Post-print of: A. Kwasniewska et al., "Deep Learning Optimization for Edge Devices: Analysis of Training Quantization Parameters," IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society, Lisbon, Portugal, 2019, pp. 96-101, doi: 10.1109/IECON.2019.8927153.

Deep Learning Optimization for Edge Devices: Analysis of Training Quantization Parameters

Alicja Kwasniewska
Artificial Intelligence Products Group
Intel Corporation
San Diego, USA
Dept. of Biomedical Engineering
Gdansk University of Technology
Gdansk, Poland
alicja.kwasniewska@intel.com

Maciej Szankin
Artificial Intelligence Products Group
Intel Corporation
San Diego, USA
maciej.szankin@intel.com

Mateusz Ozga
Artificial Intelligence Products Group
Intel Technology Poland
Gdansk, Poland
mateusz.ozga@intel.com

Jason Wolfe
Artificial Intelligence Products Group
Intel Corporation
San Diego, USA
jason.wolfe@intel.com

Arun Das
Artificial Intelligence Products Group
Intel Corporation
San Diego, USA
Dept. of Electrical and Computer Engineering, University of Texas
San Antonio, TX, USA
arun.das@utsa.edu

Adam Zajac
Artificial Intelligence Products Group
Intel Corporation
San Diego, USA
adam.zajac@intel.com

Jacek Ruminski
Dept. of Biomedical Engineering
Gdansk University of Technology
Gdansk, Poland
jacek.ruminski@pg.edu.pl

Paul Rad
Dept. of Electrical and Computer Engineering
University of Texas
San Antonio, TX, USA
peyman.najafirad@utsa.edu

***Abstract*—This paper focuses on convolution neural network quantization problem. The quantization has a distinct stage of data conversion from floating-point into integer-point numbers. In general, the process of quantization is associated with the reduction of the matrix dimension via limited precision of the numbers. However, the training and inference stages of deep learning neural network are limited by the space of the memory and a variety of factors including programming complexity and even reliability of the system. On the whole the process of quantization becomes more and more popular due to significant impact on performance and minimal accuracy loss. Various techniques for networks quantization have been already proposed, including quantization aware training and integer arithmetic-only inference. Yet, a detailed comparison of various quantization configurations, combining all proposed methods haven't been presented yet. This comparison is important to understand selection of quantization hyperparameters during training to optimize networks for inference while preserving their robustness. In this work, we perform in-depth analysis of parameters in the quantization aware training, the process of simulating precision loss in the forward pass by quantizing and dequantizing tensors. Specifically, we modify rounding modes, input preprocessing, output data signedness, bitwidth of the quantization and locations of precision loss simulation to evaluate how they affect accuracy of deep neural network aimed at performing efficient calculations on resource-constrained devices.**

***Keywords*—Deep neural network, quantization, quantization aware training, data analysis, artificial intelligence, edge devices**

This work has been partially supported by Intel Corporation USA, Intel Technology Poland, Statutory Funds of Electronics, Telecommunications and Informatics Faculty, Gdansk University of Technology and University of Texas, San Antonio, TX, USA. We thank all our colleagues, who provided insight and expertise that greatly assisted the research.

I. Introduction

Edge computing refers to the peripheral devices of control systems often set up on a local Ethernet network in industrial electronic systems. Many solutions benefit from extending human-machine interfaces with edge computing, e.g. by improving data reliability, speeding up data flow, performing system upgrades more efficiently, as edge devices can be plugged into existing controllers and tested locally and making decisions faster by processing data outside a centralized data center [1].

Due to recent technological advances such as reinvention of deep learning, increased computational capabilities and more powerful network technologies, perception abilities and quality of service have also increased. Various applications of deep learning at the edge have already been discussed, e.g. cache decision support [2], combining reinforcement learning, convolutional and recurrent Neural Networks (NN) with Information Centric Networking and Internet of Things [3], or reduce of a network traffic [4].

