FIX-NET: PURE FIXED-POINT REPRESENTATION OF DEEP NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep neural networks (DNNs) dominate current research in machine learning. Due to massive GPU parallelization DNN training is no longer a bottleneck, and large models with many parameters and high computational effort lead common benchmark tables. In contrast, embedded devices have a very limited capability. As a result, both model size and inference time must be significantly reduced if DNNs are to achieve suitable performance on embedded devices.
We propose a soft quantization approach to train DNNs that can be evaluated using pure fixed-point arithmetic. By exploiting the bit-shift mechanism, we derive fixed-point quantization constraints for all important components, including batch normalization and ReLU. Compared to floating-point arithmetic, fixed-point calculations significantly reduce computational effort whereas low-bit representations immediately decrease memory costs. We evaluate our approach with different architectures on common benchmark data sets and compare with recent quantization approaches. We achieve new state of the art performance using 4-bit fixed-point models with an error rate of 4.98% on CIFAR-10.

1 INTRODUCTION

Deep neural networks (DNNs) are state of the art in many machine learning challenges, pushing recent progress in computer vision, speech recognition and object detection (Deng & Yu (2014); Lecun et al. (2015); Karki et al. (2019)). However, the greatest results have been accomplished by training large models with many parameters using large amounts of training data. As a result, modern DNNs show an extensive memory footprint and high-precision floating-point multiplications are especially expensive in terms of computation time and power consumption. When deployed on embedded devices, the complexity of DNNs is necessarily restricted by the computational capability.
Therefore, efforts have been made to modify DNNs to better suit specific hardware instructions. This includes both the transfer from floating-point to fixed-point arithmetic and the reduction in bit-size. This process is termed fixed-point quantization and especially low-bit representations simultanouesly reduce memory cost, inference time, and energy consumption. A survey is given in Sze et al. (2017). Furthermore, ternary-valued weights or even binary-valued weights allow replacement of many multiplications with additions[1].
However, most quantization approaches do not fit to the common structure in modern DNNs. State of the art architectures (such as ResNet, DenseNet, or MobileNetV2) consist of interconnected blocks that combine a convolution or fully-connected layer, a batch normalization layer and a ReLU activation function. Each block can be optionally extended by a pooling layer, as shown in Figure 1. Since both convolution and fully-connected layers perform weighted sums, we summarize the two as a Linear component. In contrast to the block structure, recent quantization approaches focus on the Linear component while preserving floating-point batch normalization (BN) layers. This is crucial, since BN layers are folded into the preceding layer after the training and consequently destroy its fixed-point representation. Even when performed separately, channel-wise floating-point multiplications make a pure fixed-point representation impossible. Furthermore, many quantization methods strictly binarize activations which only works for very large models.

[1]Binary-coded weight vectors, e.g. with values from $\{-1, 1\}$ or $\{0, 1\}$, replace multiply-accumulate operations by simple additions and subtractions, respectively. Since each ternary-coded vector with values from $\{-1, 0, 1\}$ can be converted into two binary-coded vectors, the same goes for ternary-coded weights.

In this paper, we propose a soft quantization approach to learn pure fixed-point representations of state of the art DNN architectures. Thereby, we follow the block structure and transfer all individual components into fixed-point representations before combining them appropriately. We follow the same approach as Enderich et al. (2019) and formulate bit-size dependent fixed-point constraints for each component before transferring these constraints into regularization terms. To the best of our knowledge, we are the first to provide a soft quantization approach to learn pure fixed-point representations of DNNs. We extensively validate our approach on several benchmark data sets and with state of the art DNN architectures. Although our approach is completely flexible in bit-size, we test two special cases:

- A pure fixed-point model with 4-bit weights and 4-bit activations which performs explicitly well, outperforming the floating-point baseline in many cases.
- A model with ternary-valued weights and 4-bit activations that can be evaluated using additions, bit shifts and clipping operations alone (no multiplications needed).

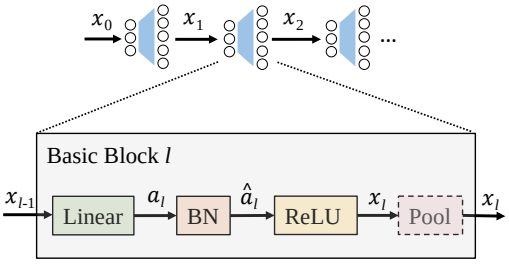

Figure 1: DNNs consist of interconnected blocks which combine a convolution or fully-connected layer (Linear), batch normalization (BN), ReLU, and pooling (optional).

Method	Linear	Act.	BN
Courbariaux et al. (2015)	1	32	32
Li & Liu (2016)	2	32	32
Hubara et al. (2016)	1	1	32
Uhlich et al. (2019)	2-4	4-32	32
Lin et al. (2016)	4	4-32	32
Yin et al. (2018)	2	32	32
Chen et al. (2017)	1	1	?
Enderich et al. (2019)	2	32	32
Achterhold et al. (2018)	2	32	32
Ours	2-4	2-4	2-4

Table 1: Survey of recent quantization methods and their bit-size configuration in different layers

2 Related Work

Considering that the optional pooling layer is of minor importance for quantization[2], three main components remain in each block: convolution and fully-connected layers (Linear), batch normalization (BN) and the non-linear activation function ReLU. Since each component differs in its computation task, different quantization strategies can be followed. Table 1 gives an overview of recent approaches including the respective bit-sizes during test time. Components encoded with 32 bits remain in high-precision floating-point arithmetic.

