A FIXED-POINT QUANTIZATION TECHNIQUE FOR CONVOLUTIONAL NEURAL NETWORKS BASED ON WEIGHT SCALING

Norbert Mitschke, Michael Heizmann

Karlsruhe Institute of Technology, IIIT
Hertzstraße 16, 76187 Karlsruhe, Germany

Klaus-Henning Noffz, Ralf Wittmann

Silicon Software
K.-Zuse-Ring 28, 68163 Mannheim, Germany

ABSTRACT

In order to make convolutional neural networks (CNNs) usable on smaller or mobile devices, it is necessary to reduce the computing, energy and storage requirements of these networks. One can achieved this by a fixed-point quantization of weights and activations of a CNN, which are usually represented by 32-bit floating-point. In this paper, we present an adaption of convolutional and fully connected layers in order to obtain a high usage of the given value range of activations and weights. Therefore, we introduce scaling factors obtained by moving average to limit the weights and activations. Our model, quantized to 8 bit, outperforms the 7-layer baseline model from which it is derived and the naive quantization by several percentage points. Our method does not require any additional operations in the inference and both the weights and activations have a fixed radix point.

Index Terms— CNNs, Fixed Point Quantization, Image Processing, Machine Vision, Deep Learning

1. INTRODUCTION

Convolutional neural networks (CNNs) have become the state of the art technique for image classification in many domains. However, most CNNs require millions of MAC operations with at least half precision. This requires both a higher energy requirement due to multiplication and fetching and a larger storage capacity than a low-bit fixed-point arithmetic. According to [1], an 8-bit fixed-point multiplication consumes 18.5 times less energy, 27.5 times less space on a chip and only half the memory compared to half precision. Due to the enormously high number of multiplications and values to be stored, such a reduction in resource requirements is necessary to operate CNNs on small or battery-powered devices such as FPGAs or mobile phones.

1.1. Related Work

Quantization methods for CNNs can be divided into three major groups: the first group contains methods that quantize a CNN after training by determining optimal quantization levels. In general, this group requires look-up tables in the inference to restore the original weights from the low-bit quantization levels. In addition, the MAC operations are performed with floating point arithmetic. It can be shown, that mixed quantization according to [2] achieves better results than linear, probabilistic or random quantization. In [3], the optimal levels are calculated analytically. *Ristretto* [4] is an iterative algorithm that converts a floating-point to a dynamic fixed-point CNN. Here, the statistical amplitude distributions of weights and activations are analyzed and then the required bit length is determined. Another approach is the so-called *deep compression* [5, 6], where the weights are divided into k clusters, which reduce the memory requirements to $\log_2(k)$ bits per weight. Huffman coding can reduce the memory further.

In the second group, regularization terms are used during training to minimize the quantization error in the test phase. The quantization effects have to be taken into account while training the CNN in order to avoid a large loss during the test phase. In [7], a quantization scheme for MobileNets [8] is presented, where both quantization noise of the weights and too large activations are punished by a regularization term.

The methods of the last group use scaling factors during training to find an optimal value range for the parameters. In [9], dynamic fixed-point arithmetic is presented. Here, the bit length is given and a scaling factor s_t is fit adaptively determining the radix point for each layer. Therefore, two overflow rates are calculated, which represent the amount of related parameters which exceed the limits of the present quantization range and its half. Other algorithms use a greedy approach to find the optimal radix point [10]. In [11], an integer-only framework is proposed. In contrast to methods mentioned above, here the activations are also quantized. During training, 8-bit integers are mapped to real numbers in order to transform the matrix multiplications into integer multiplications. Therefore, additional scaling is necessary, since the weights are limited to $[-127, 127]$.

1.2. Contributions

Our approach is based on scaling factors like the methods of the third group. Not only do we consider the quantization of the activations required for an FPGA-based implementation,

but we also propose a scheme that avoids additional operations due to scaling (such as mapping). In the convolutional layers, we take advantage of the separability of the individual kernels by using a scaling factor per kernel to achieve a better mapping to the available value range.

2. BASIC CONSIDERATIONS

In this chapter we present the major aspects of CNNs for our approach, namely properties of the ReLU function with respect to scaling and quantization issues.