To make deep learning more efficient on resource-constrained devices, it is important to re-design model architectures. Optimization can be two-fold: 1) modification of a network structure to create more efficient NN, e.g. MobileNets that use depth-wise separable convolutions [5] or proper organization of convolutional filters in Fire modules as in SqueezeNet [6] 2) quantization introduced to constrain the weights and activation from floating point precision to a few discrete levels [7], [8]. It has been also shown that deep neural networks can be trained with quantized-inference in mind what improves the accuracy. B. Jacob et al. [9] proposed to simulate

precision loss in specific network parts during training to carry out more accurate integer-only arithmetic inference.

Although various studies have already evaluated accuracy of deep learning after quantization, including studies on different bit-precision [10], to the best of our knowledge there is no comprehensive study of different quantization algorithms, rounding parameters and various locations of the network where precision-loss is introduced. Case study research outlined in this work contributes to a more detailed understanding of how to quantize deep neural networks for inference on resource-constrained edge devices to reduce computational overhead while preserving high accuracy. Specifically, our contribution is as follows: 1) we consider stages necessary for reduction of the floating-point numbers of the deep neural network for a chosen object classification model i.e ResNet-50; 2) the extensive benchmark performed by us contains the analysis done with the quantization aware training to model quantization error present in integer-arithmetic only inference; 3) we evaluate various quantization parameters to provide a better insight of how to perform quantization on edge devices, widely used in smart home solutions and autonomous driving, as previously presented by us [11] [12] [13].

The next sections are structured as follows: Section II overviews state-of-the-art methods for Deep Neural Networks quantization. Section III provides the steps of simulating the precision loss during training and the quantization performed during inference. In addition, we present details about the performed extensive benchmark evaluation, including quantization parameters tested by us. Preliminary results are collected in Section IV and discussed in Section V. The work is summarized in the last section, providing conclusions and ideas for future applications of the presented methods.

II. Related work

A. Industrial applications of quantized inference on the edge

Facebook optimized performance and memory by reducing the precision to 8-bit integers from 32-bit floats before deploying neural networks to mobile platforms [14]. B. Jian et al. [15] showed how neural networks with limited precision weights (LPWNN) can be applied to digit recognition task reducing processing time by an order of magnitude on ARM 9 embedded system. Another computer vision application proved that object detection from live video stream can be performed in real-time on the embedded Zynq UltraScale+ (XCZU3EG) device by using different levels of quantization [16]. Significant reduction of convolutional neural network parameters has been achieved by introducing weight quantization with a novel low rank sparse quantization (LRSQ) [17]. LRSQ method has been evaluated on various convolutional neural networks resulting in a very small accuracy drop ($< 1\%$) comparing to the full precision mode. The study conducted by Wang Y. et al. proved 40x energy efficient gain by using RRAM-based CNN with 8-bit weights [18].

B. Quantization techniques applied to the training

Many authors consider methods that are generalization of simple quantization method, and obtain algorithms for finding the optimal number of precision. For example, Gupta et al. [19] proposed limited precision data of large-scale deep learning neural network for a training. They have shown that deep learning neural network might be trained using stochastic rounding number with 16-bit wide fixed-point representation of the number. Fixed-point methods with stochastic-rounding number can be also implemented on an energy-efficient hardware [19]. In light of more modern quantization methods, Kster et al. [20] proposed to represent data as flex-point number and replace 32-bit floating point format both for training and inference. Kim and Smaragdis [21] demonstrated that the quantization methods can be used in bitwise neural network. Moreover it has been show that bitwise network can be trained with binary format of the weight tensors and then modified to work with full precision digits [21]. Miyashita, et al. [22] used base-2 logarithmic representation of digits at 5-bits for weights and activation for end-to-end training. The achieved final accuracy was higher than linear representation at 5-bits. Instead of direct weight quantization, it also common to retrain the model with lower precision, as proposed by [23] and [24].

