Hardware-Software Codesign and Ternary Neural Network Applying Batch Normalization Technique on an FPGA

Ing-Chao Lin and Vu Le Minh Nguyen
Department of Computer Science and Information Engineering
National Cheng Kung University

***Abstract*—In recent years, there has been a growing interest in the study of Convolutional Neural Networks (CNNs), which consists of the 2D convolutional layers and a Deep Neural Network, is widely used. However, a huge amount of calculations and memory usage require a massive amount of computing and memory resources. In order to accelerate CNN and reduce memory usage, many data quantization methods and hardware accelerators have been proposed. In this paper, we proposed a Ternary Neural Network(TNN) on an FPGA which treats only 3-values (-1, 0, +1) for the inputs and the weights. For hardware implementation, using ternaried inputs and weights is more suitable. However, the TNN requires the Batch Normalization techniques to retain the classification accuracy. In that case, the additional multiplication and addition require extra hardware, also, the memory access for its parameters reduces system performance. The proposed CNN treats the tenaried inputs and weights with the integer bias. Moreover, we will also propose a library-based framework to allows software developers to exploit the benefits of FPGA acceleration.**

I. Introduction

In recent years, there has been a growing interest in the study of Convolutional Neural Networks (CNNs), which consists of the 2D convolutional layers and a Deep Neural Network, is widely used. Owing to the high accuracy and good performance, CNNs have been widely adopted in many applications such as image classification, face recognition, digital video surveillance and speech recognition, etc. General-purpose processors and GPUs are commonly used to accelerate the operations.

However, general-purpose processors (i.e., CPUs, low-end GPUs) tend to become unfit for these tasks because of their low energy efficiency, especially for those applications that belong to the Internet-of-Things (IoT) and high-end embedded system domains that demand high energy-efficiency and scalability. Therefore, FPGA-based accelerators have started to take their place as viable and promising solutions, thus, investing in their efficient implementation forms an emerging highly valuable design paradigm.

Among these platforms, FPGA is especially suitable as the hardware accelerator owing to its flexibility and efficiency. FPGA-based CNN accelerators can use a minimum precision which reduces the hardware resources and increases the clock frequency, while the GPU cannot do it. Besides that, they also allow low latency operation by enabling the customization of the acceleration architecture, utilizing of hundreds to thousands of on-chip DSP blocks and greater flexibility due to reconfigurability. A large number of operations and kernel weights in the state-of-the-art deep CNN algorithms have become a well-known challenge for their implementations on embedded platforms, which are constrained by limited hardware computing resources and costly off-chip communication. FPGA has held an outstanding performance in low-power and large-scale parallel computing domain and is an excellent candidate for this work due to its reconfigurability.

For any CNN algorithm implementation, there are a lot of potential solutions that result in an enormous design space for exploration. In this project, we introduce an efficient system for deep learning and consider hardware-software codesign to achieve high accuracy and performance. Hardware and software are a perfect couple in deep learning. They not only solve some complex algorithms in CNN but also transfer floating-point data to fixed-point data for weight and input-data in order to increase throughput and reduce bit-width. A quantizer design for fixed-point implementation of CNN is so important because it can significantly improve inference speed. We will formulate and solve an optimization problem to identify optimal fixed-point bit-width allocation across CNN layers. Moreover, the low precision neural network is applied such as the binary and ternary networks, in which the costly multiply-accumulate operations are replaced by accumulations or even binary logic operations, make the on-chip training of CNN quite promising.

With regard to hardware optimization, we will explore input and weight data reuse to reduce memory accesses and energy. We will investigate the trade-off between hardware cost, power consumption, and inference accuracy. Finally, we will propose a library-based framework that allows software developers to exploit the benefits of FPGA acceleration without requiring any expertise on HDL development and low-level design.

In general, we provide a set of optimization knobs, allowing the developer to tune the accelerator according to the requirements of the respective use case. The primary objective

