A Memory-Based Realization of a Binarized Deep Convolutional Neural Network

Hiroki Nakahara
Tokyo Institute of Technology, Japan

Haruyoshi Yonekawa
Tokyo Institute of Technology, Japan

Tsutomu Sasao
Meiji University, Japan

Hisashi Iwamoto
Poco a Poco Networks, Japan

Masato Motomura
Hokkaido University, Japan

***Abstract*—A pre-trained deep convolutional neural network (CNN) is a feed-forward computation perspective, which is widely used for the embedded systems, requires high power-and-area efficiency. This paper realizes a binarized CNN which treats only binary 2-values (+1/-1) for the inputs and the weights. In this case, the multiplier is replaced with an EX-NOR circuit. To reduce both power and area, we realize the 2-valued CNN by off- and on-chip memories. Since our 2D convolution operations are realized by the on-chip memory, our implementation consumes lower power than the DSP block. We decompose the memory part, and realize them by a cascade of memories (LUT cascade). By introducing a batch normalization technique, the classification error for the binarized CNN can be improved. We implemented the CIFAR-10 benchmark on the NetFPGA-SUME board, which has the Xilinx Inc. Virtex 7 FPGA and three on-chip QDR II+ Synchronous SRAMs. Compared with the conventional FPGA realizations, the performance is 2.82 times faster, the power efficiency is 1.76 times, and the area efficiency is 29.13 times better.**

I. Introduction

Recently, a deep convolutional neural network (CNN), which consists of the 2D convolutional layers and a deep neural network, is widely used. With the increase of the number of layers, the CNN can increase recognition accuracy. Thus, a large-scale CNN is desired. Most CNN implementations on FPGAs use high-end FPGAs and off-chip memories. Since 2D convolution operations on the CNN occupy more than 90% of computation time [3], they used many DSP blocks and consumes much power. In contrast, our implementation uses on-chip memories instead of the DSP blocks. Since it is a memory-based one, the power consumption is lower than the DSP block-based one. Also, we used the binarized CNN [1], which restricts the internal values to 2-valued (-1 and 1) [4]. The binarized one achieved near state-of-the-art error rate with only a single bit per weights and activation functions with a batch normalization technique. Thus, the CNN implementation would consume less FPGA resources than the conventional one.

II. Deep Convolutional Neural Network (CNN)

The CNN has multiple **layers**, the typical layer consists of **a 2D convolutional layer**, **a pooling layer**, and **a classifier layer**. Each layer consists of multiple **feature maps**.

[1]The code for a binarized CNN with a BN is available at http://www.hirokinakaharaoboe.net/binCNN

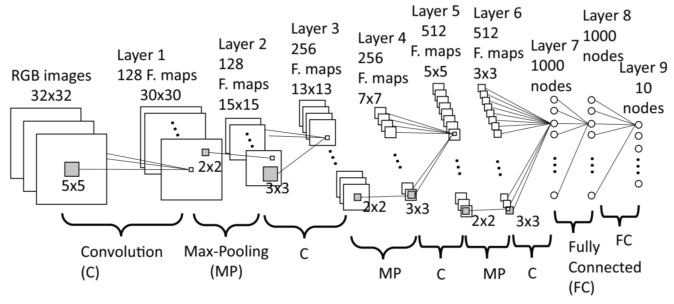

Fig. 1. Example of a CNN for the CIFAR-10 [2].

To recognize the input image, first, the feature map reacts corresponding subdivided teacher data by 2D convolutional layers with pooling layers. Then, the classifier selects the appropriate reactions from feature maps. Fig. 1 shows the CIFAR-10, which consists of nine layers. The front seven layers consist of 2D convolutional layers and max-pooling layers, while the rear two layers consist of fully connected neural networks. In this paper, for layer i, K_i denotes the kernel size, N_i denotes the number of feature maps, and L_i denotes the feature map size. The 2D convolution computes the output by shifting a $K \times K$ size **kernel**. For (x, y) at the output feature map value $i + 1$, the following MAC (multiply-accumulation) operation is performed:

$$\begin{aligned} U_{i+1,x,y} &= \sum_{k=0}^{N_i-1} (Y_{k,x,y} + \sum_{m=0}^{K-1} \sum_{n=0}^{K-1} X_{k,x+m,y+n} W_{k,m,n}) \\ Z_{i+1,x,y} &= f(U_{i+1,x,y}), \end{aligned}$$

where X denotes an input, W denotes a weight, Y denotes a bias, U denotes an internal output, f denotes an activation function, and Z denotes an output value to be mapped to (x, y) at the output feature map $i + 1$. In the fully connected layer, $N_i = 1$ and $K_i = 1$. Note that, in this paper, a capital letter denotes an integer, while a small letter denotes a binary value. The previous work [3] showed that 2D convolution operations occupy more than 90% of the computation time. Thus, we focus on the acceleration of the 2D convolutional layers using less amount of hardware. Note that, the original BinaryNet paper showed that the first layer must be realized by a non-binarized one. Thus, in the paper, we realize binarized operations at layers 3, 5, 7, 8 and 9. Also, we used the binarized activation functions for them except for the 9 layer.

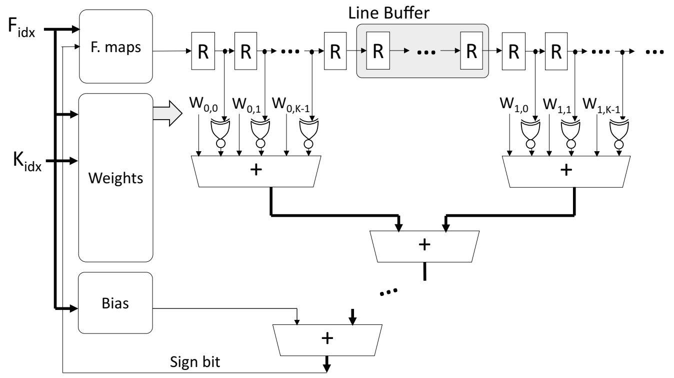

Fig. 2. Straightforward realization of layers 3,5, and 7.

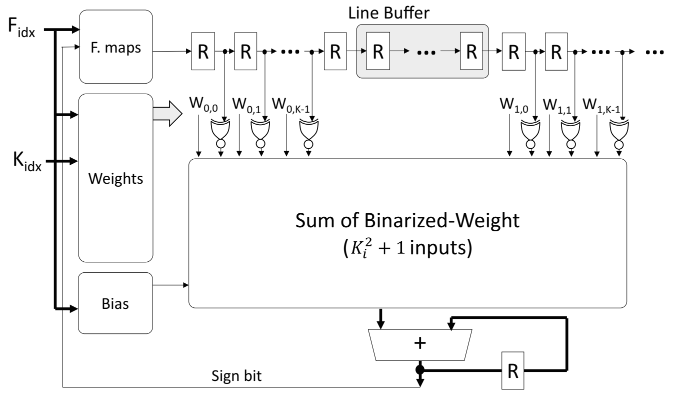

Fig. 3. Memory reduction by time-multiplexing.

III. Memory-based Realization for a Binarized CNN

A. Binarized CNN

The binarized CNN restricts the weight and the input values to only -1 and 1. The binarized 2D convolutional layer performs **a sum of binarized weights** operation as follows:

$$\begin{aligned} U_{i+1,x,y} &= \sum_{k=0}^{N_{i+1}-1} (y_{k,x,y} \\ &\quad + \sum_{m=0}^{K-1}\sum_{n=0}^{K-1} \overline{x_{k,x+m,y+n} \oplus w_{k,m,n}}) \qquad (1) \\ z_{i+1,x,y} &= Sign(U_{i+1,x,y}), \end{aligned}$$

where $Sign(x)$ denotes the sign function as follows:

$$Sign(x) = \begin{cases} 1 & (if\ x \geq 0) \\ -1 & (otherwise) \end{cases}$$

Note that, in layer 1, since the input is the RGB color images, the accuracy for the input is 8 bits. The feature maps are stored in the off-chip buffer memory (F. map). Its initialization value is the input RGB color images. Weights are stored in the weights memory (Weights), while the bias is stored in the bias memory (Bias). First, it stores the input feature map value into the shift register. Then, it reads the corresponding kernel values from the shift register. Third, it generates the internal outputs through inverters depending on the sign of weights and an adder-tree. Finally, it generates the binarized output. Fig. 2 shows a straightforward realization of layers 3, 5 and 7. Since the inputs are also binarized, the multipliers are realized by the EXNOR gates.

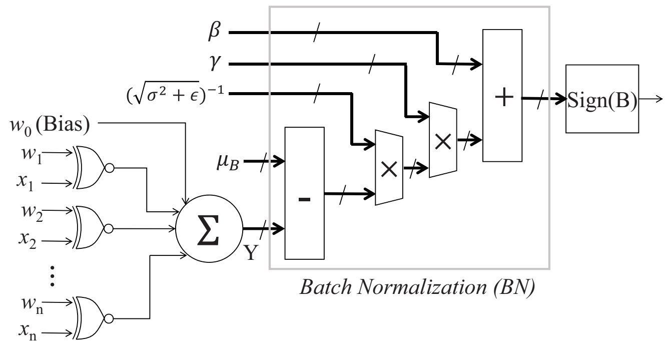

Fig. 4. Binarized neural network with a BN.

B. Batch Normalization (BN)

Especially, since the binarized CNN only handles +1 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 (BN)** [13], the mean value for the internal variable in the binarized CNN can be approximated to zero. Therefore, the classification error can be improved. At the training, the BN finds parameters γ and β in order to regularize the variance to 1 and the average to 0 for each mini-batch. The BN algorithm for training is shown as follows:

Algorithm 3.1: Input: Mini-batch $B = \{X_1, X_2, \ldots, X_m\}$ Output: Parameters γ and β.

1. Obtain average for each mini-batch: $\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} X_i$.
2. Obtain variance for each mini-batch: $\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(X_i - \mu_B)^2$.
3. Perform normalization: $\hat{X}_i \leftarrow \frac{X_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$.
4. Obtain γ and β, that regularizes the variance to 1 and the average to 0 for $B_i \leftarrow \gamma\hat{X}_i + \beta$.
5. Terminate.

Algorithm 3.1 performs the normalization for each mini-batch. A hyper parameter ϵ is set for a coefficient stabilization, which is used to adjust the training time. Both γ and β have been already trained during classification, and μ_B and σ_B^2 have been already obtained by training data set. The BN just reads them from the off-chip memory. Fig. 4 shows a binarized neural network with a BN operation, where w_i, x_i, z are binary variables, and other parameters are integer ones. It requires additional cost for the multiplier and the adder for area, while the memory accesses for trained parameters.

C. Comparison of Classification Errors

As for the classification errors, we compared with the binarized CNN with the float32 bit precision one. We used the CIFAR-10 CNN by the Chainer deep neural network framework [6]. Note that, both CNNs used the same layer structure. Fig. 5 compares the classification error for the float32

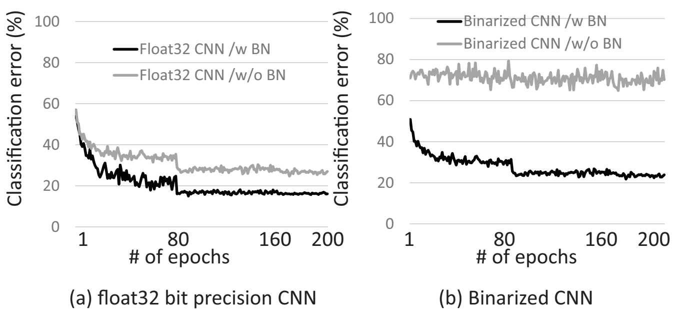

Fig. 5. Classification errors for CIFAR-10.

