INT8 Activation Ternary or Binary Weights Networks

Ninnart Fuengfusin
Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology,
2-4 Hibikino, Wakamatsu-ku, Kitakyushu, Fukuoka, 808-0196, Japan
Hakaru Tamukoh
Research Center for Neuromorphic AI Hardware,
Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology,
2-4 Hibikino, Wakamatsu-ku, Kitakyushu, Fukuoka, 808-0196, Japan
E-mail: fuengfusin.ninnart553@mail.kyutech.jp[*]*, tamukoh@brain.kyutech.jp*
https://www.brain.kyutech.ac.jp/~tamukoh/

Abstract

We propose binary or ternary weights with 8-bit integer activation convolutional neural networks. This model is designed as a middle ground between 8-bit integer and 1-bit or 2-bit quantized models. We discover that conventional 1-bit or 2-bit only-weight quantization methods (i.e., BinaryConnect and Ternary weights network) can be utilized jointly with 8-bit integer activation quantization without significant fractions. Based on these two methods, we evaluate our model with a VGG16-like model and CIFAR10 dataset. Our model provides competitive results to a conventional floating-point model.

.*Keywords*: Quantization, Image Recognition, Model Compression

1. Introduction

With the invention of deep learning, the neural network (NN) has achieved a better performance than human experts, especially in the image recognition task [1]. However, to perform well, the NN must have a large number of parameters. Moreover, to deploy this model into mobile or edge devices in real-time processing is a challenge from several constraints: the device's low computational capacity, memory bandwidth, and others.

One of the methods to reduce the computational time is to convert the 32-bit floating-point (FP32) operations into easier to compute formats such as the fixed-point, integer, or other formats. This process is called quantization. Currently, the default quantization technique supported by major deep learning frameworks (i.e., PyTorch [2] and TensorFlow [3]) is an 8-bit integer (INT8) quantization [4]. In general, INT8 quantization transforms almost all FP32 parameters to INT8 via an affine transformation. The conversion allows the model to operate with only INT8 operations faster than FP32 operations. For instance, in NVIDIA Ampere architecture, NVIDIA A100 delivers around 32 times the number of INT8 operations per second compared to the number of FP32 operations that A100 can perform [5]. Furthermore, INT8 also reduces NN's overall memory footprint by factors of four compared with FP32.

To reduce bit-width to less than 8-bit, several researches show that 1 or 2-bit quantization is possible [6, 7] with some degree of loss in the model performance. In this lower-bit width quantization researches, researches can be categorized into quantizing only-weights and quantizing both weights and activation directions. In general, the quantized only-weight model performs better but consumes higher latency and hardware resources if implemented into the hardware.

We propose an INT8 Activation Ternary or Binary Weights Networks (ITBWN) model to lower bundles of the only-weight quantized model. ITBWN is based on a lower-bit width quantization method; however, we also utilize INT8 based quantization method to quantize the model's activations. This reduces overall hardware resource utilization and latency of only-weight quantized model. Furthermore, we show that both INT8 and lower-bit width quantization methods can be utilized jointly with competitive performance to floating-point model

2. Related Works

In this section, we cover two related work sections: INT8 quantization and lower-bit quantization related researches.

2.1. *INT8 Quantization*

INT8 quantization research problem is framed as how to map between floating-point to INT8 variables. In [4], a floating-point tensor is approximated using an affine transform with floating-point scaling factors S, 8-bit unsigned integer (UINT8) zero-points *Z*, and an INT8 tensor *q*, as shown in Eq. (1). A number variable of *S* and *Z* is either a number of channels of r (per-channel quantization) or a scalar (per-tensor quantization).

$$r = S(q - Z) \tag{1}$$

The optimal S and Z can be found by tracking statistical information (i.e., a minimum and maximum value) during the training or inference. To further reduce more complexity of this approximation, TQT [8] proposed to remove zero-point variables *Z* and affects Eq. (1) to become Eq. (2). In Eq. (2), TQT designs *S* as a scaler variable with the power-of-two quantization. This allows for replacing floating-point multiplication with shift-operations between *S* and *q*.

$$r = S(q) \tag{2}$$

In contrast, instead of using statistical information, TQT uses a training process with a straight-through estimator [9] to decide on *S* and *t* threshold values. *t* is used to clip minimum and maximum values of *r* before using Eq. (2). There are several TQT implementation. One of them Xilinx Brevitas [10] provides an easy-to-use PyTorch implementation of TQT.