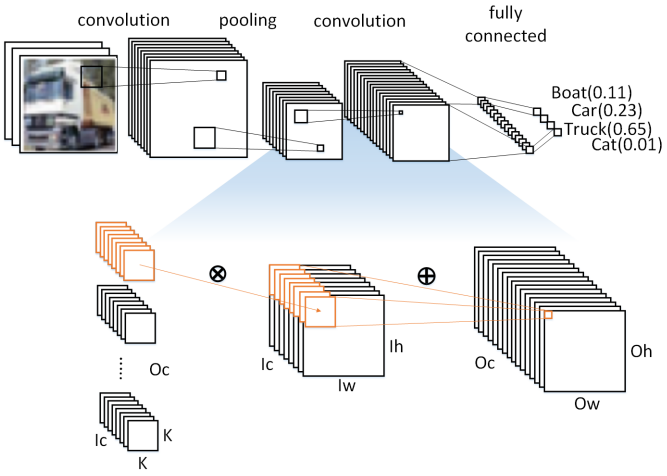

Fig. 1. CNN overview and details of a convolutional layer

of this project will focus on not only optimizing the deep learning models to achieve higher precision with a smaller workload but also achieve higher peak performance with improved energy efficiency. We will also propose a library-based framework to allows software developers to exploit the benefits of FPGA acceleration. Taking all factors into account helps us in hardware-software codesign with a highly efficient deep learning on the FPGA system.

II. Background

A. Convolution Neural Network

CNN is mainly composed of convolutional layer, pooling, and fully-connected layer,as shown in Fig. 1. The concept of convolutional layers makes CNN often used in related applications of image processing and identification. CNN performs feature extractions through multiple convolutional layers. Therefore, CNN is quite excellent in the accuracy and output performance of image processing related applications. In recent years, more and more CNN applications have been designed using the feature of high accuracy, such as object detection [1] and image classification [2].

However, the multiple layers of convolution layers on CNN require considerable memory and computing resources.As shown in Fig. 1, in terms of memory, the weights of each convolution layer require $K * K * Ic * Oc * Wsize$ bytes of memory space, where $K * K$ is kernel size, Ic is input channel, Oc is output channel, and $Wsize$ is number of Bytes for each weight. Furthermore, in terms of computation, each convolution layer requires $K * K * Ic * Oc * Ow * Oh$ times multiplications and additions, where Ow is output width, Oh is output high. The larger the input and output channels, the more memory space and computation resources are required. Therefore, simplified CNN becomes a popular research direction recently.

B. Low Precision Neural Network

Recent research works have considerably reduced the CNN model size and computation complexity by quantizing the weights and activations to low bit-widths while retaining similar accuracy [3], [4]. In simplified CNN researches, quantization is to constrained the data representation to a smaller and discrete set. It can reduce memory space requirements by using lower-precision data. Research in quantization include [5] and [6]. These studies quantize data into 16-bit or 8-bit fixed-point in flexible quantization algorithms. Therefore, the memory usage is only 1/2 to 1/4 and can maintain full-precision accuracy.

At the same time, more simplified data quantization, that is Ternary Neural Network (TNN) [7] which restricts weights and inputs to 3-valued (-1, 0, +1), have been proposed. This method only 2 bits to represent a weight or input in CNN, which can significantly reduce the storage requirement. Therefore, related works which use low-precision CNN accuracy is proposed.

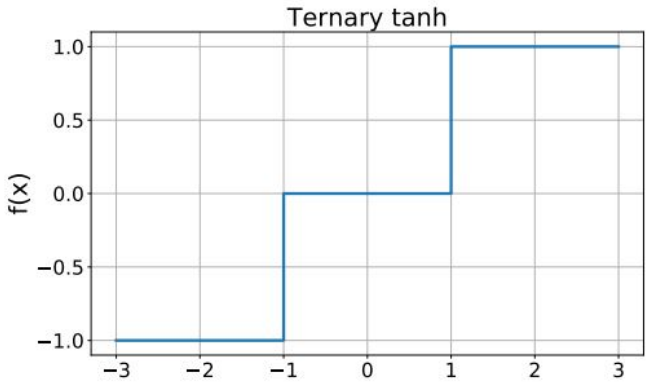

Fig. 2. TNN Activation functions used to define the models

III. Method

A. Ternary Neural Network with Batch Normalization

TNN are extreme examples of quantized neural networks. A network is quantized when the numerical representation of its parameters is a fixed precision. This precision could be constant across the full network or specific for each components (e.g., for different layers). Quantization allows one to reduce the computing resources of a given model for inference and it can be tuned so that it comes with little or no loss in terms of performance. The ternaried *tanh* activation functions are implemented by testing the sign and magnitude of the input and yielding the corresponding value 1 or 0 as seen in Fig. 2. A ternaried *tanh* activation layer preceded by a batch normalization(BN) [8] layer can be further optimized. The BN transformation is:

$$y = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}\gamma + \beta$$

given the mean μ, variance σ^2 , scale γ, and shift β computed during the network training. For a BN followed by a ternary *tanh* activation, the sign of y is enough to determine a node output value. To avoid calculating the scaling of x using FPGA logic, the four BN parameters are used to compute the value of x at which y flips sign. Similarly, the two values of x around which the output of the ternary tanh activation changes are also calculated at compile-time. In the FPGA, each node output is then simply compared against these pre-computed thresholds, outputting the corresponding -1, 0 or +1. This procedure allows one to further save FPGA resources.

