolute difference of floating point and quantized weights. Changes of quantized weights are marked in the second column. Weight update for W_f is performed based on backpropagation of the modified loss function (10). Successfully quantized weights are marked in the third column. After Epoch 1 (a) the weights are hardly regularized and QR is relatively large. Epoch 2 (b) shows that due to regularization, weights are pulled closer to the quantization levels $Q_n = \{0, \pm 0.25, \pm 0.5, \pm 0.75\}$ and QR gets smaller. For three of the weights the resulting quantization level changed, and weights decrease their distance to the next quantization level. Epoch 3 (c) shows further reduction of the QR term.

weight values. Therefore, we define the WQR term as

$$\text{WQR} = \sum_{n}^{N} \sum_{i}^{\text{card}(W_n)} \left(\frac{| W_{n_i} - Wq_{n_i} || W_{n_i} |}{\max(Q_n)^2 * \text{card}(W_n)} \right) \qquad (11)$$

and similarly to (10) we can weight the tradeoff between accuracy and weight regularization with λ_1 and λ_2 (12)

$$\text{Modified Loss} = \text{Loss} + \lambda_1 * \text{QR} + \lambda_2 * \text{WQR}. \qquad (12)$$

Again during training the parameters λ_1 and λ_2 have to be adjusted carefully to reach the desired improvement of Acc_{Mq}, without at the same time decreasing Acc_M. In our experiments we found a linear increasing λ_2 to work best (e.g., $\lambda_2 = 10 *$ epoch). For fine-tuning, we use Algorithm 2.

Fig. 8 illustrates the fine-tuning process for 4-bit equal bit-width Po2 quantized all-convolutional net for CIFAR-10. At the beginning of the fine-tuning process the term $\lambda_2 *$ WQR increases due to the increasing λ_2 value, while the WQR-Term decreases exponentially. At epoch 200, learning rate is decreased from $1e^{-4}$ to $1e^{-5}$ leading to the drop of $\lambda_2 *$WQR. This can be explained by fact that a larger learning rate leads to larger weight changes. If the weights are already close to the quantization levels a lower learning rate can lead to better approximation of the weights to the quantization levels.

C. Summary and Analysis

In combination, the discussed techniques for quantization (Section III-A), layer-wise precision scaling (Section III-A2), fine-tuning with QR (Section III-B1), and WQR (Section III-B2) facilitate DNN weight compression.

Algorithm 2 Fine-Tuning With QR and WQR

```
procedure TRAINED QUANTIZATION($M$, $b_n$, $\lambda_1$, $\lambda_2$)
    for epochs do
        for all Mini Batches do                ▷ Train on Training Data
            for $n >= N$ do                    ▷ Quantize all Layers
                $W_{q_n}$ = Quantize($W_n$,$Q_n$)
            end for
            Loss + $\lambda_1 * QR(W_n, W_{q_n}) + \lambda_* WQR(W_n, W_{q_n})$
            Backpropagation(Loss,QR,WQR,$\lambda_1$,$\lambda_2$)
        end for
        Compute $Acc_M$                        ▷ Test Accuracy
        Compute $Acc_{M_q}$                    ▷ Quantized Test Accuracy
    end for
    return $M_q$                               ▷ Return Quantized Model
end procedure
```

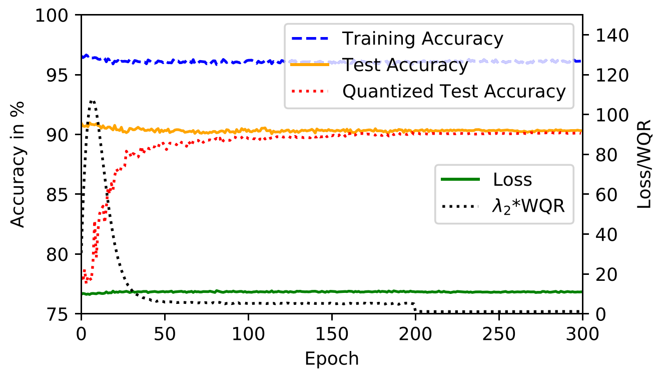

Fig. 8. Fine-tuning of all-convolutional net for CIFAR-10 with linear increasing λ_2 for 4 bits equal bit-width Po2 quantization. ΔAcc is decreased from 14.09% to 0.14% resulting in quantized test accuracy of 90.18% in comparison to initial floating point test accuracy 90.83%.