2.1. Scaling Parameters

Each neuron in a CNN uses a non-linearity $\sigma(\cdot)$, a weight vector $\vec{h}$ and a bias b to process an output y from its input vector $\vec{x}$:

$$y = \sigma(\vec{h} \cdot \vec{x} + b). \tag{1}$$

Using the ReLU function as non-linearity $\sigma(\cdot)$, it can be shown that

$$s \cdot y = s \cdot \sigma(\vec{h} \cdot \vec{x} + b) = \sigma(s \cdot \vec{h} \cdot \vec{x} + s \cdot b), \tag{2}$$

where $s \geq 0$ is a scaling factor corresponding to the neuron y. This applies because for $\vec{h} \cdot \vec{x} + b > 0$ the right side of eq. (1) is linear and else zero. Since a convolution can be written as matrix-vector-multiplaction, eq. (1) and eq. (2) are in principle also valid for convolutional layers. However, since several neurons share a weight vector during convolution, it is useful to apply one scaling factor per filter kernel in convolutional layers. From eq. (2) we get the 2-D activation map $\vec{Y}$ from the filter kernel $\vec{h}$, bias b and the input tensor $\vec{X}$ by:

$$s \cdot \vec{Y} = \sigma((s \cdot \vec{h}) \ ** \ \vec{X} + s \cdot b). \tag{3}$$

2.2. Quantization Issues

We have to consider both the limited value range and the granularity in order to minimize the losses caused by quantization. Outliers are problematic when quantizing in fixed-point arithmetic. This applies because a high ratio of maximum absolute amplitude to mean amplitude leads to a higher quantization error, since the maximum amplitude determines the value range implicitly. Another problem according to [7] is batch normalization, which is responsible for a high loss of accuracy after quantization.

For the further procedure we define the distance between two quantization steps $\Delta := 2^{-L_\mathrm{F}}$ and the Amplitude $A := 2^{L_\mathrm{I}}$, where L_F is the floating point bit length and L_I the integer bit length granted for the representation of the respective value. In this way we can represent signed values between $-\frac{A}{2}$ and $\frac{A}{2} - \Delta$. Unsigned values – these are the activations after a ReLU – are representable between 0 and $A - \Delta$.

3. PROPOSED METHOD

In contrast to other methods, we aim not to determine the value range of quantization in retrospect or adaptively. Our idea is to define the value range a-priori and to learn the weights in such a way that weights and activations lay within the given value range. Thus, the goal is to train the CNN in such a way that no unexpected losses occur due to quantization. Therefore, we modify the layers of the CNN by introducing scaling factors, using a new variant of ReLU and abandoning batch normalization.

3.1. Adapting Convolutional Layers

According to section 2.1, we extend the convolutional layer by a scaling factor $s_i > 0$ for each filter kernel $\vec{h}_i$. These factors are determined during training by moving average. The task of the scaling factors is to limit both the activation and the weights. With the designations from eq. (3) we get the three conditions

$$Y_\mathrm{max} := \max\{\sigma((s_i \cdot \vec{h}_i) * *\vec{X} + s_i \cdot b_i)\} \overset{!}{\leq} A_y - \Delta_y \tag{4}$$

$$h_\mathrm{max} := \max\{|s_i \cdot \vec{h}_i|\} \overset{!}{\leq} \frac{A_w}{2} - \Delta_w \tag{5}$$

$$b_\mathrm{max} := |s_i \cdot b_i| \overset{!}{\leq} \frac{A_w}{2} - \Delta_w, \tag{6}$$

where we determine A_w, A_y, Δ_w and Δ_y from the bit lenghts $L_{\mathrm{I},w}$, $L_{\mathrm{F},w}$, $L_{\mathrm{I},y}$ and $L_{\mathrm{F},y}$ as defined in section 2.2. The scaling factor is determined by moving average according to

$$\begin{aligned} &s_i \leftarrow m \cdot s_i + (1 - m) \cdot z_i \text{ with} \\ &z_i = \min\left\{ \frac{\alpha \cdot M_A}{Y_\mathrm{max} + \epsilon}; \frac{\alpha \cdot M_w}{\max\{h_\mathrm{max}; b_\mathrm{max}\} + \epsilon} \right\}, \end{aligned} \tag{7}$$

where m is the momentum, α is a buffer and ϵ is a summand for numerical stability. The amplitude of s_i is limited and initialized to s_max. To prevent the problem of outliers mentioned in 2.2 we use a L_2 regularizer.