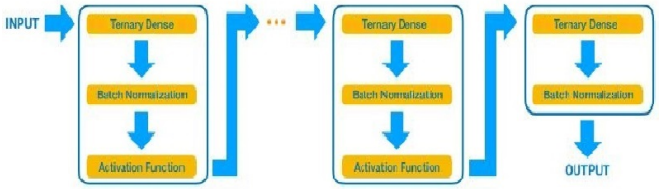

Fig. 3. The MLP architecture used in this study, consisting of a sequence of repeating blocks. Each block, fully connected to the previous and following one, consists of a dense layer, a batch normalization layer, and an activation layer. The last block does not have an activation layer.

In this work, we focus on ternary to a multilayer perceptron (MLP). The basic structure for the adopted architectures is shown in Fig. 3. Each model consists of a sequence of blocks, each composed of a dense, batch normalization(BN) [8], and activation layer. The BN layer shifts the output of the dense layers to the range of values in which the activation function is nonlinear, enhancing the network's capability of modeling nonlinear responses. For ternaried *tanh*, a BN + activation layer sequence can be implemented at small resource cost, which makes this choice particularly convenient for fast inference on edge devices.

B. Library-based Framework

As a starting point, we study the Ternary Neural Network (TNN), and specifically focus on tuning the precisions of weights and activations, to investigate their impacts on the overall classification accuracy. Although our proposed a library-based framework support diverse precisions of the activations crossing layers, in this experiment, we still use unified precision for the intermediate activations to reduce the searching space. Our ultimate goal is to support the most popular CNN model ingredients (layers, activation functions, etc.) and an interface to the most popular DL training libraries, directly (e.g., for TensorFlow [9], Keras [10], and PyTorch [11]) or through the ONNX [12] interface. The library is under development and many of these ingredients are already supported. In general, the library-based framework provides a user-friendly interface is shown in Fig. 5 to deploy custom CNN models on FPGAs, used as co-processing accelerators or as digital circuits in resource-constrained, low-latency computing environments.

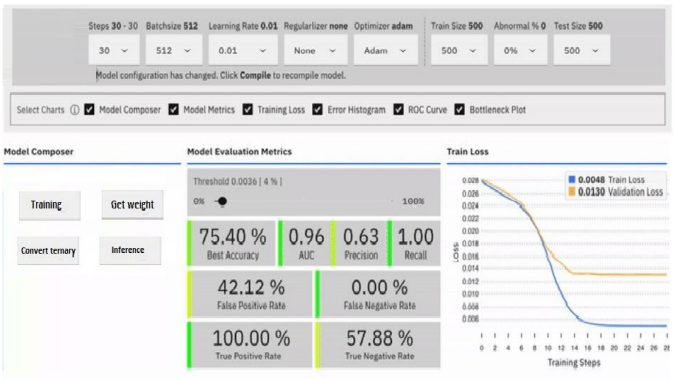

Fig. 4. A user-friendly interface

IV. Experimental Results

In order to allow a fair comparison against related works, we perform our experiments in similar configurations. On Cifar-10, we only use MLPs. We minimize cross entropy loss using stochastic gradient descent with a mini-batch size of 128. During training we use random rotations up to 10 degrees. We report the accuracy rate and the loss rate after 100 epochs. We do not perform any preprocessing on Cifar-10 and we use a threshold-based on the input. We implemented the Ternary Neural Network for the LeNet-5 on the Xilinx Inc. Zynq-Z2 XC7Z020-1CLG400C evaluation board. Since the TNN without the BN only handles +1, 0 and -1, the average output value is unbalanced. As a result, the activation would be also unbalanced. Thus, the classification accuracy tend to be worse compared with the float32 bit precision CNN. By introducing a batch normalization, the mean value for the internal variable in the TNN can be approximated to zero. Therefore, the classification error can be improved. As for the classification errors, we compared with the TNN with the float32 bit precision one. We used the Cifar-10 dataset, and designed the LeNet-5 CNN deep neural network framework [13]. To improve the classification error, we used following tricks.