Courbariaux et al. (2015) use binarized weights during both forward and backward passes, but update their high-precision counterparts instead, which are kept during the whole optimization process. Thus, stochastic gradient descent is able to converge by doing small steps in the direction of the negative gradients. Li & Liu (2016) increased the model capacity by combining ternary-valued weights and a real-valued step-size. Since the step-size is a non-trainable parameter, its value is optimized by approximating the euclidean distance between the scaled ternary-weights and their high-precision counterparts. Hubara et al. (2016) amplifed the approach of Courbariaux et al. (2015) by discretizing both weights and activations to ± 1 during the forward pass. During the backward pass, the *straight-through estimator* (STE, Hinton (2012)) is used to estimate the local gradient of the rounding function. This way, the upstream gradient can be passed on during backpropagation.

Recently, approaches have been proposed that operate on fixed-point quantization functions with learnable function parameters. Jain et al. (2019) investigated signed quantization functions whose uniform step-size can be learned for a given number of bits. Uhlich et al. (2019) extended this approach and learned both step-size and dynamic range of symmetric and uniform quantization functions. Additionally, Uhlich et al. (2019) proposed optimization constraints to limit the overall

[2]Pooling reduces dimensionality in DNN propagation. While average-pooling was used mainly for historical reasons, max-pooling has recently gained acceptance due to better performance and practical reasons. However, both computations can also be done in fixed-point arithmetic.

memory costs.
Moreover, Enderich et al. (2019) proposed a soft quantization approach to train DNNs with multimodal fixed-point weights. Soft quantization means to use high-precision weights during the training, but simultaneously promote posterior distributions that are well qualified for post-quantization. Another soft quantization approach by Choi et al. (2018) investigated regularization terms for discrete activations. Furthermore, Achterhold et al. (2018) proposed a Bayesian method to train DNNs with quantizing priors that result in multimodal weight distributions.
However, high-precision BN layers are still a critical factor for success. The channel-wise floating-point multiplications within BN significantly increase the model capacity, especially for low-bit quantization, but at the same time eliminate the possibility of pure fixed-point arithmetic on dedicated hardware. Chen et al. (2017) introduced a hard quantization framework to completely train DNNs in low-bit fixed-point arithmetic, including fixed-point BN layers. But, Chen et al. (2017) focus only on computations and variables within the training procedure. Since BN layers operate on different statistics after training, the fixed-point representation during evaluation stays unclear. Furthermore, Chen et al. (2017) use 32-bit fixed-point multiplications which makes it impossible to fold the BN layer into the preceding Linear layer after quantization.

2.1 Core contributions

In this work, we complete the soft quantization approach and learn pure fixed-point representations on state of the art DNN architectures. We claim the following contributions:

- We follow the block structure in Figure 1 and formulate fixed-point constraints for each component that can be used during training. The fixed-point BN constraint is an especially novel and important feature in soft quantization. The resulting fixed-point block can be used in nearly all state of the art DNN architectures.
- We propose an exponentially increasing regularization parameter to control the model capacity immediately at training time. A set of clipping functions improves numerical stability and accelerates training time.
- We demonstrate that our fixed-point model outperforms other quantization approaches on common benchmark data sets and varying model sizes, from small- to large-scale.

3 Fixed-Point Representation

Fixed-point numbers consist of an N-bit - signed or unsigned - integer and a global scaling factor. The scaling factor is always a power of two whose exponent indicates the position of the decimal point: $Integer \times 2^{-f}$, $f \in \mathbb{Z}$. Thus, multiplications with powers of two result in bit shift operations, which can significantly accelerate computations on adequate fixed-point hardware (Hennessy & Patterson (2017)). In order to evaluate DNNs using pure fixed-point arithmetic, all individual layers must be converted into fixed-point representations and put together in a meaningful way. Depending on the layer type, different conditions must be fulfilled and we describe those conditions by several regularization terms R_i. In the end, the training objective is a composition such that the actual learning task, which is described by the cost function C, and the fixed-point constraints R_i are solved simultaneously during training:

$$C_{\text{total}} = C + \sum_i \lambda_i R_i \,. \tag{1}$$

3.1 Quantization functions

A quantization function maps an input signal x to a smaller set of discrete values $\widetilde{x}$. If certain properties apply, the quantization function can be expressed using basic operations like scaling, rounding, and clipping. Since different layer types deal with different value ranges, we use three

types of quantization functions:

$$\widetilde{x} = \begin{cases} Q_{\text{sym}}(x; N, \Delta) = \text{clip}\left(\left\lfloor \frac{x}{\Delta} \right\rceil, -2^{N-1}+1, 2^{N-1}-1\right)\Delta & \text{symmetric uniform,} \\ Q_{\text{uni}}(x; N, \Delta) = \text{clip}\left(\left\lfloor \frac{x}{\Delta} \right\rceil, 0, 2^{N}-1\right)\Delta & \text{unsigned uniform,} \\ Q_{\log}(x) = \text{sign}(x)\, 2^{\lfloor \log_2(|x|) \rceil} & \text{logarithmic.} \end{cases} \tag{2}$$

In this notation, $\lfloor \cdot \rceil$ rounds to the closest integer and clip(x, min, max) truncates all values to the domain [min, max]. The uniform quantization functions are parameterized by their uniform step-sizes Δ and the number of available bits N. Obviously, the fixed-point representation is fulfilled if and only if the step-size is a power of two, hence $\Delta = 2^{-f}, f \in \mathbb{Z}$. In this case, the scaling operation is replaced by a bit shift. Therefore, we use an additional logarithmic quantizer that rounds all input values to the closest power of two. That way, we are able to formulate appropriate fixed-point constraints for all important layers.