At the end of the training phase, the scaling factors are multiplied to their corresponding kernels and biases as in eq. (3). The results are the weights to be quantized and used in the inference.

3.2. Adapting FC Layers

In order to comply with the specified range of weights $[-\frac{A_w}{2} + \Delta_w, \frac{A_w}{2} - \Delta_w]$, we limit the maximum amplitude of weights and biases by a constraint for FC layers.

To limit the activation, we use the activation function from fig. 1. During the training phase, we use soft clipping to prevent the gradient from vanishing. The ReLU becomes

$$\sigma(x) = 0.1x + 0.9 \cdot (A_y - \Delta_y) \tag{8}$$

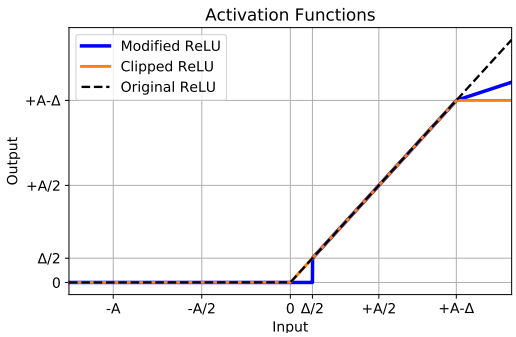

Fig. 1: ReLUs used in 1st FC layer: A hard clipping (orange) must be performed for the inference due to the limited bit length. Therefore a modified ReLU (blue) is used in the training instead of the original ReLU (black).

for $x > (A_y - \Delta_y)$. We set $\sigma(x) = 0$ for $x < \frac{\Delta}{2}$ to prevent too small activations. In the inference phase, we use hard clipping.

The output of the last FC layer is signed. Within the test phase the output is thus clipped at $-\frac{A_y}{2} + \Delta_y$ and $\frac{A_y}{2} - \Delta_y$ and the classification is done by argmax.

In the training phase, however, we need a softmax activation in order to calculate the crossentropy. For the same reason we use our modified ReLU, during training we thus apply between the last FC layer and softmax the function

$$\rho(x) = \begin{cases} 0.1x + 0.9 \cdot (\frac{A_y}{2} - \Delta_y) & \text{for } x \geq +\frac{A_y}{2} - \Delta_y \\ 0.1x - 0.9 \cdot (\frac{A_y}{2} - \Delta_y) & \text{for } x \leq -\frac{A_y}{2} + \Delta_y \\ x & \text{else} \end{cases}. \tag{9}$$

4. EXPERIMENTAL SETUP

In this section the four data sets we use, the baseline model and the training process are described.

4.1. Data Sets

The CIFAR-10 data set [12] consists of 60,000 RGB images of size 32×32 and represents an object classification task with 10 classes. The data set is split up in 50,000 images for training and 10,000 images for testing the classifier.

The second data set is the Street View House Numbers (SVHN) data set [13], which contains RGB images of real world digits. The size of the images is 32×32 and the data set is divided into 73,257 training and 26,032 test images.

The Fashion-MNIST data set [14] also represents an object classification task in which 70,000 fashion articles have to be distinguished between 10 different classes. The monochromatic images are upscaled to 32×32 pixels. The data set is composed of 60,000 training and 10,000 test images. We also

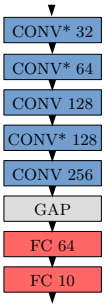

Fig. 2: The baseline model consists of five convolutional layers with 32, 64, 128 and 256 kernels. The layers marked with * are followed by max-pooling. We use a global averaging pooling layer and two FC layers with 64 and 10 neurons.

choose the MNIST data set [15], which represents a character recognition task. Upscaling and partioning of training and test data sets are chosen as for the Fashion-MNIST data set.

The images of all data sets are given with a pixel value p between 0 and 255 and are preprocessed by

$$p' = 2^{L_{\mathrm{F},y}-1} \cdot \frac{p - 128}{128}. \tag{10}$$

4.2. Baseline Model and Proposed Model

As baseline model we choose the CNN shown in fig. 2 with five convolutional and two FC layers. We use the normal ReLU activation function in the convolutional layers and max-pooling as shown in fig. 2. Both FC layers use dropout with a probability of 0.5. We use an L_2 regularizer with the parameter $\lambda_2 = 10^{-4}$.