2.2. *Lower-bit Quantization*

BinaryConnect (BC) [6] and Ternary Weight Networks, both (TWN) [7] are only-weight and lower-bit quantization methods. BC converts weights w into binary weights w^q using Eq. (3).

$$w^q = sign(w) \tag{3}$$

To make this BC trainable with Eq. (3) which discretizes the gradient to w, Eq. (4) transfers the gradient from quantized weights to floating-point weights. Eq. (4) makes Eq. (3) the same as the identity function during back-propagation.

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial w^q} \tag{4}$$

TWN quantizes weights w into either $\{-S,\ 0, S\ \}$ with Eq. (5) where S and Δ are both positive floating-point variables. TWN applies Eq. (4) to make the model trainable with Eq. (5).

$$w^q = \begin{cases} S: w > \Delta \\ 0: |w| \leq \Delta \\ -S: w < -\Delta \end{cases} \tag{5}$$

Δ can be found as Eq. (6). Where $\boldsymbol{E}$ is an expected value or mean-average of $|w|$.

$$\Delta = 0.7 \times \boldsymbol{E}(|w|) \tag{6}$$

S can be found as Eq. (7). Eq. (7) can be summarized as mean-average of $|w|$ that have values more than Δ.

$$S = \boldsymbol{E}_{i \in \{i | |w_i| > \Delta\}}(|w_i|) \tag{7}$$

3. INT8 Activation Ternary or Binary Weights Networks

In only-weight quantized model, the multiplication between floating-point activations and binary {-1, 1} or ternary weights {-1, 0, 1} can be done using only logic gates [11]. However, in [11], the accumulation of feature maps still requires floating-point accumulations. The motivation of ITBWN is to improve [11] by reducing the complexity of this floating-point accumulation. Converting all activation into INT8 reduces the floating-point accumulation into INT8 accumulation instead. A less complex INT8 datatype reduces overall the computational time. Since INT8 methods are based on approximating floating-point variables, well-have INT8 approximation should behave the same as the floating-point variable.

With this motivation in mind, ITBWN jointly utilizes methods from TQT, BC, and TWN together. ITBWN applies TWN or BC to quantize its weights; however, for its activations, ITBWN uses TQT. An overview of the datatypes in a block of ITBWN model is shown in Fig. 1.

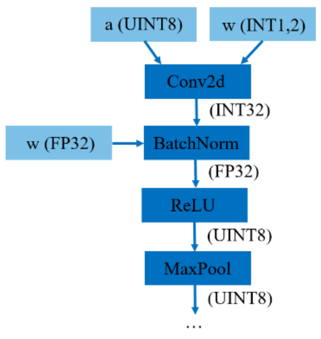

Fig. 1. Overview of datatypes in a block of ITBWN. Where *a* is the activation, *w* is weights or both weights and biases, UINT8 represents 8-bit unsigned-integer.

In this work, there are some modifications to BC, TWN, and TQT. For BC, we do not use clip functions to clip the weights to a range of [-1, 1]. For TWN, we did not use any scaling factors *S*. For TQT, we utilize a default implementation from Xilinx Brevitas, which has minor differences with the TQT setting [12]. Notable differences are Brevitas default settings do not use power-of-two quantization to the scaling factor, and Brevitas applies a different method to initialize the scale factor. At last, we use neither BC, TWN, nor TQT method in the last layer because it may affect the model's performance.

4. Experimental Results and Discussion

In this section, we conducted an experiment with the CIFAR10 dataset. We utilized a model based on VGG-16 [13]. To make VGG-16 operates with the CIFAR10 dataset, we removed the first two fully-connected layers and adjusted a number of input neurons in the last fully-connect layer to 512. We set the hyper-parameters of this experiment as shown in Table 1.

Table 1 Hyper parameters for VGG-16 in CIFAR10 setting.

Hyper Parameters	Value
Epoch	200
Batch size	256
Weight decay	0.0005
Learning rate	0.1

We preprocessed images with mean and standard deviation from each channel of RGB of training dataset. We applied the data augmentation as follows. During the training, the image was padded with zeros to its boundary and randomly cropped back to the original size. Each image was also randomly horizontally flipped. The stochastic gradient descent with momentum is selected to optimize our models. Finally, the learning rate is reduced with Cosine annealing [14].

We set notations as follows: *binary* for BC with TQT, *ternary* for TWN with TQT, *int8* for TQT only, and *float* for all FP32 floating-point model. The test accuracy over training epochs of each model is shown in Fig. 2. The best test accuracy of each model is shown in Table 2.