3.2 Convolution and fully-connected layers

Convolution and fully-connected layers are summarized as Linear components in Figure 1 since both perform weighted sums[3]. With x_{l-1} being the input of basic block l, the computation can be written as $a_l = x_{l-1} * w_l + b_l$, whereas $*$ is either a convolution operator or a matrix-vector multiplication, and $\{w_l, b_l\}$ is the set of parameters with w_l being either a weight-tensor or -matrix. Since additions play a subordinate role for complexity, we focus on weight multiplications. For this purpose, Enderich et al. (2019) recommend individual Gaussian priors to change the weight distribution from an unimodal distribution to a symmetric and multimodal distribution. The priors are induced by the L^2-norm and include the symmetric quantization function:

$$R_w = \sum_{l=1}^{L} \sum_{i=1}^{M_l} \frac{1}{M_l} \left\| w_{l,i} - Q_{\text{sym}}\left(w_{l,i}; N, \Delta_l\right) \right\|_2^2 , \tag{3}$$

where L is the number of layers, M_l the number of weights in layer l, and Δ_l the step-size in layer l. As recommended in Enderich et al. (2019), we use the layer-wise mean to give wide layers with many parameters a greater flexibility to compensate the quantization loss. Furthermore, we determine the layer-wise step-size on pre-trained weights by minimizing R_w under the constraint of $\Delta_l = 2^{-f}, f \in \mathbb{Z}$. In the next chapter, however, we see that the actual step-sizes are learned within the batch normalization component.
Effectively, R_w gives individual Gaussian priors to each network weight with respect to the closest fixed-point mode. The priors are updated with every forward pass, enabling the weights to continuously switch between neighboring modes. The gradient with respect to a single weight is

$$\frac{\partial R_w}{\partial w_{l,i}} = \frac{2}{M_l} \left(w_{l,i} - Q_{\text{int}}\left(w_{l,i}; N\right)\right) \left(1 - \frac{\partial Q_{\text{int}}\left(w_{l,i}; N\right)}{\partial w_{l,i}}\right) = \frac{2}{M_l} \left(w_{l,i} - \widetilde{w}_{l,i}\right). \tag{4}$$

Due to real-valued weights and a unique rounding function, the partial derivative $\partial Q_{\text{int}} / \partial w_{l,i}$ can be assumed to be zero. The final gradient is a scaled version of the corresponding quantization error. After training, the weights are quantized as follows

$$\widetilde{w}_{l,i} \leftarrow Q_{\text{sym}}\left(w_{l,i}; N, \Delta_l\right) . \tag{5}$$

3.3 Batch normalization

BN layers are located between convolution or fully-connected layers on one side and non-linear activation functions on the other. After the training, BN layers can be folded into the preceding layers to increase efficiency. Therefore, we first derive the BN fixed-point constraint before combining both layers to one coherent fixed-point module.

[3]Indeed, each convolution layer can be converted into a fully-connected layer by using the Toeplitz Matrix

3.3.1 FIXED-POINT CONSTRAINT

BN is performed channel-wise, with each channel being normalized and transformed linearly. The number of BN channels is equal to the number of output channels of the preceding convolution or fully-connected layer. If a denotes the BN input variable, the calculation is

$$\widehat{a}_c = \begin{cases} \dfrac{a_c - \mathrm{E}[a_c]}{\sqrt{\mathrm{Var}[a_c] + \epsilon}}\,\gamma_c + \beta_c & \text{while training,} \\ \dfrac{a_c - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}\,\gamma_c + \beta_c & \text{while evaluating,} \end{cases} \tag{6}$$

with c being the channel index, $\widehat{a}_c$ being the output variable, and $\{\gamma_c, \beta_c\}$ being the learnable affine transformation. During training, each channel is normalized using mean and variance of the current mini batch. At test time, normalization is done by the layer statistics $\{\mu_c, \sigma_c\}$, which have been continuously updated during training. Thus, each channel is first shifted and then multiplied with $\gamma_c / \sqrt{\sigma_c^2 + \epsilon}$, which can be turned into a bit-shift operation if the multiplier is a power of two. Since γ_c is the only learnable parameter in this expression, we propose the following regularization term:

$$R_\gamma = \sum_{l=1}^{L} \sum_{c=1}^{C_l} \left\| \frac{\gamma_{l,c}}{\sqrt{\sigma_{l,c}^2 + \epsilon}} - Q_{\log}\left(\frac{\gamma_{l,c}}{\sqrt{\sigma_{l,c}^2 + \epsilon}} \right) \right\|_2^2 , \tag{7}$$

where l is the layer index, L the number of layers, c the channel index, and C_l the number of channels in layer l. Thus, we utilize the L^2-norm to give individual fixed-point priors while taking into account that $\gamma_{l,c}$ is divided by the standard deviation after training. The corresponding gradient is

$$\frac{\partial R_\gamma}{\partial \gamma_{l,c}} = \frac{2}{\sqrt{\sigma_{l,c}^2 + \epsilon}} \left(\frac{\gamma_{l,c}}{\sqrt{\sigma_{l,c}^2 + \epsilon}} - Q_{\log}\left(\frac{\gamma_{l,c}}{\sqrt{\sigma_{l,c}^2 + \epsilon}} \right) \right) . \tag{8}$$