Networks quantization can lead to significant memory and processing power reduction. Quantization aware training is important to achieve higher accuracy during inference, as it models quantization error during training to match quantization effects during the inference [9]. Quantization awareness is performed by introducing so-called fake quantizations, which means that quantization is directly followed by dequantization. In this way, we simulate precision loss that is later observed in arithmetic-only inference. In this study, we focus only on the training step, as inference quantization depends heavily on the hardware specification. Proper quantization should ensure that quantization errors and fusions at inference time are accurately modeled at the training time. Although training with simulated precision loss have been already proposed, to the best of our knowledge there is no existing in-depth comparison of parameters selection for quantization-aware training. This work aims at providing evaluation of these parameters and determining how they affect network accuracy to ease inference quantization design decisions.

III. Methodology

In this study, we model quantization error in the forward pass using simulated precision loss, while full precision is used for the back propagation to preserve small gradient updates, similarly to [9]. Yet contrary to previous studies, we compare different fake quantization operations available in the TensorFlow framework and perform experiments for various locations of these ops. In addition, we evaluate precision and recall of models trained using different quantization parameters. Analysis is performed for the Resnet v1 50 model [25] trained on the Cifar10 [26] set, that contains 6k 32x32 RGB images (5k train and 1k test sets) divided into 10 categories.

A. Training procedure

All variants of the model were trained using TensorFlow models repository [27]. For a fair comparison, we were changing only quantization parameters, other network hypermarmeters remained constant and were set to their default values: weight decay 0.00004, optimizer RMSProp, optimizer epsilon 1.0, momentum 0.9, learning rate 0.01, learning rate decay exponential down to 0.0001 every 2 epochs, batch size 8. Number of optimal training steps was determined using early stopping method on the not-quantized model. As soon as accuracy on the validation set stopped improving, the training was stopped. The determined optimal number of training step was 350000.

In our experiments we followed one-factor-at-a-time method (OFAT) modifying one quantization parameter at a time instead of multiple variables simultaneously due to limited training resources. Yet, in future work we want to perform similar experiments while changing multiple factors at once in order to estimate possible interactions between quantization parameters.

To perform OFAT experiments, we set all quantization parameters defined in following subsections to their default values in the first run, and then we were changing each of them at a time, training ResNet model again with new configurations. After training, all models were evaluated on the validation set producing accuracy and recall metrics.

B. Simulating precision loss

To simulate precision loss, each input v is at first quantized and then dequantized back to *fp32* output O as follows:

$$O = \frac{r_{m|rhe,rhu}(c(v, \min_{in}, \max_{in}) * s_f)}{s_f}. \tag{1}$$

where: min_{in} and max_{in} are minimum and maximum values determined from the input or specified manually; v, c is a clamping operation; s_f is the scaling factor and r_m is a rounding mode. All parameters are described in details in following subsections.

C. Modified quantization parameters

1) Operations in TensorFlow for precision loss simulation: TensorFlow supports various two main operations for simulating precision loss: *FakeQuantWithMinMaxVars* and *QuantizeAndDequantizeV2*. The main difference is that quantization range in *FakeQuantWithMinMaxVars* is defined as unsigned ($[0; 2^{n_{bits}} - 1]$) or $[1; 2^{n_{bits}} - 1]$), while in QuantizeAndDequantizeV2 output signedness can be specified. For detailed differences in implementation please check TensorFlow documentation [28].

2) Locations of precision loss simulations: Quantization aware training described in [9] proposes a quantization scheme that quantizes convolution weights and activations. We follow this approach and perform experiments using the same locations for simulated precision loss (default placement is *Relu* for activations and *VariableV2, Identity* for weights). Yet, contrary to the proposed quantization scheme, we also quantize inputs to *Add* operation, as presented in Fig. 1 This modification was motivated by the fact that skip connection present in ResNet may operate within different scale than the main branch, therefore to take both branches into account equally, ranges should be updated before adding them. Inputs and outputs of deep models have proven sensitive to quantization [15], therefore the full precision for these operations is preserved.