The above discussed two quantization schemes behave differently during the quantization process and require different quantization strategies. For bit-widths higher than 7-bit DFP can be applied without any retraining and still achieves almost floating point accuracy. For equal bit-width DFP quantization with 7 bits and less, Δacc increases and fine-tuning is necessary to reach the accuracy of the original network. On the other hand Po2 quantization always requires retraining, as even the use of bit-widths higher than five reduce the quantization error only to a certain extent.

To increase the compression ratio when applying DFP quantization, layer-wise precision scaling is an effective method, since not all layers require the same the bit-width for high accuracy. For instance, in modern convolution-only networks, layers with fewer parameters require larger bit-widths [13]. As a result, when applying layer-wise precision scaling the weights of the output and input layers are kept at high precision, as they usually have the fewest parameters. In comparison to DFP, for Po2 quantization, layer-wise precision scaling does not proof to be as effective, as almost the same bit-width is recommended throughout the network to achieve best accuracy.

For fine-tuning of Po2 and DFP quantized networks, QR and WQR can be added to the loss function as regularization

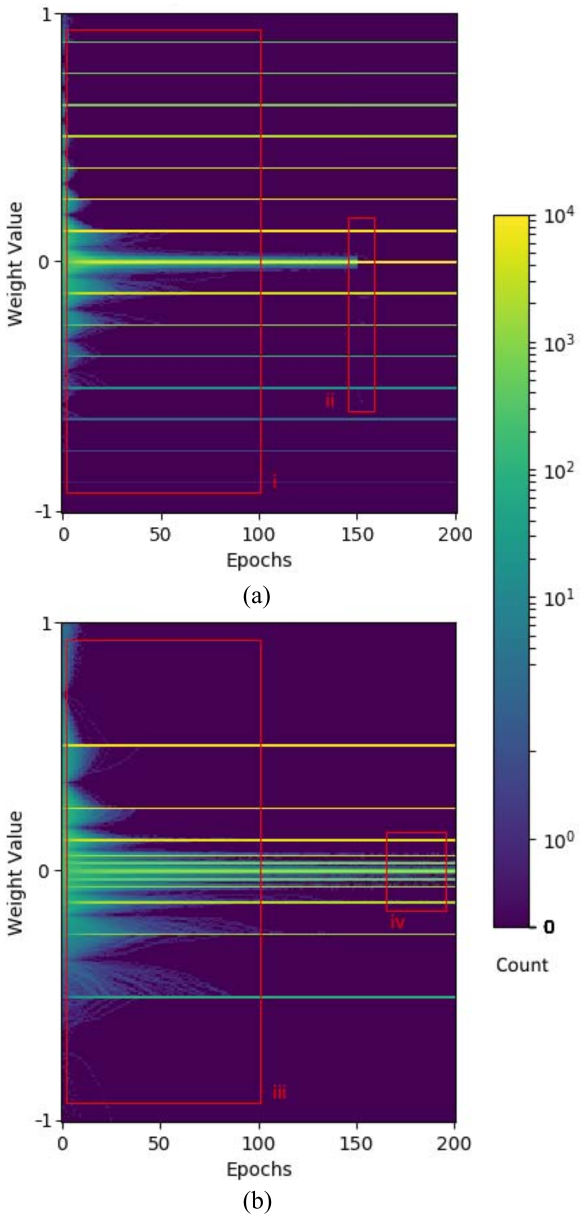

Fig. 9. Distribution of weights in layer 5 of AllConvNet (CIFAR-10) during fine-tuning with WQR and QR to (a) 4-bit DFP and (b) 4-bit Po2. DFP (a) is trained with $\lambda_2 = \text{epoch} * 10$. Region (i) shows the faster quantization of weights with high magnitudes and slower quantization of weights close to 0. At epoch 150, QR is applied with $\lambda_1 = 100$ to quantize also weights with smaller values (ii). Po2 (b) is trained with $\lambda_2 = \text{epoch} * 10$. Region (iii) also shows slower quantization for small magnitude weights. Due to the distribution of quantization levels, weights around 0 are already close to the next quantization level (iv).

terms to force quantization during retraining. While the QR term forces all weights equally to reduce the distance to the next quantization level, the WQR term is reduced for weights with small magnitudes. Therefore, WQR operates less restrictive than QR, as the QR value is multiplied with a factor from 0 to 1 (normalized weight magnitude). As a consequence it is an effective strategy to apply WQR followed by QR fine-tuning.