Travelling in the direction of the negative gradient optimizes $\gamma_{l,c}$ in the sense that, divided by the standard deviation, the closest power of two is approximated. In doing so, both the normalization statistics $\{\mu, \sigma^2\}$ and the affine transformation $\{\gamma, \beta\}$ can be adapted simultaneously such that the learning task and the fixed-point constraint are fulfilled. After training, BN parameters are quantized as follows:

$$\widetilde{\gamma}_{l,c} \leftarrow Q_{\log}\left(\frac{\gamma_{l,c}}{\sqrt{\sigma_{l,c}^2 + \epsilon}} \right) , \quad \widetilde{\sigma}_{l,c} \leftarrow 1 , \quad \widetilde{\epsilon} \leftarrow 0 . \tag{9}$$

3.3.2 FOLDING

Folding BN layers into the preceding Linear component reduces both computation time and memory capacity. After quantizing with Equation 9, the BN calculation simplifies to

$$\widehat{a}_{l,c} = (a_{l,c} - \mu_{l,c})\,\widetilde{\gamma}_{l,c} + \beta_{lc} . \tag{10}$$

Replacing the input variable by its calculation using the preceding Linear layer gives

$$\widehat{a}_{l,c} = (x_{l-1} * \widetilde{w}_{l,c} + b_{l,c} - \mu_{l,c})\,\widetilde{\gamma}_{l,c} + \beta_{lc} = x_{l-1} * \widetilde{w}_{l,c}\widetilde{\gamma}_{l,c} + (b_{l,c} - \mu_{l,c})\,\widetilde{\gamma}_{l,c} + \beta_{l,c} \tag{11}$$

where $\{\widetilde{w}_{l,c}\widetilde{\gamma}_{l,c}, (b_{l,c} - \mu_{l,c})\,\widetilde{\gamma}_{l,c} + \beta_{l,c}\}$ is the parameter set of the folded layer. Let us see if the fixed-point constraint is still fulfilled. After training and quantization, $\widetilde{w}_l$ consists of signed integers and a layer-wise step-size. By folding, the step-size is channel-wise multiplied with $\widetilde{\gamma}_{l,c}$. This turns the layer-dependent step-size into a channel-dependent step-size. Since both multipliers are powers of two, the newly created step-size is a power of two as well. Consequently, the BN fixed-point constraint enables to learn individual step-sizes that fulfill the fixed-point constraint.

3.4 RELU

ReLU is the state-of-the-art non-linear activation function in DNNs. In order to approximate the non-negative ReLU output, we use the uniform but unsigned quantization function Q_{uni} to quantize the network activations during each forward pass. During the backward pass, we utilize the STE to define the partial derivative of the rounding function as follows: $\partial \lfloor x \rceil / \partial x = 1$. That way, the local gradients with respect to the input and the step-size are as follows:

$$\frac{\partial Q_{\text{uni}}}{\partial \Delta_l} = \begin{cases} 0 & \text{if } x < 0 \\ 2^N - 1 & \text{if } x > \left(2^N - 1\right) \Delta_l \\ \left(x_q - x\right) / \Delta_l & \text{else} \end{cases}, \qquad \frac{\partial Q_{\text{uni}}}{\partial x} = \begin{cases} 0 & \text{if } x \leq 0 \\ 0 & \text{if } x > \left(2^N - 1\right) \Delta_l \\ 1 & \text{else} \end{cases}.$$

The gradient with respect to the input is passed on if x does not fall into the saturation bounds, otherwise it is set to zero. The gradient with respect to the step-size is non-zero if x is positive. Furthermore, it varies inside the range $[-0.5, 0.5]$ if x is within the range of the quantization steps. Otherwise, the gradient is equal to the highest N-bit integer.
In order to fulfill the fixed-point constraint of Q_{uni}, the step size has to be a power of two. Therefore, we use the logarithmic quantization function and propose the following regularization term

$$R_x = \sum_{l=1}^{L} \left\| \Delta_l - Q_{\log}\left(\Delta_l\right) \right\|_2^2 \quad \text{with the step-size gradient} \quad \frac{\partial R_x}{\Delta_l} = 2 \left(\Delta_l - \widetilde{\Delta}_l \right), \tag{12}$$

where l is the layer index, L is the amount of layers, and Δ_l the step size in layer l. The gradient is a scaled version of the quantization error and approximates the fixed-point constraint during training.