The baseline model is compared to our modified model, where we adapt the layers of the baseline model as described in section 3. We set the integer bit length of the weights to $L_{\mathrm{I},w} = 2$ and $L_{\mathrm{F},w}$ to be variable, i. e. we get $A = 4$ and $\Delta = 2^{-L_{\mathrm{F},w}}$. For the activations we set $L_{\mathrm{I},y} = 3$ and $L_{\mathrm{F},y}$ variable as well. Empirically, we found for eq. (7) the momentum $m = 0.98$ and the buffer $\alpha = 0.94$. In order not to learn outliers of the scaling factors, we limit the maximum factor to 4 and ignore values z_i that are more than three times larger than s_i. To make the results comparable, we set the L_2 regularization parameter to $\lambda_2 = 10^{-4}$ in both models.

4.3. Training Process

We train both the baseline model and the adapted model for 90 epochs on the four data sets mentioned above. We use the *Nadam* optimizer [16] with an initial learning rate of $2 \cdot 10^{-3}$ and a batch size of 32. We use cyclic annealing according to [17] with three cycles. The algorithm is implemented in Keras (Tensorflow) [18].

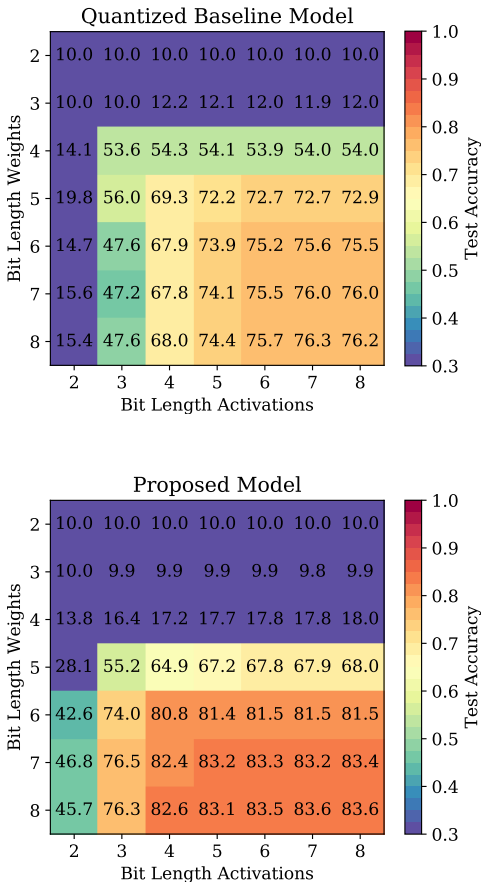

Fig. 3: CIFAR-10 test accuracy for different quantization levels in per cent.

After the training, the scaling factors are multiplied to the corresponding weights in our proposed model. The weights are then quantized in both models and the test accuracy is then determined. The activations are also quantized in the inference. Our results are then compared to the test accuracy of the baseline model with its 32-bit floating-point arithmetic.

5. RESULTS

The test accuracy on the CIFAR-10 data set for different quantization levels is illustrated in fig. 3. The 32-bit baseline accuracy is $79.52\,\%$. First, we find that with our quantized model, we can become more accurate than the 32-bit model. We assume that scaling has a normalizing effect on the activations similar to batch normalization. Qualitatively, we also find that our model has a higher robustness to the quantization of the activations, but is less robust to a coarse quantization of the weights. For example, the accuracy of our model at 4-bit weights is less than $20\,\%$, while it is about $50\,\%$ in the baseline model. It is noticeable that in our model at high weight

	32-Bit	Q. Baseline Model		Proposed Model	
CIFAR	79.52	67.96	76.20	82.63	**83.58**
SVHN	93.46	62.46	88.87	94.35	**94.84**
MNIST	99.42	93.21	99.29	99.41	**99.49**
Fashion	92.08	62.41	91.52	92.46	**92.93**

Table 1: The Table shows the test accuracy of the inference on the data sets from sec. 4.1. We compared the 32-bit baseline model to the quantized baseline model and our proposed model. For both quantized models the floating-point bit lengths are $L_{\mathrm{F},w} = 6$ and $L_{\mathrm{F},y} = 1$ (left column) and $L_{\mathrm{F},w} = 6$ and $L_{\mathrm{F},y} = 5$ (right column).

bit lengths there is no significant decrease in accuracy if the bit length of the activations goes back to four bits. Even at two bits, an accuracy up to $46.8\,\%$ can be achieved. These observations also apply to the other four data sets.