Fig. 9 shows the distribution of weights in layer 5 during Fig. 9(a) 4-bit DFP and Fig. 9(b) 4-bit Po2 quantization of the all-convolutional network for the CIFAR-10 dataset. For fine-tuning to 4-bit DFP quantization [Fig. 9(a)] λ_2 is increased linearly by a factor of 10 with each epoch. As QR scale factor λ_1 we use 0 before and 100 starting at epoch 150. Region (i) in Fig. 9(a) shows the decelerated quantization of small magnitude weights. Starting with epoch 150, also weights close to 0 are quantized, due to the application of QR [Fig. 9(a)ii]. For 4-bit Po2 quantization only WQR is necessary ($\lambda_2 = \text{epoch}*10$) to achieve quantization. Due to the nonequidistant distribution of quantization levels in Po2 quantization, the weights with high magnitudes take longer than in DFP quantization to reach the quantization levels [Fig. 9(b)iii]. Additional QR is not necessary since, due to the high density of quantization levels close to 0, the weights with smaller magnitudes induce a very small quantization error [Fig. 9(b)iv].

TABLE III
ALL-CNN-C ARCHITECTURE AND THE NUMBER OF WEIGHTS AND MAC-OPERATIONS FOR ONE FORWARD COMPUTATION WITH BATCH-SIZE ONE

Layer (WxH)	**Output Dim. (WxHxD)**	**#Weights**	**#MACs**
Input	32x32x3	-	-
Conv. 3x3	32x32x96	2K	2.3M
Conv. 3x3	32x32x96	83K	74.6M
Conv. 3x3	32x32x96	83K	74.6M
Pooling 2x2	16x16x96	-	-
Conv. 3x3	16x16x192	165K	32.5M
Conv. 3x3	16x16x192	332K	65M
Conv. 3x3	16x16x192	332K	65M
Pooling 2x2	8x8x192	-	-
Conv. 3x3	8x8x192	332K	11.9M
Conv. 1x1	8x8x192	37K	2.4M
Conv. 1x1	8x8x10	2K	0.1M
Pooling 8x8	10	-	-
Total	-	**1386K**	**329.7M**

QR and WQR enable trained quantization to improve performance in comparison to direct quantization. Regularization-based quantization is very simple to implement and applicable for any quantization scheme. As a consequence QR and WQR could be employed alongside other effective quantization techniques such as stochastic quantization. Besides the simplicity of the approach, it also allows a deeper analysis of quantization schemes by recording the weight distribution during training.

While enabling compression for more efficient inference, during training QR and WQR add overhead due to the mandatory quantization step after each mini-batch and the necessary computations for calculation of QR and WQR. The fact that all weights have to be stored as floating point values and quantized values, increases the weight memory during training by a factor of 2.

IV. Experimental Results

The following section describes the experimental results for direct and trained quantization of All-CNNs on three datasets.

A. Experimental Setup

For experimental evaluation, we apply the proposed methods on all-convolutional networks for the datasets

TABLE IV
RESULTS IN COMPARISON TO FLOATING POINT BASELINE OF ALL-CNN FOR CIFAR-100 (CR = COMPRESSION RATIO)