3) Scale s_f: Given a set of n points in the range of the float precision, $a^{(1)}, ...a^{(n)} \in fp32$, find n numbers that are maximum or minimum value of the matrix, as follows:

$$\min_{\forall x \in \mathbb{R}^2} f(x) \quad \text{and} \quad \max_{\forall x \in \mathbb{R}^2} f(x). \tag{2}$$

$f(x)$ is defined to be: $f(x) = \{x : -128 \leq x \leq 127 \; x \in \mathbb{R}^s\}$ and $f(x) = \{x : 0 \leq x \leq 255 \; x \in \mathbb{R}^u\}$ for output bitwidth set to 8 bits, where: $\mathbb{R}^s$ and $\mathbb{R}^u$ are for signed and unsigned values of the input. These min_{in} and max_{in} values could be controlled by the use of the special TensorFlow flag, i.e range given. If this flag is set on $False$ the values will be measured automatically from the input tensor (by default range given is set to False), otherwise min_{in} and max_{in} values will be determined through these parameters.

Consequently, if $O \in \mathbb{R}^s$ and the number of bits is equal 8 then s_f might be defined as $\frac{-128}{min_{in}}$ and $\frac{+127}{max_{in}}$. Otherwise, if $O \in \mathbb{R}^u$ then s_f will be determined as follows: the $min_{in} = 0$ and only max_{in} is used.

Additionally, the min_{in} and max_{in} values for activation are determined during training using moving average.

4) Rounding mode r_m: Clearly, these methods allow to modify the value of the O tensor by using popular rounding-number modes. There are differentiate stochastic and non-stochastic methods e.g. round half to even, half up.

1) Round half to even rhe - That is default tie-brekaing rule of r_m. This rule defined output round value without negative or positive bias and without bias away toward from zero. The primary goal of this method is to obtain minimization of the expected error for an activation and convolution operations:

$$r_{m|rhe} = \begin{cases} \lfloor x \rfloor & \text{if } x < 0 \text{ and } x \in \mathbb{R}, \\ \lceil x \rceil & \text{otherwise.} \end{cases} \tag{3}$$

2) Round half up rhu - That is rule broadly used to round always values up:

$$r_{m|rhu} = \lceil x \rceil \quad \text{for all} \quad x \in \mathbb{R}. \tag{4}$$

5) Input and output data: Signedness of the output from the fake quantization operation can be specified if *QuantizeAndDequantizeV2* is used. Output signedness determines calculation of scale factor s_f. If output is unsigned, the minimum is set to 0 and only maximum value is used. By default output is signed, but we perform additional experiments for unsigned outptut as well.

In addition, we also modify image preprocessing step by using both *vgg* and *inception* methods, as defined in [27]. In *vgg* preprocessing (default setting), given mean values

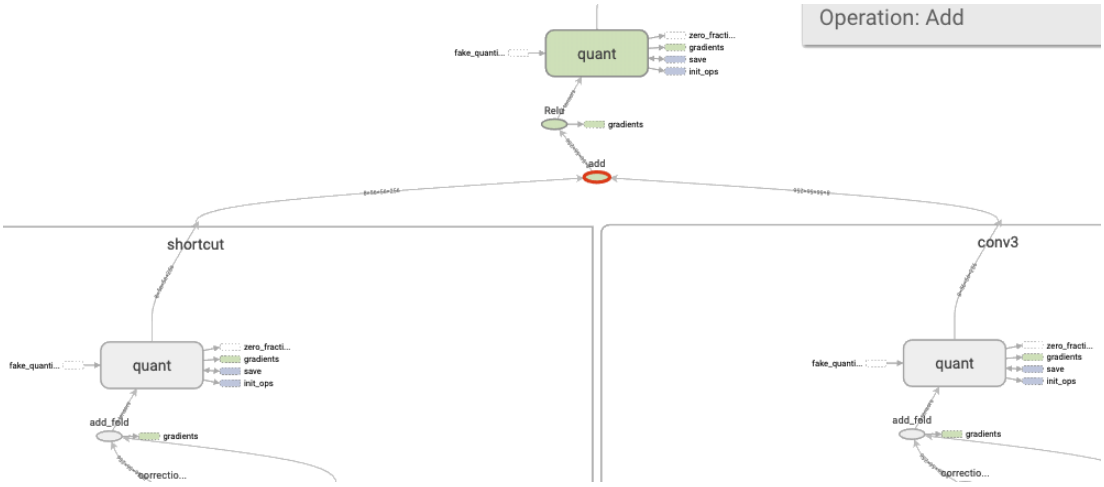

Fig. 1: Part of the quantized ResNet v1 50 TensorFlow graph - fake quantization operation inserted on inputs to *Add* operation

are subtracted from each image channel, but input is not normalized. In *inception* preprocessing, input data is rescaled to the range $[-1, 1]$.

