SUGISAKA MASANORI

Journal of Robotics, Networking and Artificial Life
Vol. 9(2); September (2022), pp. 171–176
ISSN (Online):2352-6386; ISSN (Print):2405-9021
https://alife-robotics.org/jrnal.html

Research Article

INT8 Activation Ternary or Binary Weights Networks: Unifying Between INT8 and Lower-bit Width Quantization

Ninnart Fuengfusin[1], Hakaru Tamukoh[2]
[1]*Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology, 2-4 Hibikino, Wakamatsu-ku, Kitakyushu, Fukuoka, 808-0196, Japan*
[2]*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*

ARTICLE INFO

Article History

Received 15 December 2021
Accepted 01 July 2022

Keywords

Quantization
Image recognition
Model compression

ABSTRACT

This paper proposes ternary or binary weights with 8-bit integer activation convolutional neural networks. Our proposed model serves as the middle ground between 8-bit integer and lower than 8-bit precision quantized models. Our empirical experiments established that the conventional 1-bit or 2-bit only-weight quantization methods (i.e., BinaryConnect and ternary weights network) can be used jointly with the 8-bit integer activation quantization. We evaluate our model with the VGG16-like model to operate with the CIFAR10 and CIFAR100 datasets. Our models show competitive results to the general 32-bit floating point model.

1. Introduction

The invention of residual networks [1] has drastically improved the performances of neural networks (NNs); these networks now far exceed human performances, especially in image recognition tasks. However, NNs require numerous parameters to perform well. When the number of parameters increases, more computational operations are required. This might prove to be challenging when deploying these models to edge devices that lack high computational capacities.

These problems have led to the emergence of several research fields. One such field is quantization, which converts the conventional 32-bit floating-point (FP32) datatype into simpler datatypes, such as the fixed-point datatype and integer datatype. Currently, the default quantization technique supported by major deep learning frameworks (i.e., PyTorch [2] and TensorFlow [3]) is the 8-bit integer (INT8) quantization [4]. INT8 quantization has become the standard because INT8 datatypes and operations are widely supported by vendors of central processing units and graphics processing units. Therefore, the INT8 quantized model can be used across devices. In general, INT8 quantization transforms almost all FP32 parameters to INT8 via an affine transformation. This conversion allows the quantized model to operate primarily with INT8 operations, which are faster than FP32 operations. For example, in the NVIDIA Ampere architecture, NVIDIA A100 delivers approximately 32 times more INT8 operations per second compared with FP32 operations [5]. Furthermore, INT8 reduces the NN's overall memory storage by a factor of four compared with FP32.

INT8 quantization also positively affects FPGA implementations. For example, when we constructed FP32 and INT8 accumulation between two variables with Xilinx Vitis HLS 2021.2 using the target board Xilinx

Corresponding author's E-mail:

Zynq UltraScale+ MPSoC ZCU102 at 100 MHz, we found that the FP32 accumulation consumed 2-3 clock cycles. In contrast, the INT8 accumulation was done in less than 1 clock cycle. Furthermore, in terms of hardware utilization, FP32 accumulation consumes 2 digital signal processing, 180 flip-flops, and 249 lookup tables, whereas INT8 accumulation uses only 15 lookup tables. This shows that INT8 operations provide significantly lower latencies and less hardware utilization than FP32 operations.

In some cases, we may consider reducing the bit width to less than 8 bits. Several research studies have shown that 1- or 2-bit quantization is possible [6,7] with some reduction in the model performance. This low bit-width quantization research can be categorized into two categories: (i) only-weights quantization research and (ii) weights and activation quantization research. In [8,9], we proposed mixed precision weight networks (MPWNs) that determine the model with the optimized layer combinations between the floating-point, ternary, and binary weight layers. In this case, our model's ternary and binary weights are categorized as only-weights quantization. When the only-weight quantized model was compared with the weights and activation quantized models, it was found that the only-weight quantized model outperformed the other models by a significant margin. However, from our FPGA implementation from [9], the only-weight quantized models suffered from higher latency and hardware resource from the floating-point feature maps accumulation.