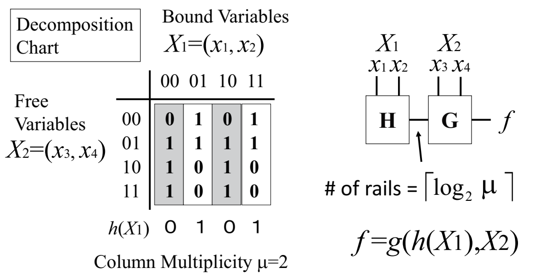

Fig. 6. Functional decomposition.

bit precision CNN with that for the binarized one with/without the BN, and that for the binarized one with the BN or without one. From Fig. 5, the BN improves the classification error. Especially, that for the binarized CNN is drastically improved. Compared with the binarized CNN using the BN with the float32 one, the difference of the classification error rate is 6.5%. Thus, the BN is an essential techniques for the binarized CNN.

D. Memory-based Realization

Since the straightforward realization requires many DSP blocks, it is a power hungry. In this paper, to achieve area-and-power efficient implementation, we realize the 2D convolution operations by on-chip memory. Unfortunately, since the number of inputs is $n = N_i \times K_i^2 + 1$, it requires a huge memory. Thus, we use the time multiplexing technique, which sequentially reads $K_i^2 + 1$ inputs as shown in Fig. 3. Although it requires an additional adder and register, the memory requirement is reduced by $\frac{1}{N_i}$. In the next section, we will introduce the LUT cascade, which is obtained by recursively decomposing the memory part.

IV. Memory Size Reduction by an LUT Cascade

A. Functional Decomposition

Consider a function $F(X) : B^n \to \{0, 1, \ldots m-1\}$, where $X = (x_1, x_2, \ldots, x_n)$, $x_i \in B$, and $B = \{0, 1\}$. Let (X_L, X_H) be a partition of X into two parts. A *decomposition chart* of F is the two-dimensional matrix, where each column label has distinct assignment of elements in X_L, and each row label has distinct assignment of elements in X_H, and the corresponding matrix value is $F(X_L, X_H)$. The number of different column patterns in the decomposition chart is the *column multiplicity* μ. X_L denotes the *bound variables*, and X_H denotes the *free variables*. Fig. 6 shows a functional decomposition. Connections with adjacent LUTs are called *rails*, where $r = \lceil log_2 \mu \rceil$. Let $|X_L| = n_1$ and $|X_H| = n_2$. In this case, the total amount of memory for the functional decomposition is $2^{n_1} \times r_1 + 2^{r_1 + n_2}$ bits. Thus, a function with small μ can be reduce the memory size.

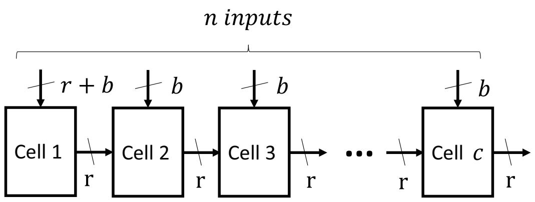

Fig. 7. LUT cascade for layers 3,5 and 7 (Realized by distributed memories).

By applying functional decompositions recursively, we have *an LUT cascade* [12] as shown in Fig. 7. In Fig. 7, each LUT is called by **a cell**. In this paper, c denotes the number of cells. We have $c = \lceil \frac{n-r}{b} \rceil$, where b denotes the number of external inputs to a cell. Since the amount of each cell is $2^{r+b}r$ bits, the total amount of memory is $M = c \cdot 2^{r+b}r$. The Xilinx FPGA consists of 18Kb BRAMs and 64b distributed RAMs. In the next section, we will show a design method for the LUT cascade with given size of r and n.

V. Memory-based Realization by LUT Cascades

A. Design for LUT cascades

As shown in the previous section, the amount of memory for the LUT cascade depends on the number of rails r and the number of inputs n. In the memory-based binarized CNN for the CIFAR-10, since $n = K^2$, the amount of memory depends on r, i.e., the column multiplicity μ. In this section, first, we introduce the upper bound on μ for the MAC operation in the binarized CNN. Then, we obtain the upper bound on the amount of memory for the LUT cascade.