b_n	Type	$W_{mem}[Bit]$	CR	Sparsity[%]	non-0 MACs	MAC Sparsity[%]	acc_{tr}[%]	acc_{init}[%]	acc_{M_q}[%]
32 bit	float	44344K	1	0	329.7M	0	83.24	63.03	**63.03**
8 bit	DFP-eq	11086K	4	5.8	304.8M	7.6	83.5	62.43	**62.99**
7 bit	DFP-eq	9700K	4.6	11.5	280.3M	15	82.65	61.86	**62.38**
6 bit	DFP-eq	8314K	5.3	21.9	234.1M	29	81.02	58	**61.19**
5 bit	DFP-eq	6928K	6.4	38.9	161M	51.2	76.99	43.72	**59.54**
[9 9 9 9 6 5 7 9 9]	DFP-lw	9485K	4.7	14.6	289.5M	12.2	83.44	62.90	**62.93**
[9 9 9 9 5 5 5 9 9]	DFP-lw	8490K	5.2	23.2	279.2M	15.3	82.45	62.45	**62.64**
[9 9 9 5 4 4 5 9 9]	DFP-lw	7163K	6.2	36.4	243.4M	26.2	80.76	61.70	**62.29**
[9 6 5 5 3 3 4 7 9]	DFP-lw	5513K	8.0	60.4	133.6M	59.5	73.76	55.69	**58.58**
4 bit	Po2-eq	5543K	8	18.3	255.2M	22.6	79.47	51.27	**60.94**

TABLE V
RESULTS IN COMPARISON TO FLOATING POINT BASELINE OF ALL-CNN FOR CIFAR-10

b_n	Type	W_{mem}[Bit]	CR	Sparsity[%]	non-0 MACs	MAC Sparsity[%]	acc_{tr}[%]	acc_{init}[%]	acc_{M_q}[%]
32 bit	float	43791K	1	0	329.7M	0	96.69	90.83	**90.83**
8 bit	DFP-eq[4]	10947K	4	4.2	303.5M	7.6	96.64	90.65	**90.85**
4 bit	DFP-eq	5474K	8	42.3	140.7M	57.2	94.4	44.47	**88.63**
[8 8 8 5 4 4 3 8 8]	DFP-lw[4]	5929K	7.38	36.8	243M	26	96.49	90.14	**90.81**
[7 7 7 4 4 3 3 7 7]	DFP-lw	5432K	8.1	46.1	203.9M	37.9	96.01	90.07	**90.32**
4 bit	Po2-eq	5474K	8	13.2	255.9M	22.1	96.13	76.73	**90.18**

TABLE VI
RESULTS IN COMPARISON TO FLOATING POINT BASELINE OF ALL-CNN FOR SVHN

b_n	Type	W_{mem}[Bit]	CR	Sparsity[%]	non-0 MACs	MAC Sparsity[%]	acc_{tr}[%]	acc_{init}[%]	acc_{M_q}[%]
32 bit	float	43791K	1	0	329.7M	0	97.70	95.84	**95.84**
4 bit	DFP-eq	5474K	8	43.5	166.8M	49.2	96.27	86.17	**95.37**
[6 4 4 3 3 3 4 5 6]	DFP-lw	4690K	9.3	62.7	126.6M	63.2	96.43	95.03	**95.89**
[5 4 4 3 3 3 3 3 3]	DFP-lw	4276K	10.23	70.5	116.7M	64.4	95.52	89.86	**95.36**
4 bit	Po2-eq	5474K	8	17.0	272.8M	16.9	97.02	91.93	**96.02**

CIFAR-10 [25], CIFAR-100 and SVHN [26]. We use all-convolution network model All-CNN-C from [23] for evaluation. The CNN architecture summed up in Table III consists of nine convolution layers and a global average pooling layer. Due to the similar filter-width and height the number of weights in the convolution layers mainly depends on the filter-depths. The number of operations also depends on the layer-output dimensions. However, it needs to be noted that the proposed technique is neither architecture nor dataset bound.

For the three datasets, we use the predefined training and test sets. For the floating point baselines we trained the CNN for 350 epochs with initial learning rate 10^{-3} multiplied by a fixed multiplier after epochs 200 and 300. To avoid overfitting, we use dropout with dropout rate 0.5 after layers. In contrast to the original All-CNN paper [23], the models are not regularized by weight decay to avoid interfering with the studied regularization methods. In terms of data augmentation we only apply horizontal flipping and random shifting by a maximum of 3 pixels.