To reduce the latency and hardware requirements of the only-weight quantized models, we proposed an INT8 activation ternary or binary weights network (ITBWN). ITBWN uses the same weight spaces as MPWN. However, ITBWN aims to address the MPWN problems of high latency and hardware resource utilization from floating-point accumulations. ITBWN is based on both the lower bit-width and INT8-based quantization methods. ITBWN uses the INT8 quantization method to quantize its activations and uses lower-bit-width method to quantize its weights. The activation is quantized into INT8, which migrates the floating-point to INT8 accumulations, involving faster and lower hardware utilization. In [10], we proved that both INT8 and lower bit-width quantization methods could be used jointly with competitive performance to the floating-point model using the CIFAR10. In this research, we have extended our work in [10] by showing that these methods also provide competitive performance in the CIFAR100 dataset.

2. Related Works

This section will cover two related research fields: INT8 and lower-bit quantization.

2.1. *INT8 Quantization*

The INT8 quantization research problem is framed as follows: "How do we accurately approximate a floating-point tensor with an INT8 tensor and several other variables?" In [4], a floating-point tensor is approximated using an affine transform with the floating-point scaling factor S, an 8-bit unsigned integer (UINT8) zero-point variable Z, and an INT8 tensor q, as shown in Eq. (1). S and Z can be selected either as a vector with a channel size of r (per-channel quantization) or as a scalar (per-tensor quantization). In general, per-channel quantization should provide better performances with more parameters to approximate r.

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

The optimal S and Z values can be found by tracking the statistical information (i.e., the minimum and maximum values) during either the training or inference. To further reduce the complexity of this INT8 approximation, the trained quantization thresholds (TQT) method [11] proposed the removal of the zero-point variable Z, which transformed Eq. (1) into Eq. (2). In Eq. (2), TQT designs S as a scaler variable with the power-of-two quantization. This replaces the floating-point multiplication with shift operations between S and q.

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

In contrast, instead of using the statistical information, TQT uses the training process with the straight-through estimator [12] to *determine S and the threshold values*. The threshold is used to clip the minimum and maximum values of r before using Eq. (2). There are several TQT implementations and one of them is Xilinx Brevitas [13]. Xilinx Brevitas provides an easy-to-use PyTorch implementation of TQT.

2.2. *Lower-bit Quantization*

BinaryConnect (BC) [6] and ternary weight networks (TWN) [7] are only-weight and lower-bit quantization methods. BC converts the 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) is used to transfer the gradient from the quantized weights to the 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 the weights w into $\{-S, 0, S\}$ 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), similar to the case for BC.

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

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

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

S is given in Eq. (7). Eq. (7) is 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-weights quantized models, such as BC and TWN, the floating-point activations and binary $\{-1, 1\}$ or ternary weights $\{-1, 0, 1\}$ can be multiplied using only simple logic operations [9]. However, the feature maps are still accumulated with complex floating-point accumulations. The motivation for proposing the ITBWN model is to further improve the only-weights quantization by reducing the complexity of this floating-point accumulation. This can be done by converting all activation into INT8; therefore, the floating-point accumulations were converted into INT8 accumulations. The less complex INT8 datatype reduces the overall latency from the accumulation. The INT8 methods are based on approximating the floating-point tensors; therefore, a well-behaved floating-point approximation with the INT8 tensor should behave in the same way as the floating-point tensor.

With this motivation, in ITBWN, we use the TQT, BC, and TWN methods. ITBWN applies either TWN or BC to quantize its weights; however, for its activations, ITBWN uses the TQT INT8 quantization. Fig. 1 shows an overview of the datatypes in a block of the ITBWN model.

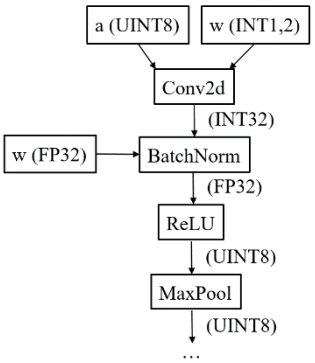

Fig. 1. Overview of the ITBWN datatypes in a basic building block of VGG16. Here, *a* is the activation, *w* are the weights or both weights and biases, and UINT8 represents the 8-bit unsigned integer.

In this study, several modifications are applied to BC, TWN, and TQT. For BC, we did not use the *clip* functions to clip the weights to the [−1, 1] range. For TWN, we did not use any scaling factor *S*, that is, we assigned $S = 1$. For TQT, we used the default implementation provided by Xilinx Brevitas, which has minor differences with the TQT settings [14]. The notable differences are that the Brevitas settings do not use the power-of-two quantized scaling factor, and Brevitas assigns a different initialization method to the scale factor. Finally, we did not use the BC, TWN, or TQT methods in the last layer, because these methods directly affect the model's performance.