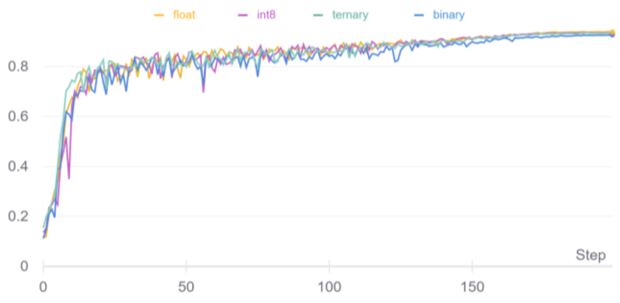

Fig. 2. Test accuracies of different methods per the training epoch.

Table 2 The best accuracy of VGG16 with different quantization methods.

Model	Test Accuracy
binary	0.9269
ternary	0.9351
int8	0.9353
float	0.9389

In this experiment, we found that *float* provides the best test accuracy; however *ternary* and *int8* also provide competitive results to *float*, which is quite surprising. In terms of *binary*, it gives the worst test accuracy. Further analysis of these outcomes, we scope them as future work.

5. Conclusion

We proposed ITBWN or a BC or TWN model with an INT8 quantized activations. Our experiment shows ITBWN with ternary weights provides the competitive result to both INT8 quantized and floating-point models. Converting floating-point activation into INT8 with TQT allows only-weight quantized model to deploy in hardware without worrying about the cost of floating-point accumulation.

For future work, we planned to further analyze why the performance between *ternary* and *int8* is so close to each other. We also would like to find why binary drops a high amount of test accuracy compared to ternary while only 1-bit differs.

Acknowledgements

This research is based on results obtained from a project, JPNP16007, commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

References

1. He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016.
2. Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen et al. "Pytorch: An imperative style, high-performance deep learning library." *Advances in neural information processing systems* 32 (2019): 8026-8037.
3. Abadi, Martín. "TensorFlow: learning functions at scale." In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, pp. 1-1. 2016.
4. Jacob, Benoit, 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*, pp. 2704-2713. 2018.
5. NVIDIA A100 Tensor Core GPU Architecture, Accessed December 13, 2021, https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf.
6. Courbariaux, Matthieu, 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.
7. Li, Fengfu, Bo Zhang, and Bin Liu. "Ternary weight networks." *arXiv preprint arXiv:1605.04711* (2016).
8. Jain, Sambhav R., Albert Gural, Michael Wu, and Chris H. Dick. "Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks." *arXiv preprint arXiv:1903.08066* (2019).
9. Bengio, Yoshua, Nicholas Léonard, and Aaron Courville. "Estimating or propagating gradients through stochastic neurons for conditional computation." *arXiv preprint arXiv:1308.3432* (2013)
10. Pappalardo, Alessandro. Xilinx/brevitas. Zenodo, 2021. https://doi.org/10.5281/zenodo.3333552.
11. Fuengfusin, Ninnart, and Hakaru Tamukoh. "Mixed-precision weights network for field-programmable gate array." *PloS one* 16, no. 5 (2021): e0251329.
12. Cite to quantization techniques in QuantIdentity(bit_width=8) and QuantReLU(bit_width=8) Accessed December 13, 2021, https://github.com/Xilinx/brevitas/issues/370.
13. Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014)
14. Loshchilov, Ilya, and Frank Hutter. "Sgdr: Stochastic gradient descent with warm restarts." *arXiv preprint arXiv:1608.03983* (2016).

Authors Introduction

Dr. Ninnart Fuengfusin

He received his B.Eng. degree from King Mongkut's University of Technology Thonburi, Thailand, in 2016. He received his M.Eng. and D.Eng degrees from Kyushu Institute of Technology, Japan, in 2018 and 2021, respectively. Currently, he is a post-doctoral researcher at the Kyushu Institute of Technology, Japan. His research interests include deep learning, efficient neural network design, and digital hardware design.

Prof. Hakaru Tamukoh

He received the B.Eng. degree from Miyazaki University, Japan, in 2001. He received the M.Eng. and the PhD degree from Kyushu Institute of Technology, Japan, in 2003 and 2006, respectively. He was a postdoctoral research fellow of 21st century center of excellent program at Kyushu Institute of Technology, from 2006 to 2007. He was an Assistant Professor of Tokyo University of Agriculture and Technology, from 2007 to 2013. He is currently an Associate Professor in the Graduate School of Life Science and System Engineering, Kyushu Institute of Technology, Japan. His research interest includes hardware/software complex system, digital hardware design, neural networks, soft-computing and home service robots. He is a member of IEICE, SOFT, JNNS, IEEE, JSAI and RSJ.