B. Performance Analysis

We evaluate the performance of the proposed method comparing the test accuracy of the original network with the resulting accuracies after direct and trained quantization. For Tables IV–VI, we make use of the abbreviations in Table II. We aim to achieve high compression rates in terms of weight memory while maintaining high test accuracy. In addition to bit-width reduction we also consider the resulting sparsity as an important factor for possible further compression. Assuming skipping of multiplications with 0 weights, sparsity also reduces the amount of required multiply-accumulate (MAC)-operations for forward computation.[3] For all experiments except the two marked, bit-width for activations is 32-bit fixed point.

1) CIFAR-100: CIFAR-100 is an image classification dataset consisting of a training set of 50 000 and a test set of 10 000 32×32 color images representing 100 different categories such as airplanes, automobiles, birds, cats, deers, dogs, frogs, horses, ships, and trucks [25]. The training batches contain exactly 5000 images from each class. Table V shows the resulting accuracies after fine-tuning with 200 epochs of WQR with $\lambda_2 = \text{epoch} \times 10$. In addition, λ_1 is set to 100 starting at epoch 150. This setup is not optimal for all configurations, as in some cases training with only QR would be sufficient, but it allows using the same parameterization for each iteration increasing comparability. Compared to a full implementation (32 bits), the proposed layer-wise quantization (DFP-lw) saves ~4×–8× in terms of weight memory and has an accuracy loss by ~0.1%–4.5%. Compared to the reduced equal bit-width quantization, the proposed layer-wise quantization has higher sparsity and smaller weight memory.

[3]The number of skipped MACs due to a pruned weight depends on the layer-input dimensions. Therefore, sparsity in weights is unequal to sparsity in MACs.

[4]8-bit DFP for activations.

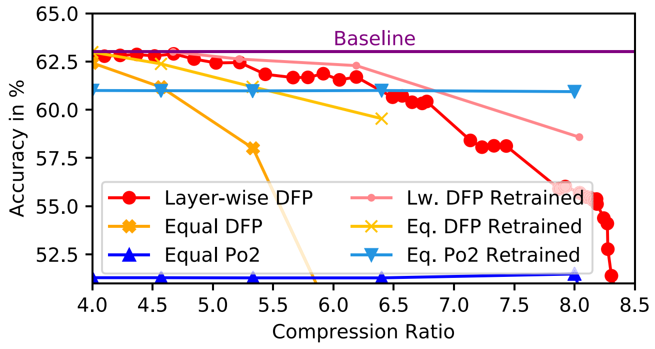

Fig. 10. Results for direct and trained DFP and Po2 quantization with equal bit-widths on ALL-CNN for CIFAR-100. For DFP also layer-wise precision scaling with direct and trained quantization is illustrated.

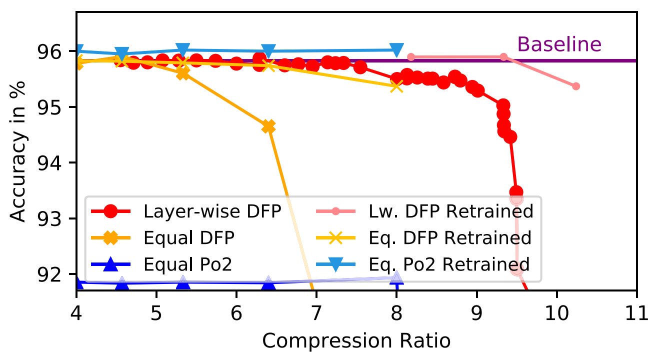

Fig. 11. Results for direct and trained DFP and Po2 quantization with equal bit-widths on ALL-CNN for SVHN. For DFP also layer-wise precision scaling with direct and trained quantization is illustrated.

In addition, Fig. 10 depicts the results of trained quantization in comparison to direct quantization. We can see that retraining with QR and WQR in all cases increases classification accuracy in comparison to direct quantization.

Comparing equal bit-width quantization with layer-wise precision scaling for DFP data type, we can see that for similar compression ratios, equal bit-width (DFP-eq) never reaches the accuracy of layer-wise precision scaling (DFP-lw), even when retraining with WQR/QR is applied. For Po2 quantization, we found equal 4-bit quantization (Po2-eq) the most effective method as higher bit-widths did not increase accuracy and layer-wise precision scaling for lower than 4 bits leads to drastic accuracy drop.