Suppose that the bias value is multiplied by the constant one. As shown in Expr. (1), the memory part shown in Fig. 3 realizes the function:

$$f_{WS} = \sum_{i=0}^{K^2-1} W_i X_i,$$

where f_{WS} is **a weighted-sum (WS) function**. The column multiplicity for a WS function has been analyzed in [11].

Theorem 5.1: When the output of the WS function is represented by q bits, its column multiplicity is at most 2^q.

Corollary 5.1: When the output of the WS function is represented by q bits, its number of rails for the LUT cascade is at most q.

From the above analyses, we can estimate the amount of memory for the given kernel size K^2 and the accuracy q for the output of the 2D convolution operation. In this section, we use the minimum on-chip memories to realize an area-efficient LUT cascade.

B. Proposed Architecture

Fig. 8 shows a proposed architecture. Weights, bias, and values of the feature maps are stored in the QDR II+ synchronous SRAMs (SSRAMs). For each layer, first it switches

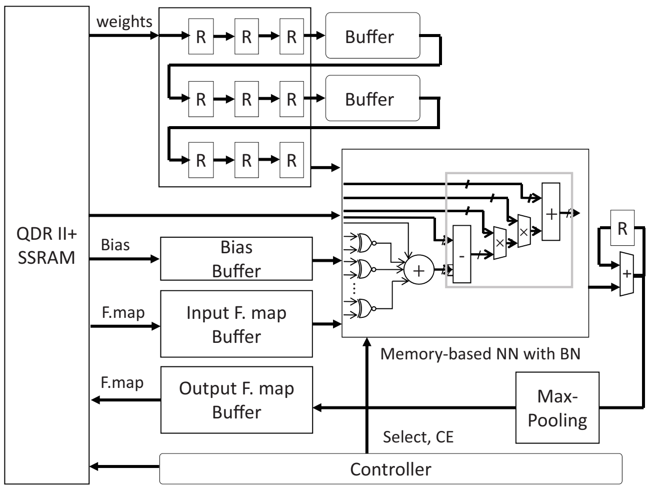

Fig. 8. Proposed Architecture.

TABLE I. COMPARISON WITH OTHER FPGA REALIZATIONS.

Year	2015 [15]	2016 [14]	Proposed
Platform	Virtex7 VX485T	Zynq XC7Z045	Virtex7 XC7V690T
Clock (MHz)	100	150	450
Memory Bandwidth (GB/s)	12.8	4.2	13.8
Quantization Strategy	32bit float	16bit fixed	1bit (Binary)
Power (W)	18.61	9.63	15.44
Performance (GOPS)	61.62	187.80	**530.40**
Area Efficiency (GOPS/Slice$\times 10^{-4}$)	8.12	35.8	**1043.2**
Power Efficiency (GOPS/W)	3.31	19.50	**34.37**

to the adequate LUT cascades. Then, it performs the 2D convolution operations. Next, it operates the max-pooling by comparators. The proposed architecture has the shift registers and buffers to access indices for the corresponding kernel. This structure is similar to the previous architecture [14].

VI. Experimental Results

We implemented the binarized CNN for the CIFAR-10 on the Digilent Inc. NetFPGA-SUME evaluation board, which has the Xilinx Virtex 7 FPGA (XC7V690T, 108,300 Slices, 2,940 18Kb BRAMs, 3,600 DSP48Es) and the Cypress Semiconductor Corp. three 72 Mb QDRII+ SSRAM CY72263KV18. We used the Xilinx Inc. Vivado 2015.1 with timing constrain 450 MHz. Our implementation used 5,088 Slices, 372 18Kb BRAMs, and no DSP48Es. Also, it satisfied the timing constraint for real-time applications. The number of operations for the implemented CNN was 530.8 GOPS (Giga Operations Per Seconds). We measured the power consumption without that for the power sources on the board: It was 15.44 W. Since the implemented CNN operated 530.8 GOPS with 5,088 Slices, the area efficiency is 1043.2 (GOPS/Slice$\times 10^{-4}$). Also, the power efficiency is 34.37 (GOPS/W).