1) Augmentation: At the training, we used two techniques for the data augmentation. That is, we sliding the image to up, down, left, or right in a random manner in the range from -4 to +4 pixels. The other is performed horizontally inverted at a random. However, these augmentation data are not used during the classification.
2) Normalization: We subjected to contrast adjustment for training dataset. It is a kind of input data normalization.
3) Dropout: We attached the dropout layer to the output of fully connection layers in order to inhibited overfitting.
4) Scheduling: We reduced the training rate after a certain number of epochs (in other word, an epoch means the number of trainings).

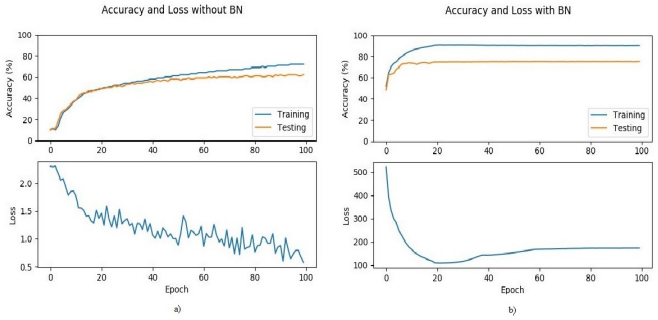

Fig. 5. Comparison TNN with/without the BN

In Fig. 5 comparison the accuracy and loss rate for the TNN with/without the BN. Especially, the accuracy of TNN with the BN is drastically improved 15%. Thus, the BN is an essential technique for the TNN. Therefore, as for the classification accuracy, the TNN is considerable.

In this study, we propose TNN for resource-efficient applications of deep learning. Energy efficiency and area efficiency are brought by not using any multiplication nor any floating-point operation. Through experimental evaluation, we demonstrate the performance of TNN both in terms of accuracy and resource-efficiency, with CNNs as well as MLPs. In terms of accuracy, TNN with BN perform better than without BN with relatively smaller networks in all of the benchmark datasets except one. Unlike previous works, TNNs use ternary neuron activations using a step function with two thresholds. This allows each neuron to choose a sparsity parameter for itself and gives an opportunity to remove the weights that have very little contribution. In that respect, TNNs inherently prune the unnecessary connections. We also develop a purpose-built hardware for TNNs that offers significant throughput and area efficiency and highly competitive energy efficiency. It also often has higher accuracy with our new training approach.

V. Conclusion

This paper proposed the Ternary Neural Network with Batch Normalization which treats only 3-values (+1, 0, -1) for the inputs and the weights and the integer biases. We implemented the LeNet-5 CNN on the Xilinx Inc. Zynq-Z2 XC7Z020-1CLG400C evaluation board. Compared with the TNN without BN, it was 25% faster for testing accuracy and 15% for training accuracy. We also research a library-based framework that provides a user-friendly interface to deploy custom CNN models on FPGAs, used as co-processing accelerators or as digital circuits in resource-constrained, low-latency computing environments. Moreover, it also allows software developers to exploit the benefits of FPGA acceleration. In the future, we are going to focus improve hardware design to increase high-performance, processing speed and efficiency-power.

References

[1] S. Gidaris and N. Komodakis, "Object detection via a multi-region and semantic segmentation-aware cnn model," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1134–1142.

[2] Y. Wei, W. Xia, M. Lin, J. Huang, B. Ni, J. Dong, Y. Zhao, and S. Yan, "Hcp: A flexible cnn framework for multi-label image classification," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 9, pp. 1901–1907, 2015.

[3] M. C. et al, "Trainedtraining deep neural networks with low precision multiplications," *arXiv:1412.7024*, 2014.

[4] P. G. et al, "Hardware-oriented approximation of convolutional neural networks," *arXiv:1604.03168*, 2016.

[5] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 26–35.

[6] B. Jacob, S. Kligys, B. Chen *et al.*, "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.

[7] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," *arXiv preprint arXiv:1612.01064*, 2016.

[8] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv:1502.03167*.

[9] M. A. et al, "Tensorflow: Large-scale machine learning on heterogeneous system," 2015. [Online]. Available: http://tensorflow.org/

[10] F. C. et al, "Keras," 2015. [Online]. Available: https://github.com/fchollet/keras

[11] A. P. et al, "Automatic differentiation in pytorch," *in NIPS-W*, 2017.

[12] J. B. et al, "Onnx: Open neural network exchange," 2019. [Online]. Available: https://github.com/onnx/onnx

[13] "Chainer: A powerful, flexible, and intuitive framework of neural networks." [Online]. Available: http://chainer.org/

[14] K. Guo, L. Sui, J. Qiu *et al.*, "Angel-eye: A complete design flow for mapping cnn onto embedded fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 1, pp. 35–47, 2017.

[15] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.