2) CIFAR-10: CIFAR-10 is a benchmark image classification dataset equal to CIFAR-100 in terms of image and dataset sizes, which instead of 100 classes divides the images into ten classes. We use the same training method as for CIFAR-100. The results for trained quantization of All-CNN for CIFAR-10 are shown in Table V. To allow comparing to other works for the two marked configurations, we also quantized the activations to 8-bit DFP. In contrast to ALL-CNN for CIFAR-100, for this dataset higher compression rates can be achieved. For compression ratio ~8, DFP-lw gives lowest accuracy degradation of 0.51% points. For equal bit-with quantization Po2 outperforms DFP by 1.55% points. For compression ratio 7.38 classification accuracy of DFP with layer-wise precision scaling is only 0.01% points below the floating point baseline.

3) SVHN: The SVHN image classification dataset consists of 694K 32×32 color images for training and 26K images for testing. The images represent digits form 0 to 9. Similarly to CIFAR-100 and CIFAR-10 we perform layer-wise precision scaling and retraining with WQR and QR for model compression. The results are shown in Table VI and Fig. 11. For the SVHN dataset at compression ratio ~ 8, Po2 quantization and DFP with layer-wise precision scaling both outperform the original baseline model.

Comparing the results for the datasets CIFAR-100, CIFAR-10, and SVHN, we can conclude that the attainable compression ratio for lossless trained quantization with QR/WQR not only depends on the selected data type and bit-widths, but also on the selected datasets. While for CIFAR-10 and SVHN, lossless trained quantization achieves compression ratio 8.0 and 9.4, respectively, for CIFAR-100 the maximum compression ratio is 4.7 for lossless compression. In the case of CIFAR-100 further increasing compression ratio up to 8.0 reduces accuracy by 2.09% points. In contrast to this tradeoff, stronger compression for CIFAR-10 and SVHN immediately leads to drastic accuracy reduction. We suspect that this difference can be explained by the relation between complexity of the dataset and the selected network. To better understand this relation, in future the techniques have to be applied for further datasets and network topologies.

While with DFP lossless compression can always be achieved, Po2 can sometimes lead to slight performance degradation. layer-wise precision scaling turns out to be more effective for DFP than for Po2. Po2 still reaches maximum accuracy at uniform 4-bit bit-width while achieving higher accuracy than uniform 4-bit and even 5-bit DFP.

V. Comparison With Related Work

The proposed method of WQR presents a novel technique for trained quantization of DNNs. However, in some works performing weight-binarization similar regularization methods are used to achieve weights with values "+1" or "−1" [27]. Today trained quantization is mostly performed by stochastic rounding during training [14], [19], [28], [29]. Gupta *et al.* [29] applied stochastic training for CIFAR-10 dataset to achieve fixed point quantization to 16 and 12 bits. Their accuracy is reduced by 0.8% and 4.2% points, respectively, compared to the floating point baseline. In comparison to that with our method we reach 8-bit DFP quantization without any performance drop.

Courbariaux *et al.* [14] performed quantization based on stochastic round for 10-bit DFP weights and activations. Their accuracy drops in comparison to the baseline networks 3.14% points for CIFAR-10 and 2.58% points for SVHN since in contrast to us, they also perform weight-update with 12-bit DFP which decreases comparability. For our CIFAR-10 and SVHN network, we experimentally also applied 8-bit DFP for weights and activations, and found that accuracy even increased after QR/WQR retraining. Gysel *et al.* [19] used their CAFFE-based tool Ristretto for layer-wise precision scaling and fine-tuning with stochastic rounding. On CIFAR-10, their accuracy lies

0.3% below the floating point baseline accuracy, when quantizing not only weights but also activations to 8-bit DFP. By applying layer-wise precision scaling we are able to increase compression ratio from 4 to 7.38 and after QR/WQR-retraining achieve equal to baseline accuracy while inducing higher sparsity due to the stronger compression. Other than fine-tuning with stochastic rounding, Zhou *et al.* [11] presented an incremental retraining method to perform Po2 weight quantization. They achieved lossless 5-/4-bit quantization for several DNNs for the ImageNet dataset. Even for lower bit-rates incremental quantization achieves state-of-the-art results. Even though this method seems highly promising, in contrast to this paper, it is only verified to work for Po2 quantization.