6) Bitwidth of the quantization bt: Bitwidth parameter specifies the amount of data (measured in bits) used to represent the output of the quantizer. By default bitwidth is set to 8 bits, but can be set to any value that is a power of 2. Specified bitwidth in the output determines minimum and maximum of quantized values:

$$bt = \begin{cases} [0; 2^{n_{bits}} - 1] & \text{if } \textit{FakeQuant}, \\ [-(2^{n_{bits}-1}); 2^{n_{bits}-1} - 1] & \text{if } \textit{QDQv2} \text{ and } O^s, \\ [0; 2^{n_{bits}} - 1] & \text{if } \textit{QDQv2} \text{ and } O^u. \end{cases} \quad (5)$$

where O^s means that output is signed and O^u means that the output is unsigned.

IV. Results

Table I presents the accuracy and recall scores achieved by baseline model and models with different quantization parameters on the CIFAR10 test set. Plots of training loss for different quantization delay values, fake quantization operations, minimum and maximum values setup, and different placement of simulated precision loss are presented in Fig. 2, 3, 4, 5, respectively.

V. Discussion

The preliminary studies of the experiments described in this work show the impact of quantization of deep learning models on the accuracy and recall metrics. The experiments were divided into 9 categories, each representing separate quantization parameter with it's different options tested within the category.

As a baseline to our experiments, the ResNet50 model was trained for 350.000 iterations on the CIFAR10 data set with default configuration and without quantization operations. Quantized models used in the experiments described below were trained using the same set of parameters as the baseline model, with the exception of the quantization configuration.

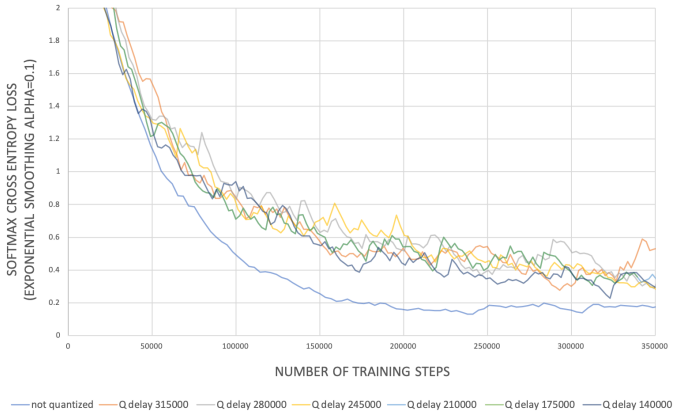

Fig. 2: Softmax cross entropy loss for different values of quantization delay

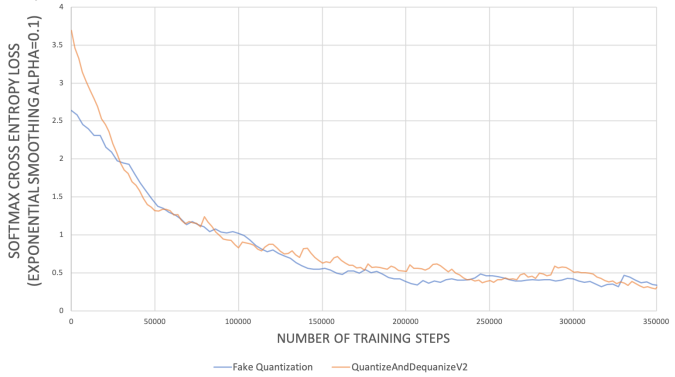

Fig. 3: Softmax cross entropy loss for different simulate precision loss TensorFlow operations

The baseline model achieved accuracy and recall of 91.14% and 99.42%, respectively.