4. Experimental Results and Discussion

In this section, we describe the experiments conducted with the CIFAR10 and CIFAR100 datasets. The CIFAR10 dataset consists of 10 classes, and each class

contains 5,000 32 × 32 color training images and 1,000 test images. However, the CIFAR100 dataset contains 100 classes, and each class consists of 500 32 × 32 color training images and 100 test images. With fewer images per class and a higher number of classes than CIFAR10, the CIFAR100 dataset was a more challenging benchmark. In both experiments, we used the VGG16-based model [15], and we modified the VGG16 model to operate with the CIFAR10 and CIFAR100 datasets by removing the first two fully connected layers and adjusting the input and output of the neurons of the last fully connected layer recordings to the dataset. In both the CIFAR10 and CIFAR100 experiments, we set the hyperparameters as shown in Table 1.

Table 1 Hyperparameters for VGG16 in the CIFAR10 and CIFAR100 settings.

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

The images were normalized using the mean and standard deviations of the red, green, and blue channels of images in the training dataset. We performed data augmentation using the following steps. During the training, we zero-padded the image to the image's boundaries and randomly cropped back to the original size. Then, each image was also randomly horizontally flipped. The stochastic gradient descent with the momentum was selected to optimize our models. Finally, the learning rate was scheduled with Cosine annealing [16].

We set the notations as follows: *binary* for BC with TQT, *ternary* for TWN with TQT, *int8* for TQT only, and *float* for all the FP32 floating-point models. The CIFAR10 and CIFAR100 test accuracies over the training epochs are shown in Fig. 2 and Fig. 3, respectively. The best CIFAR10 and CIFAR100 test accuracies are shown in Table 2 and Table 3, respectively.

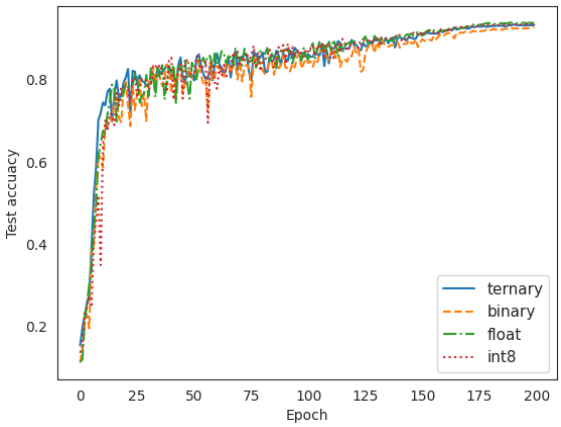

Fig. 2. CIFAR10 test accuracies of different methods as per the training epoch.

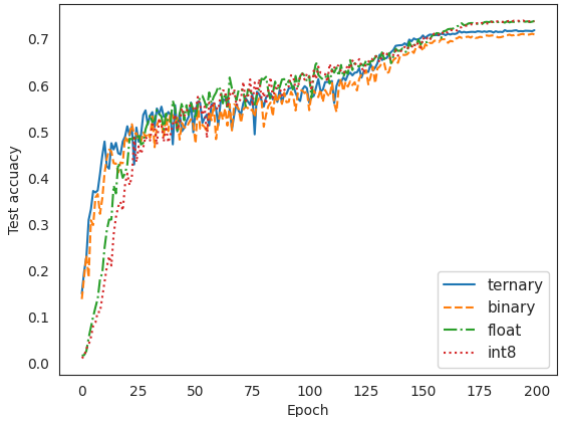

Fig. 3. CIFAR100 test accuracies of different methods as per the training epoch.

Table 2 CIFAR10's best accuracy for the VGG16-like model with and without quantization methods.

Model	Test Accuracy
binary	0.9269
ternary	0.9351
int8	0.9353
float	**0.9389**

Table 3 CIFAR100's best accuracy for the VGG16-like model with and without quantization methods.

Model	Test Accuracy
binary	0.7117
ternary	0.7195
int8	**0.7403**
float	0.7394

We had expected that an increase in the bit precision of the model would improve the test accuracy. Under this

hypothesis, the *float* would provide the best test accuracy, followed by *int8, ternary,* and *binary*, in that order. This hypothesis was confirmed in the CIFAR10 experiment; however, it failed to meet one case in the CIFAR100 experiment, in which *int8* outperformed *float* by a minor margin. We hypothesized that in the CIFAR100 experiment, *int8* quantization might cause a suitable degree of regularization compared with *binary*, *ternary,* and *float. binary* and *ternary* quant ization may have too strong regularization effect, while *float* only has a regularization effect from weight decay. Therefore, with the right degree of regularization from both weight decay and *int8* quantization, the *int8* model outperformed *binary*, *ternary,* and *float*.