VI. Conclusion

We propose QR and WQR as techniques for improving accuracy after bit-width reductions inflicted by a quantization scheme. WQR/QR allow fine-tuning for weights in any quantization scheme and also works for nonuniform bit-widths (layer-wise precision scaling). For ALL-CNN with the CIFAR-10 benchmark WQR reaches lossless compression up to a ratio of 7.38× with DFP and layer-wise precision scaling. Compared to the 32-bit floating point baseline in the All-CNN network, WQR with DFP obtains a weight memory compaction of 8.0×–10.23× and a MAC sparsity of 37.9%–64.4% in the benchmark tasks. For these cases with maximal compaction we observe between 0.48% and 4.45% points reduction of classification accuracy. A high MAC sparsity benefits HW implementations because it potentially reduces the number of multiply accumulate operations.

Note, that WQR/QR is not a stand-alone tool, but can be combined with other techniques and is typically complementary to those. We have observed, that it has some limitations when applied to very low bit-widths; hence, its combination with stochastic rounding methods is considered as future work.

Furthermore, we have studied two quantization schemes, DFP and Po2, and we find that DFP is usually preferable to Po2 because it is as good as or better than Po2 in most cases, it can reach floating point accuracy when increasing bit-width, and it does not necessarily require retraining (retraining improves accuracy but for Po2 it is absolutely necessary). However, in a few special cases with low bit-width Po2 is slightly better and it might be preferred for HW implementations because it requires only shift operations instead of multiplications. Thus, when an optimized HW implementation is developed, Po2 could be considered as a useful option.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[2] J. Deng *et al.*, “ImageNet: A large-scale hierarchical image database,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Miami, FL, USA, 2009, pp. 248–255.

[3] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, “DeepFace: Closing the gap to human-level performance in face verification,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Columbus, OH, USA, 2014, pp. 1701–1708.

[4] P. Rajpurkar, A. Y. Hannun, M. Haghpanahi, C. Bourn, and A. Y. Ng, “Cardiologist-level arrhythmia detection with convolutional neural networks,” Stanford Univ. Mach. Learn. Group, Stanford, CA, USA, Project Rep. 17, 2017. [Online]. Available: https://arxiv.org/pdf/1707.01836.pdf

[5] M. Wess, P. D. S. Manoj, and A. Jantsch, “Neural network based ECG anomaly detection on FPGA and trade-off analysis,” in *Proc. IEEE Int. Symp. Circuits Syst.*, Baltimore, MD, USA, 2017, pp. 1–4.

[6] J. Zhang and C. Zong, “Deep neural networks in machine translation: An overview,” *IEEE Intell. Syst.*, vol. 30, no. 5, pp. 16–25, Sep./Oct. 2015.

[7] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, 2016, pp. 770–778.

[8] C. Zhang *et al.*, “Optimizing FPGA-based accelerator design for deep convolutional neural networks,” in *Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays*, Monterey, CA, USA, 2015, pp. 161–170.

[9] P. D. S. Manoj *et al.*, “A scalable network-on-chip microprocessor with 2.5D integrated memory and accelerator,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 6, pp. 1432–1443, Jun. 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7819521/

[10] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–14.

[11] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, “Incremental network quantization: Towards lossless CNNs with low-precision weights,” in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017. [Online]. Available: https://openreview.net/pdf?id=HyQJ-mclg

[12] X. Chen, X. Hu, H. Zhou, and N. Xu, “FxpNet: Training a deep convolutional neural network in fixed-point representation,” in *Proc. IEEE Int. Joint Conf. Neural Netw.*, Anchorage, AK, USA, 2017, pp. 2494–2501.

[13] D. D. Lin, S. S. Talathi, and V. S. Annapureddy, “Fixed point quantization of deep convolutional networks,” in *Proc. Int. Conf. Mach. Learn.*, New York, NY, USA, 2016, pp. 2849–2858.