Since different implementations used different FPGAs and DCNNs, we used the performance per area efficiency (GOPS/Slice) to do fair comparison. Table I compares various FPGA implementations. From Table I, compared with the recent implementation [14], our implementation is 2.82 times faster, it is 29.13 times better with respect to the area efficiency, and it is 1.76 times lower with respect for the power efficiency.

VII. Conclusion

A binarized CNN treats only 2-values (+1/-1) for the inputs and the weights. Since our binarized 2D convolution operations were realized by the on-chip memory, it consumed lower power than the DSP block-based ones. Also, we introduced the batch normalization technique, which improved the classification error. We decomposed the memory parts, and realized them by the LUT cascades. In this paper, we analyzed an upper bound for the memory size of the LUT cascade, and showed the design method which uses the minimum memory resource on the FPGA. Compared with the conventional FPGA realizations, the performance is 2.82 times faster, the power efficiency is 1.76 times, and the area efficiency is 29.13 times better. Thus, the binarized realization is sutable for the FPGA based CNN.

VIII. Acknowledgments

This research is supported in part by the Grants in Aid for Scientistic Research of JSPS, and an Accelerated Innovation Research Initiative Turning Top Science and Ideas into High-Impact Values program (ACCEL) of JST.

References

[1] S. Cadambi, A. Majumdar, M. Becchi, S. Chakradhar and H. P. Graf, "A programmable parallel accelerator for learning and classification," *Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT)*, 2010, pp.273-284.

[2] The CIFAR-10 data set, http://www.cs.toronto.edu/ kriz/cifar.html

[3] J. Cong and B. Xiao, "Minimizing computation in convolutional neural networks," *Int'l Conf. on Artificial Neural Networks and Machine Learning (ICANN)*, 2014, pp. 281-290.

[4] M. Courbariaux, I. Hubara, D. Soudry, R.E.Yaniv, Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," *Computer Research Repository (CoRR)*, Mar., 2016, http://arxiv.org/pdf/1602.02830v3.pdf

[5] M. Courbariaux, Y. Gengio, and D. J. Pierre, "BinaryConnect: Training deep neural networks with binary weights during propagations," *arXiv:1511.00363*, 2016.

[6] Chainer: A powerful, flexible, and intuitive framework of neural networks, http://chainer.org/

[7] C. Farabet, C. Poulet, J. Y. Han and Y. LeCun, "CNP: An FPGA-based processor for convolutional networks," *Int' Conf. on Field-Programmable Logic and Applications (FPL)*, 2009, pp.32-37.

[8] C. Y. Lee, P. W. Gallagher and Z. Tu, "Generalizing pooling functions in convolutional neural networks: Mized, gated, and tree," *arXiv prprnt arXiv:1509.08985*, 2015

[9] M. Peemen, A. A. A. Setio, B. Mesman and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," *Int'l Conf. on Computer Design (ICCD)*, 2013, pp.13-19.

[10] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto and H. P. Graf, "A massively parallel coprocessor for convolutional neural networks," *Int'l Conf. on Application-specific Systems, Architectures and Processors (ASAP)*, 2009, pp.53-60.

[11] T. Sasao, "Analysis and synthesis of weighted-sum functions," *IEEE Trans. on CAD*, Vol. 25, No. 5, 2006, pp. 789-796.

[12] T. Sasao, *Memory-based logic synthesis*, *Springer*, 2011.

[13] I. Sergey and S. Christian, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.

[14] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang and H. Yang, "Going deeper with embedded FPGA platform for convolutional neural network," *Int'l Symp. on Field-Programmable Gate Arrays (ISFPGA)*, 2016, pp. 26-35

[15] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," *ISFPGA*, 2015, pp.161-170.