3.5 Regularization parameter

$$C_{\text{total}} = C + \lambda_w \, R_w + \lambda_\gamma \, R_\gamma + \lambda_x \, R_x \,, \tag{13}$$

is the training objective according to Equation 1, with C representing the learning task, $R_i, i \in \{w, \gamma, x\}$, being the particular fixed-point constraint, and λ_i being the corresponding regularization parameter which controls the weighting between learning task and quantization. In fact, regularization parameters add additional cost since their values must be determined empirically on a validation set. Then again, they allow an easy control of the model capacity, which is a fundamental problem in deep learning. On one hand, too much capacity often leads to overfitting. On the other hand, there must be sufficient capacity to enable optimal convergence, especially at the beginning of the training (Li et al. (2016)). Indeed, a training-time dependent regularization parameter can be used to control the model capacity immediately at training time. In this context, Enderich et al. (2019) recommend a linearly increasing regularization parameter that shifts the weighting towards the quantization constraint. However, we have found that exponential growth is better suited to change capacity. The calculation is

$$\lambda(e) = \lambda(0) \cdot \exp(\alpha_E e) \,, \tag{14}$$

where $\lambda(0)$ is the initial value, e denotes the current training epoch, and α_E is the growth factor which depends on the total number of training epochs E. In our experiments, we used the same configuration for each model on each data set: $\alpha_E = 10/E$, $\lambda_w(0) = 10$, $\lambda_\gamma(0) = \lambda_x(0) = 10^{-4}$. The values were determined on a CIFAR-10 validation set. Notice that the gradient $\partial R_w / \partial w$ from Equation 4 is divided by the number of layer weights. Therefore, the corresponding start value $\lambda_w(0)$ is accordingly higher.

3.6 Parameter clipping

Weight clipping: Soft quantization approaches train in high precision, but aim for posterior distributions that are well qualified for post quantization. These posterior distributions are promoted by suitable quantization constraints. In case of the Linear component, the fixed-point constraint limits the potential parameter space to the discrete values $\pm\Delta_l\left(2^{N-1} - 1\right)$. This can be utilized as prior knowledge since weights should not exceed this interval during training. For example, a weight quantization using $N = 2$ bits and $\Delta = 1$ leads to the possible quantization values $\{-1, 0, 1\}$. If a single weight already has the value -1, it is useless to optimize in the negative direction. Therefore we clip all Linear weights within $\left[-\Delta_l \left(2^{N-1} - 1\right), \Delta_l\left(2^{N-1} - 1\right)\right]$ after each update step to promote reasonable weight adaptation.

Step-size clipping: The physical limitation of quantization steps is to be strictly positive. Furthermore, very small step-sizes could cause numerical problems in the denominator. Therefore, we limit all quantization step-sizes to be $\geq 2^{-8}$ and clip the values after each update step, respectively.

Gradient clipping: Before the update step is done, the quantizing gradients from Equations 4, 8, and 12 are scaled by the corresponding regularization parameter. For numerical stability, we clip the scaled gradients to the absolute value of 0.1. This also prevents the regularization parameter from being too high.

4 EXPERIMENTS

In this section, we evaluate our fixed-point model on three common benchmark datasets: MNIST, CIFAR-10, and CIFAR-100. All experiments are done using stochastic gradient descent with a nesterov momentum of 0.9 and a linearly decreasing learning rate from 0.01 to 0.001. The batch size is 64. We compare our fixed-point model with all approaches from Table 1. Results are shown in Table 2. In order to provide a detailed comparison, we use two different bit-size configurations:

1. **Fix-Net**: A pure fixed-point model with 4-bit weights, 4-bit activations and bit-shift batch normalization. The performance is comparable to the floating-point baseline with all multiplications performed in fixed-point arithmetic.
2. **Add-Net**: A model with symmetric 2-bit weights (*ternary-valued*), 4-bit activations and bit-shift batch normalization. The evaluation during test time can be done without any multiplications. A simplified example of the computational graph is given in the appendix A.1. For the MNIST data set, we use 2-bit activations.

4.1 MNIST

MNIST is a handwritten-digits classification task. The dataset consists of 28×28 gray scale images and is divided into 60,000 training and 10,000 test samples (LeCun & Cortes (2010)). We use LeNet5 from Lecun et al. (1998) and preprocess the images by subtracting the mean and dividing by the standard-deviation over the training set.
Our Add-Net configuration achieves 0.65% test error after 40 epochs of training with 2-bit weights and activations. The result is similar to SGM and TWN, although both only quantize convolution and fully-connected layers. Our Fix-Net further decreases test error down to 0.59%. Both networks outperform the floating-point baseline of 0.71%.

4.2 CIFAR-10

CIFAR-10 is an image classification task with 10 different classes. The data consists of 32×32 RGB images and is divided into 50,000 training and 10,000 test samples (LeCun & Cortes (2010)). We preprocess the images as recommended in Huang et al. (2017). For detailed testing, we use three different model architectures: VGG7 from Simonyan & Zisserman (2015), DenseNet (L=76, k=12) from Huang et al. (2017) and ResNet20 from He et al. (2015). VGG7 is a conventional CNN architecture with 7 layers, BN and many parameters. In contrast, both DenseNet and ResNet20 show an efficient architecture with comparatively less parameters. Due to their lower number of redundancies, DenseNet and ResNet20 are considered as difficult to quantize.

VGG7: With an error rate of 6.22%, our Add-Net performs best among all models with binary- or ternary-valued weights. The performance is comparable to SGM which uses full-precision activations and batch normalization. With an error rate of 4.98%, our Fix-Net performs best in accuracy and even outperforms the floating-point baseline of 5.42% which proves its regularization characteristic. The bit-size configuration of Fix-Net is mostly comparable to Miyashita et al. (2016) and Lin et al. (2016) with error rates of 6.21% and 8.30%, respectively.