[14] M. Courbariaux, Y. Bengio, and J.-P. David, “Training deep neural networks with low precision multiplications,” in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015. [Online]. Available: https://arxiv.org/pdf/1412.7024.pdf

[15] X. Dong, S. Chen, and S. Pan, “Learning to prune deep neural networks via layer-wise optimal brain surgeon,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4860–4874.

[16] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both weights and connections for efficient neural networks,” in *Proc. Adv. Neural Inf. Process. Syst.*, Montreal, QC, Canada, 2015, pp. 1135–1143.

[17] T.-J. Yang, Y.-H. Chen, and V. Sze, “Designing energy-efficient convolutional neural networks using energy-aware pruning,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 5687–5695.

[18] J. Yu *et al.*, “Scalpel: Customizing DNN pruning to the underlying hardware parallelism,” in *Proc. ACM Annu. Int. Symp. Comput. Archit.*, Toronto, ON, Canada, 2017, pp. 548–560.

[19] P. Gysel, “Ristretto: Hardware-oriented approximation of convolutional neural networks,” M.S. thesis, Elect. Comput. Eng., Univ. California Davis, Davis, CA, USA, 2016.

[20] Y. Zhou, S.-M. Moosavi-Dezfooli, N.-M. Cheung, and P. Frossard, “Adaptive quantization for deep neural network,” in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 1–13.

[21] S. Han *et al.*, “EIE: Efficient inference engine on compressed deep neural network,” in *Proc. IEEE Int. Symp. Comput. Archit.*, Seoul, South Korea, 2016, pp. 243–254.

[22] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, “Quantized neural networks: Training neural networks with low precision weights and activations,” *J. Mach. Learn. Res.*, vol. 18, pp. 1–30, Apr. 2018.

[23] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, “Striving for simplicity: The all convolutional net,” in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015. [Online]. Available: https://arxiv.org/pdf/1412.6806.pdf

[24] S. Shin, Y. Boo, and W. Sung, “Fixed-point optimization of deep neural networks with adaptive step size retraining,” in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2017, pp. 1203–1207.

[25] A. Krizhevsky, V. Nair, and G. Hinton. (Mar. 2009). *Cifar-10 and Cifar-100 Datasets*. [Online]. Available: https://www.cs.toronto.edu/kriz/cifar.html

[26] Y. Netzer *et al.*, “Reading digits in natural images with unsupervised feature learning,” in *Proc. NIPS Workshop Deep Learn. Unsupervised Feature Learn.*, 2011.

[27] W. Tang, G. Hua, and L. Wang, “How to train a compact binary neural network with high accuracy?” in *Proc. AAAI Conf. Artif. Intell.*, 2017, pp. 2625–2631.

[28] M. Courbariaux, Y. Bengio, and J.-P. David, “BinaryConnect: Training deep neural networks with binary weights during propagations,” in *Proc. Adv. Neural Inf. Process. Syst.*, Montreal, QC, Canada, 2015, pp. 3123–3131.

[29] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, “Deep learning with limited numerical precision,” in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1737–1746.

Matthias Wess received the B.Sc. and M.Sc. degrees from the Department of Electrical Engineering, TU Wien, Vienna, Austria, in 2013 and 2017, respectively, where he is currently pursuing the Ph.D. degree with the Institute for Computer Technology.

His current research interest includes hardware acceleration of deep neural network inference.

Sai Manoj Pudukotai Dinakarrao (S’13–M’17) received the Ph.D. in electrical and electronic engineering from Nanyang Technological University, Singapore, in 2015.

He was a Post-Doctoral Research Fellow with TU Wien, Vienna, Austria. He is a Research Assistant Professor with George Mason University, Fairfax, VA, USA. His current research interests include adversarial learning, digital design for machine learning, cyber-security for embedded processors, self-aware system on chip design, machine learning for on-chip data processing, and security in Internet of Things networks.

Dr. Dinakarrao was a recipient of the A. Richard Newton Research Young Research Fellow Award in DAC 2013.

Axel Jantsch (M’97) received the Ph.D. degree in computer science from TU Wien, Vienna, Austria.

He is a Professor of Systems on Chip (SoCs) with the Institute of Computer Technology, TU Wien. His current research interests include embedded machine learning and self-awareness in SoCs and embedded systems.