In the following we will consider an 8-bit quantization for the weights and a 4-bit or 8-bit quantization for the activations, since these are most suitable for the implementation. The test results are shown in table 1. The best result in each case is in bold. Our model with an 8-bit quantization always shows the best results. The reduction of the precision of the activations to 4 bits in the baseline model results in a significant degradation of the classifier for all four data sets, while the decrease in our model is marginal. We can also observe a relatively high loss of $2-3$ percentage points on the CIFAR and SVHN data sets when quantizing from 32-bit float to 8-bit fixed point. The loss in our model is much lower. The internal 32-bit inference of our proposed model reaches a test accuracy of $95.16\,\%$ before quantization on the SVHN data set, which is a loss of only 0.32 percentage points compared to the 8 bit inference.

The results should make it clear that our model is much better suited for training a low-bit inference than the generic approach. Unlike other methods, no additional operations in the inference are required and we achieve a normalizing effect by scaling. The results can still be improved by optimizing the topology, since we only use a seven layer CNN.

6. CONCLUSION

In this paper we presented a quantization approach for CNNs that reduces the accuracy loss due to quantization significantly compared to a baseline model. First we showed how to adapt the architecture in order to quantize a CNN efficiently. Therefore, we introduced scaling factors. Then we tested our method on four benchmark data sets and achieved a higher accuracy as the baseline model. Since we achieve quantization by scaling the weights after training, no additional arithmetic operations are required in the inference. This means that an implementation on an FPGA, for example, is more resource-efficient.

7. REFERENCES

[1] Mark Horowitz, "Computing's Energy Problem (and what we can do about it)," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 2014, pp. 10–14.

[2] Qiang Chen, Chen Xin, Chenglong Zou, Xinan Wang, and Bo Wang, "A Low Bit-Width Parameter Representation Method for Hardware-Oriented Convolution Neural Networks," in *IEEE 12th International Conference on ASIC (ASICON) 2017*. IEEE, 2017, pp. 148–151.

[3] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy, "Fixed Point Quantization of Deep Convolutional Networks," in *International Conference on Machine Learning*, 2016, pp. 2849–2858.

[4] Philipp Gysel, Mohammad Motamedi, and Soheil Ghiasi, "Hardware-Oriented Approximation of Convolutional Neural Networks," *arXiv preprint arXiv:1604.03168*, 2016.

[5] Song Han, Huizi Mao, and William J Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *arXiv preprint arXiv:1510.00149*, 2015.

[6] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, et al., "DSD: Dense-Sparse-Dense Training for Deep Neural Networks," *arXiv preprint arXiv:1607.04381*, 2016.

[7] Tao Sheng, Chen Feng, Shaojie Zhuo, Xiaopeng Zhang, Liang Shen, and Mickey Aleksic, "A Quantization-Friendly Separable Convolution for MobileNets," in *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*. IEEE, 2018, pp. 14–18.

[8] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *arXiv preprint arXiv:1704.04861*, 2017.

[9] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David, "Training Deep Neural Networks with Low Precision Multiplications," *arXiv preprint arXiv:1412.7024*, 2014.

[10] Kaiyuan Guo, Lingzhi Sui, Jiantao Qiu, Song Yao, Song Han, Yu Wang, and Huazhong Yang, "Angel-Eye: A Complete Design Flow for Mapping CNN onto Customized Hardware," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2016, pp. 24–29.

[11] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.

[13] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng, "Reading Digits in Natural Images with Unsupervised Feature Learning," in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011, vol. 2011, p. 5.

[14] Han Xiao, Kashif Rasul, and Roland Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," *CoRR*, vol. abs/1708.07747, 2017.

[15] Yann LeCun, Corinna Cortes, and CJ Burges, "MNIST Handwritten Digit Database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, 2010.

[16] Timothy Dozat, "Incorporating Nesterov Momentum into Adam," 2016.

[17] Jiawei Li, Tao Dai, Qingtao Tang, Yeli Xing, and Shu-Tao Xia, "Cyclic Annealing Training Convolutional Neural Networks for Image Classification with Noisy Labels," in *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2018, pp. 21–25.

[18] François Chollet et al., "Keras," https://keras.io, 2015.