Comparing the average accuracy and recall that were achieved by the quantized models with the same metrics scored by the not quantized model, quantized models achieved accuracy of $86.15 \pm 10.54\%$ and recall of $99.04 \pm 1.44\%$, showing that there is no significant degradation in these metrics when compared to the original model.

In the first experiment we compared original model with two different quantization operations implemented in the

TABLE I: Accuracy [%] and Recall [%] comparison of the baseline not-quantized model against models quantized with different quantization parameters. Quantization operations were inserted on inputs to ops of type Conv2D and Add; and on outputs of activation operations.

Accuracy	Recall	Quantized	Quant. op	Rounding mode	Bitwidth	Input preprocessing	Output signedness	Locations of quantization op	Range given	Quantize delay [% steps]	Theoratical Inference Model Size
91.14	99.42	×	-	-	-	-	-	-	-	-	89.6 MB
52.90	94.51	✓	QDQv2	half_to_even	4	vgg	signed	Conv, Activation	×	80%	11.42 MB
86.27	99.22	✓	FakeQuant	half_to_even	8	vgg	signed	Conv, Activation	×	80%	22.95 MB
86.60	99.29	✓	QDQv2	half_to_even	8	inception	signed	Conv, Activation	×	80%	22.95 MB
88.06	98.99	✓	QDQv2	half_to_even	8	vgg	signed	Conv, Activation	✓	80%	22.95 MB
88.65	99.10	✓	QDQv2	half_to_even	8	vgg	unsigned	Conv, Activation	×	80%	22.95 MB
88.69	99.61	✓	QDQv2	half_to_even	8	vgg	signed	Conv, Activation	×	60%	22.95 MB
88.82	99.63	✓	QDQv2	half_to_even	8	vgg	signed	Conv, Activation	×	90%	22.95 MB
88.86	99.01	✓	QDQv2	half_up	8	vgg	signed	Conv, Activation	×	80%	22.95 MB
88.87	99.28	✓	QDQv2	half_to_even	8	vgg	signed	Conv, Activation, Add	×	80%	22.95 MB
89.06	99.64	✓	QDQv2	half_to_even	8	vgg	signed	Conv, Activation	×	70%	22.95 MB
89.37	99.65	✓	QDQv2	half_to_even	8	vgg	signed	Conv, Activation	×	50%	22.95 MB
89.69	99.66	✓	QDQv2	half_to_even	8	vgg	signed	Conv, Activation	×	80%	22.95 MB
90.04	99.67	✓	QDQv2	half_to_even	8	vgg	signed	Conv, Activation	×	40%	22.95 MB
90.71	99.25	✓	QDQv2	half_to_even	32	vgg	signed	Conv, Activation	×	80%	89.6 MB
91.26	99.34	✓	QDQv2	half_to_even	16	vgg	signed	Conv, Activation	×	80%	45.90 MB

Fig. 4: Softmax cross entropy loss for different ways of minimum and maximum values selection

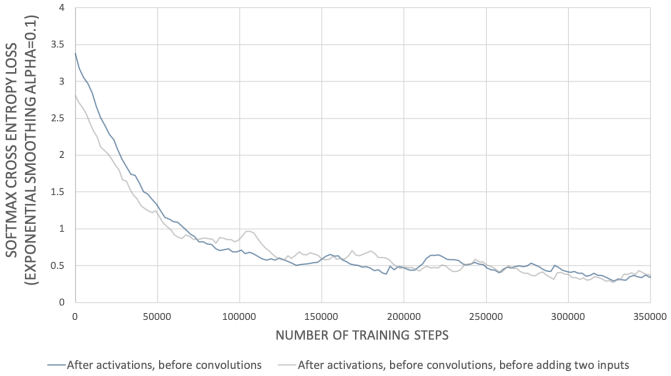

Fig. 5: Softmax cross entropy loss for different placement of simulated precision loss

TensorFlow framework - *FakeQuantWithMinMaxVars* (*FakeQuant*) and *QuantizeAndDequantizeV2* (*QDQv2*). The results produced by both of them are below the score achieved by the original model, however, in case of the *QDQv2* operation, it was only by a small margin (1.45%). For *FakeQuant* the accuracy decreased by 5.34% comparing to the not quantized model.