DenseNet: With the DenseNet architecture, Add-Net achieves 6.54% test error and outperforms the Bayesian approach of VNQ by more than 2%. SMG is slightly better with an error rate of 6.19% but quantizes only the Linear layers. The Fix-Net configuration achieves 5.63% test error and consequently beats the floating-point baseline of 5.72%.

ResNet: It is the smallest architecture in comparison, with only 0.28M parameters. Our Add-Net achieves an error rate of 10.13% which is 2% higher than the floating-point baseline. However, with only approximately 70kB memory costs and no multiplications, Add-Net achieves a significant reduction in complexity, even for small models. DQ performs slightly better, with an error rate

of 9.62% and floating-point BN layers. Our Fix-Net decreases test error down to 8.68% but still misses the floating-point baseline of 8.07%. Since ResNet20 already has a limited capacity, its regularization capability is also limited.

4.3 CIFAR-100

CIFAR-100 uses the same RGB images as CIFAR-10, but provides 10 additional sub-classes for each class in CIFAR-10. Thus, only 500 training samples are available for each of the 100 classes, which makes CIFAR-100 a challenging classification task. We use VGG11 from Simonyan & Zisserman (2015) and preprocess the images according to Huang et al. (2017).
Our Add-Net achieves 33.16% test error with at the same time lowest complexity. A visualization of the Add-Net weight distribution at different training times is given in Figure 3 in the appendix A.2. With an error rate of 30.25%, the Fix-Net configuration performs best in comparison and even outperforms the floating-point baseline of 31.42% by more than 1%.

5 Conclusion and further work

Soft quantization aims to reduce the complexity of DNNs at test time rather than at training time. Therefore, training remains in floating-point precision, but maintains consideration of dedicated quantization constraints. In this paper, we propose a novel soft quantization approach to learn pure fixed-point representations of state of the art DNN architectures. With exponentially increasing fixed-point priors and weight clipping, our approach provides self-reliant weight adaptation. In detailed experiments, we achieve new state of the art quantization results. Especially the combination of 4-bit weights, 4-bit activations and fixed-point batch normalization layers seems quite promising

Data set	Method	Model	Params	Bit-size (w/x)	Error [%]
MNIST	BNN, Hubara et al. (2016)	-	-	$1/1$	0.96
	VNQ, Achterhold et al. (2018)	LeNet5	60k	$2/32$	0.73
	TWN, Li & Liu (2016)	LeNet5	60k	$2/32$	0.65
	Add-Net	LeNet5	60k	$2/2$	**0.65**
	SGM, Enderich et al. (2019)	LeNet5	60k	$2/32$	0.63
	Fix-Net	LeNet5	60k	$4/4$	**0.59**
	Baseline	LeNet5	60k	$32/32$	0.71
CIFAR-10	FxpNet, Chen et al. (2017)	VGG7	9.3M	1/1	10.30
	Lin et al. (2016)	VGG7	3.4M	4/4	8.30
	TWN, Li & Liu (2016)	VGG7	12M	2/32	7.44
	SGM, Enderich et al. (2019)	VGG7	12M	2/32	6.27
	Miyashita et al. (2016)	VGG8	14M	4/5	6.21
	Add-Net	VGG7	12M	2/4	**6.22**
	Fix-Net	VGG7	12M	4/4	**4.98**
	Baseline	VGG7	12M	32/32	5.42
	VNQ, Achterhold et al. (2018)	DenseNet	0.49M	2/32	8.83
	Add-Net	DenseNet	0.49M	2/4	**6.54**
	SGM, Enderich et al. (2019)	DenseNet	0.49M	2/32	6.19
	Fix-Net	DenseNet	0.49M	4/4	**5.63**
	Baseline	DenseNet	0.49M	32/32	5.72
	Add-Net	ResNet20	0.28M	2/4	**10.13**
	DQ, Uhlich et al. (2019)	ResNet20	0.28M	2/4	9.62
	Fix-Net	ResNet20	0.28M	4/4	**8.68**
	Baseline	ResNet20	0.28M	32/32	8.07
CIFAR-100	TWN, Li & Liu (2016)	VGG11	32M	2/32	36.18
	BR, Yin et al. (2018)	VGG11	32M	2/32	34.13
	Add-Net	VGG11	32M	2/4	**33.16**
	Fix-Net	VGG11	32M	4/4	**30.25**
	Baseline	VGG11	32M	32/32	31.42

Table 2: Summary of the quantized performance on MNIST, CIFAR-10 and CIFAR-100.

REFERENCES

Jan Achterhold, Jan Mathias Koehler, Anke Schmeink, and Tim Genewein. Variational network quantization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=ry-TW-WAb.

Xiao Dong Chen, Xiaolin Hu, Hucheng Zhou, and Ningyi Xu. Fxpnet: Training a deep convolutional neural network in fixed-point representation. *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2494–2501, 2017.

Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. Learning low precision deep neural networks through regularization. *CoRR*, abs/1809.00095, 2018. URL http://arxiv.org/abs/1809.00095.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pp. 3123–3131, 2015.

Li Deng and Dong Yu. Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, 7(3–4):197–387, 2014. ISSN 1932-8346. doi: 10.1561/2000000039. URL http://dx.doi.org/10.1561/2000000039.

Lukas Enderich, Fabian Timm, Lars Rosenbaum, and Wolfram Burgard. Learning multimodal fixed-point weights using gradient descent. *European Symposium on Artificial Neural Networks*, 27, March 2019. URL http://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2019-69.pdf.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015.