5. Conclusion

We proposed ITBWN or a BC or TWN model with INT8 quantized activations. Our experiment shows that ITBWN with ternary weights provides competitive results for both the INT8-quantized and floating-point models. Converting the floating-point activation into INT8 with TQT or BC allowed the only-weight quantized model to deploy without worrying about the hardware or latency costs of floating-point accumulation.

In our future studies, we plan to extend the INT8 quantization-based method to cover quantization methods with lower precisions, such as INT4, INT2, and INT1. This would permit more weight space choices in designing the models. These weight spaces enable finding the best trade-offs between the performance and latency. We would like to apply this quantization technique to the dual steam VGG16 model [17], which can operate in red–green–blue depth images. This allows our model to operate in the robotic domain where depth images may become necessary.

When deploying these models in a robot, which is a hard real-time processing system, we would like to incorporate the concept of models like [18] into our system. This would allow small NNs or sub-NNs to be detachable from the NN. This would allow us to select to perform inference with a faster sub-NN or with the original NN that has better performance. A sub-NN is a part of an NN; therefore, quantizing a sub-NN will affect the original NN. In our future studies, we would like to investigate this dynamic between the sub-NN and original NN using the quantization techniques.

Acknowledgement

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

References

1. K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016
2. A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems,* vol. 32, 2019.
3. M. Abadi, "Tensorflow: learning functions at scale," in *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, pp. 1–1, 2016.
4. 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,* pp. 2704–2713, 2018.
5. NVIDIA A100 Tensor Core GPU Architecture, Accessed May 04, 2022.
6. M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in neural information processing systems,* vol. 28, 2015.
7. F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711,* 2016.
8. N. Fuengfusin and H. Tamukoh, "Mixed precision weight networks: Training neural networks with varied precision weights," in *International Conference on Neural Information Processing,* pp. 614–623, Springer, 2018.
9. N. Fuengfusin and H. Tamukoh, "Mixed-precision weights network for field-programmable gate array," *PloS one,* vol. 16, no. 5, p. e0251329, 2021.
10. N. Fuengfusin, and H. Tamukoh, "INT8 Activation Ternary or Binary Weights Networks," Proceedings *of the 2022 International Conference on Artificial Life and Robotics*, OS15-1, 2022.
11. S. Jain, A. Gural, M. Wu, and C. Dick, "Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks," *Proceedings of Machine Learning and Systems,* vol. 2, pp. 112–128, 2020.
12. Y. Bengio, N. L´eonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432,* 2013.
13. A. Pappalardo. Xilinx/brevitas. Zenodo, 2021.
14. Cite to quantization techniques in QuantIdentity(bit_width=8) and QuantReLU(bit_width=8).

15. K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556,* 2014.
16. I. Loshchilov and F. Hutter, “Sgdr: Stochastic gradient descent with warm restarts,” *arXiv preprint arXiv:1608.03983,* 2016.
17. Y. Yoshimoto and H. Tamukoh, “Fpga implementation of a binarized dual stream convolutional neural network for service robots,” *Journal of Robotics and Mechatronics,* vol. 33, no. 2, pp. 386–399, 2021.
18. N. Fuengfusin and H. Tamukoh, “A sub-model detachable convolutional neural network,” *Journal of Robotics, Networking and Artificial Life,* vol. 8, no. 1, pp. 52–55, 2021.

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, and the M.Eng. and Ph.D. degrees from the Kyushu Institute of Technology, Japan, in 2003 and 2006, respectively. He was a Postdoctoral Research Fellow with the Kyushu Institute of Technology, from 2006 to 2007. He was an Assistant Professor with the Tokyo University of Agriculture and Technology, from 2007 to 2013. He is currently a Professor with the Graduate School of Life Science and Systems Engineering, Kyushu Institute of Technology. His research interests include digital hardware design, soft-computing, and home-service robots. He was the author of works that won the Best Paper Award at IEEE/INNS IJCNN 2019, the Best Live Demonstration Award at IEEE ISCAS 2019, and the Best Paper Award at ICONIP 2013. He is a member of IEICE, JNNS and IEEE.