Default half to even rounding mode has been proved to produce slightly better evaluation metrics, however it was only used with bitwidth of 8. It may turn out that rounding of lower precision values would be better with different rounding modes. We will perform these experiments in future studies.

We found the theoratical inference model size by utilizing *TensorFlow Graph Transform Tool* for all bitwidths considered. Accuracy-model size tradeoff showed that 8 bit quantized models performed well with little loss in accuracy while providing model compression by a factor of four.

In another experiment models with different numerical precision were tested. No significant changes in the accuracy and recall were observed, with the exception of the model trained with 4bit precision, which achieved the lowest scores in both metrics (accuracy decrease of 41.96%).

Next test investigated the impact of starting the quantization in different points of the training process (40%, 50%, 60%, 70%, 80% and 90% of training iterations). The results show that starting the quantization earlier is beneficial for the performance of the model, as the highest accuracy was achieved by the model that began quantizing after 40% of the iterations, while models with larger delays trended toward lower scores.

In another test, the choice of minimum and maximum values for calculatitng quantization scaling was compared. It's not surprising that minimum and maximum values determined automatically from the inputs lead to better accuracy and recall results, as the range is better adjusted to data on which the graph operates.

Analysis performed for placement of simulated precision loss didn't prove accuracy improvement for ranges adjusted before adding skip connection to the main ResNet branch. This may be caused by the fact that minimum and maximum values of inserted *QDQv2* ops were not tied together, so both branches might have still operated in different ranges. To solve this problem, in future work we want to ensure that all operations that have more than 1 parameter (e.g. *Add, Concat* update scales of these parameters to the same values to preserve equal contribution of inputs.

Also, the default TensorFlow preprocessing method for ResNet model (*vgg*) has turned out to be better than *inception* preprocessing that normalizes input data to $[-1, 1]$, resulting

in accuracy increase of 3.45%.

The performed extensive benchmark provides in-depth comparison of various parameters in quantization aware training applied to convolutional neural networks and can be used as a base for designing various deep learning based computer vision tasks. As a result, industrial applications can be deployed on resource-constrained edge devices without reveling privacy of data by sharing it in the cloud. Yet, it is worth mentioning that achieved results are only preliminary and experiments should be further extended in future work.

VI. Conclusion

This work focused on evaluation of quantization parameters for precision loss simulation during deep neural network training. Performed experiments showed that it's important to cautiously select quantization parameters in order to achieve the smallest accuracy drop comparing to non-quantized model. In future work, we will perform similar experiments while changing multiple factors at once, to determine whether there are interactions between quantization parameters. What's more, we will provide a mechanism to tie ranges of all inputs to a given operation to make sure they equally contribute to the output. Also, we would like to apply quantization aware training to deep neural networks utilized by us for e.g. image classification [11] or gesture recognition [29] in smart home and smart building environments on mobile platforms. In this way, data can be processed locally to preserve privacy, e.g. in solutions designed for remote vital signs monitoring [30].

References

[1] P. Garcia Lopez, A. Montresor, D. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. Barcellos, P. Felber, and E. Riviere, "Edge-centric computing: Vision and challenges," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 37–42, 2015.

[2] K. Thar, N. H. Tran, T. Z. Oo, and C. S. Hong, "Deepmec: Mobile edge caching using deep learning," *IEEE Access*, vol. 6, pp. 78 260–78 275, 2018.

[3] H. Khelifi, S. Luo, B. Nour, A. Sellami, H. Moungla, S. H. Ahmed, and M. Guizani, "Bringing deep learning at the edge of information-centric internet of things," *IEEE Communications Letters*, vol. 23, no. 1, pp. 52–55, 2019.

[4] Y. Huang, X. Ma, X. Fan, J. Liu, and W. Gong, "When deep learning meets edge computing," in *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. IEEE, 2017, pp. 1–2.