John L. Hennessy and David A. Patterson. *Computer Architecture, Sixth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 6th edition, 2017. ISBN 0128119055, 9780128119051.

Geoffrey Hinton. Neural networks for machine learning. Coursera, video lecture, 2012.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 4107–4115. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6573-binarized-neural-networks.pdf.

Sambhav R. Jain, Albert Gural, Michael Wu, and Chris Dick. Trained uniform quantization for accurate and efficient neural network inference on fixed-point hardware. *CoRR*, abs/1903.08066, 2019. URL http://arxiv.org/abs/1903.08066.

A. Karki, C. Palangotu Keshava, S. Mysore Shivakumar, J. Skow, G. Madhukeshwar Hegde, and H. Jeon. Tango: A deep neural network benchmark suite for various accelerators. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 137–138, March 2019. doi: 10.1109/ISPASS.2019.00021.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. ISSN 0028-0836. doi: 10.1038/nature14539.

Fengfu Li and Bin Liu. Ternary weight networks. *CoRR*, abs/1605.04711, 2016. URL http://arxiv.org/abs/1605.04711.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *CoRR*, abs/1608.08710, 2016. URL http://arxiv.org/abs/1608.08710.

Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 2849–2858, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/linb16.html.

Daisuke Miyashita, Edward H. Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *CoRR*, abs/1603.01025, 2016. URL http://arxiv.org/abs/1603.01025.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015. URL http://arxiv.org/abs/1409.1556.

V. Sze, Y. Chen, T. Yang, and J. S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, Dec 2017. doi: 10.1109/JPROC.2017.2761740.

Stefan Uhlich, Lukas Mauch, Kazuki Yoshiyama, Fabien Cardinaux, Javier Alonso García, Stephen Tiedemann, Thomas Kemp, and Akira Nakamura. Differentiable quantization of deep neural networks. *CoRR*, abs/1905.11452, 2019. URL http://arxiv.org/abs/1905.11452.

Penghang Yin, Shuai Zhang, Jiancheng Lyu, Stanley Osher, Yingyong Qi, and Jack Xin. Binaryrelax: A relaxation approach for training deep neural networks with quantized weights. *SIAM Journal on Imaging Sciences*, 11, 2018.

A Appendix

A.1 Special Case: The Add-Net computation path.

The combination of symmetric 2-bit weights and bit-shift BN layers allows us to evaluate the Add-Net configuration without any multiplications. In Figure 2, a simplified example is given to demonstrate the computational path. As in the experimental chapter, we consider 4-bit quantized activations. The upper part in Figure 2 shows intermediate results in decimal notation whereas the lower part provides the corresponding binary code.
According to section 3.2 and 3.3, network weights split into a signed-integer part and channel-wise step-sizes after training. We simulate a weight matrix with three channels (or columns, respectively). In case of 2-bit weights, the signed integer part is filled with ternary values $\{-1, 0, 1\}$. Thus, multiply-accumulate operations - as they result from multiplication with the input $\widetilde{x}_{l-1}$ - can be performed using additions and subtractions, depending on whether the corresponding weight value is negative or positive. In Figure 2, the block is marked as *Fixed ADD*. Subsequently, each output channel is processed by its corresponding step-size. Since step-sizes are powers of two, f_l directly specifies the decimal shift. As you can see, only the decimal point is moved, respectively.
For simplification, the bias is assumed to be zero. For optimization purposes, we apply the ReLU computation in front of the quantization function and set values with an active signed-bit to be zero. Next comes the 4-bit activation quantization Q_{uni}. As mentioned already, we use a bit-size of $N = 4$. The step-size for the activations is assumed to be 2^{-2}. Therefore, the decimal point is initially shifted two positions to the right. In case there is an active bit to the right of the decimal point, round it up, if not, round it down. Now, the clipping module considers the N-bits to the left of the decimal point. If there is another active bit to the left, the value is clipped by activating all 4-bits in the blue section. In Figure 2, this is the case for the topmost example. Since all negative values

have been clipped by the ReLU operator, the decimal point is now shifted back and the quantization is finished.
Notice: This is only a semantic representation to demonstrate the bit shift mechanism in fixed-point arithmetic. Both the dimensions and the exemplary input values are arbitrarily selected.

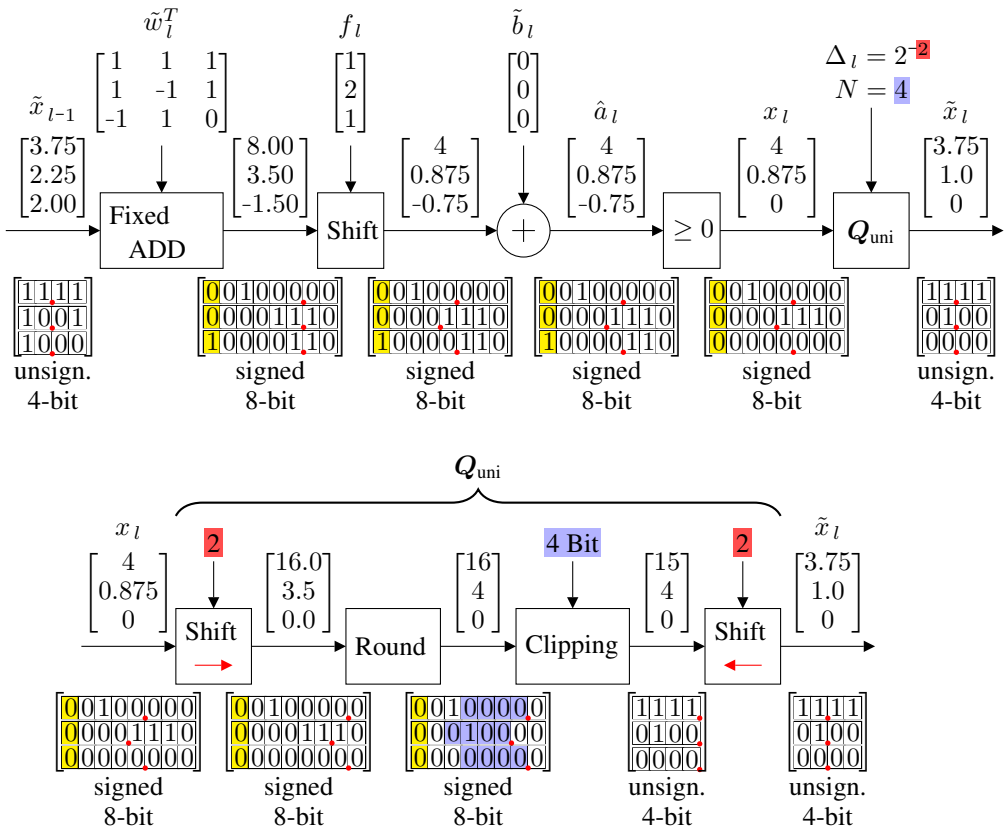

Figure 2: Clipping and saturation using bit-shifting and comparison.

A.2 Weight distribution of the VGG11 Add-Net

This section gives an insight in the training process of the VGG11 Add-Net on Cifar-100. Therefore, Figure 3 shows the weight distribution of Layer-1, Layer-4 and Layer-7 after several epochs of training. Since weight decay is used for pretraining, the initial weights resemble a unimodal distribution with a single peak at zero (epoch 0). At the start of training, two additional peaks arise at $\pm\Delta$ since layer weights are clipped to the particular quantization domain. Following this, the weights start to rearrange themselves taking into account both the fixed-point constraint and the learning task.

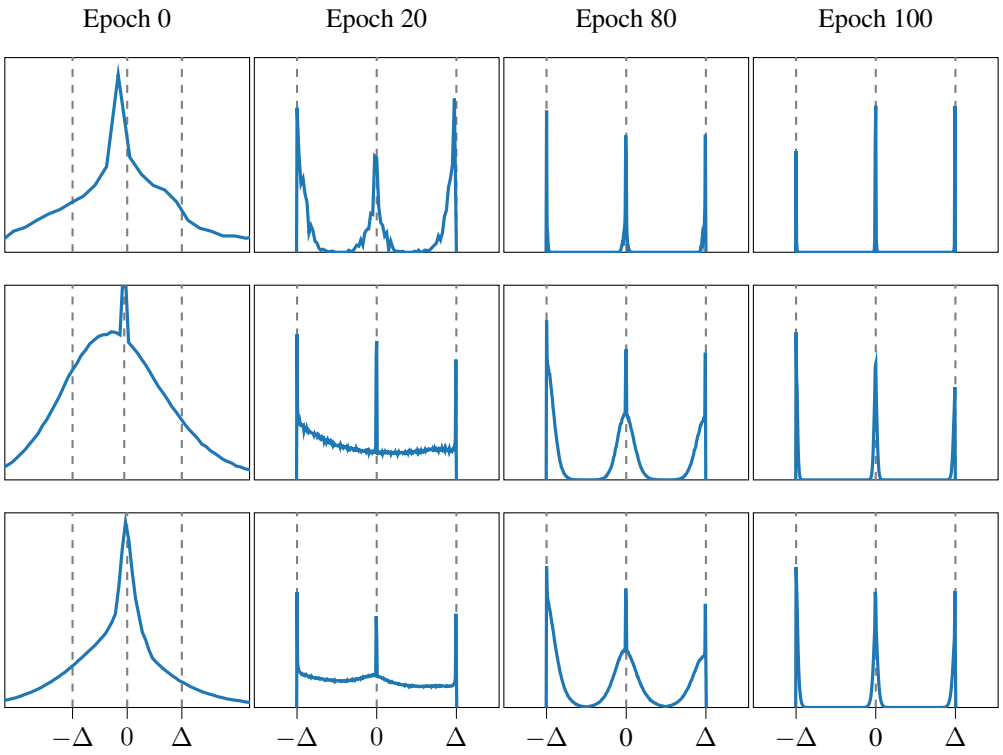

Figure 3: Weight distribution of Layer-1, Layer-4 and Layer-7 (from top to bottom) of VGG11 after several epochs. Since weight decay is used for pre-training, the weight distribution is unimodal at the beginning with a peak at zero. Then, our approach continuously rearranges the weights into a ternary-valued distribution, clearly visible at epoch 80. The variance of each mode is continuously decreased by the exponentially increasing regularization parameter. After 100 epochs, the weights are that close to the fixed-point centers that post quantization does not produce a remarkable error. Note: y-axis scaled individually for convenience, the x-axis for epoch 0 is wider to catch the whole distribution.