[5] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[6] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[7] R. Ding, Z. Liu, R. Blanton, and D. Marculescu, "Quantized deep neural networks for energy efficient hardware-based inference," in *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*. IEEE Press, 2018, pp. 1–8.

[8] F. Tung and G. Mori, "Deep neural network compression by in-parallel pruning-quantization," *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[9] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[10] S. Hashemi, N. Anthony, H. Tann, R. I. Bahar, and S. Reda, "Understanding the impact of precision quantization on the accuracy and energy of neural networks," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1474–1479.

[11] M. Szankin, A. Kwasniewska, T. Sirlapu, M. Wang, J. Ruminski, R. Nicolas, and M. Bartscherer, "Long distance vital signs monitoring with person identification for smart home solutions," in *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2018, pp. 1558–1561.

[12] M. Szankin, A. Kwaśniewska, J. Ruminski, and R. Nicolas, "Road condition evaluation using fusion of multiple deep models on always-on vision processor," in *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2018, pp. 3273–3279.

[13] L. Tang, H. Subramony, W. Chen, J. Ha, H. Moustafa, T. Sirlapu, G. Deshpande, and A. Kwasniewska, "Edge assisted efficient data annotation for realtime video big data," in *IECON 2018-44th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2018, pp. 6197–6201.

[14] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. M. Hazelwood, E. Isaac, Y. Jia, B. Jia *et al.*, "Machine learning at facebook: Understanding inference at the edge." in *HPCA*, 2019, pp. 331–344.

[15] B. Jian, C. Yu, and Y. Jinshou, "Neural networks with limited precision weights and its application in embedded systems," in *2010 Second International Workshop on Education Technology and Computer Science*, vol. 1. IEEE, 2010, pp. 86–91.

[16] T. B. Preußer, G. Gambardella, N. Fraser, and M. Blott, "Inference of quantized neural networks on heterogeneous all-programmable devices," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 833–838.

[17] X. Long, Z. Ben, X. Zeng, Y. Liu, M. Zhang, and D. Zhou, "Learning sparse convolutional neural network via quantization with low rank regularization," *IEEE Access*, vol. 7, pp. 51 866–51 876, 2019.

[18] Y. Wang, L. Xia, T. Tang, B. Li, S. Yao, M. Cheng, and H. Yang, "Low power convolutional neural networks on a chip," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2016, pp. 129–132.

[19] K. G. P. N. Suyog Gupta, Ankur Agrawal, "Deep learning with limited numerical precision," in *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2015.

[20] U. Köster, T. Webb, X. Wang, M. Nassar, A. K. Bansal, W. Constable, O. Elibol, S. Gray, S. Hall, L. Hornof *et al.*, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," in *Advances in neural information processing systems*, 2017, pp. 1742–1752.

[21] P. S. Minje Kim, "Bitwise neural networks," in *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2016.

[22] B. M. Daisuke Miyashita, Edward H. Lee, "Convolutional neural networks using logarithmic data representation," in *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2016.

[23] H. C. E. Fiesler, A. Choudry, "A weight discretization paradigm for optical neural networks," in *12-16 April. International Society for Optics and Photonics*. IEEE, 1990, pp. 164–173.

[24] C. Tang and H. Kwan, "Multilayer feedforward neural networks with single powers-of-two weights,(1993) ieee transactions on signal processing, volume 41, number 8," 1993.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition. eprint," *arXiv preprint arXiv:0706.1234*, 2015.

[26] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[27] "Tensorflow models repository," https://github.com/tensorflow/models, accessed: 2019-04-15.

[28] "Tensorflow documentation," https://www.tensorflow.org/api_docs/python/tf, accessed: 2019-04-15.

[29] K. Czuszynski, A. Kwasniewska, M. Szankin, and J. Ruminski, "Optical sensor based gestures inference using recurrent neural network in mobile conditions," in *2018 11th International Conference on Human System Interaction (HSI)*. IEEE, 2018, pp. 101–106.

[30] J. Ruminski and A. Kwasniewska, "Evaluation of respiration rate using thermal imaging in mobile conditions," in *Application of Infrared to Biomedical Sciences*. Springer, 2017, pp. 311–346.