IEICE TRANS. INF. & SYST., VOL.E101–D, NO.2 FEBRUARY 2018

PAPER Special Section on Reconfigurable Systems

A Threshold Neuron Pruning for a Binarized Deep Neural Network on an FPGA

Tomoya FUJII[†a)], ***Nonmember***, **Shimpei SATO**[†b)], ***and*** **Hiroki NAKAHARA**[†c)], ***Members***

SUMMARY For a pre-trained deep convolutional neural network (CNN) for an embedded system, a high-speed and a low power consumption are required. In the former of the CNN, it consists of convolutional layers, while in the latter, it consists of fully connection layers. In the convolutional layer, the multiply accumulation operation is a bottleneck, while the fully connection layer, the memory access is a bottleneck. The binarized CNN has been proposed to realize many multiply accumulation circuit on the FPGA, thus, the convolutional layer can be done with a high-seed operation. However, even if we apply the binarization to the fully connection layer, the amount of memory was still a bottleneck. In this paper, we propose a neuron pruning technique which eliminates almost part of the weight memory, and we apply it to the fully connection layer on the binarized CNN. In that case, since the weight memory is realized by an on-chip memory on the FPGA, it achieves a high-speed memory access. To further reduce the memory size, we apply the retraining the CNN after neuron pruning. In this paper, we propose a sequential-input parallel-output fully connection layer circuit for the binarized fully connection layer, while proposing a streaming circuit for the binarized 2D convolutional layer. The experimental results showed that, by the neuron pruning, as for the fully connected layer on the VGG-11 CNN, the number of neurons was reduced by 39.8% with keeping the 99% baseline accuracy. We implemented the neuron pruning CNN on the Xilinx Inc. Zynq Zedboard. Compared with the ARM Cortex-A57, it was 1773.0 times faster, it dissipated 3.1 times lower power, and its performance per power efficiency was 5781.3 times better. Also, compared with the Maxwell GPU, it was 11.1 times faster, it dissipated 7.7 times lower power, and its performance per power efficiency was 84.1 times better. Thus, the binarized CNN on the FPGA is suitable for the embedded system.

key words: *machine learning, deep learning, pruning, FPGA*

1. Introduction

1.1 Embedded Computer Vision Systems

The embedded computer vision systems emulates the human vision, and they are used in the wide applications as follows: a human face recognition [27], a human and object detection [14], a human pose estimation [30], a string recognition in a scene [15], a road traffic sign recognition [6], a sport scene recognition [19], and a human action recognitions [9], [18], respectively. These application requires high accuracy, low power, and high performance.

Manuscript received May 2, 2017.
Manuscript revised September 12, 2017.
Manuscript publicized November 17, 2017.

[†]The authors are with Department of Information and Communications Engineering, Tokyo Institute of Technology, Tokyo, 152–8552 Japan.
a) E-mail: t-fujii@eda.ict.e.titech.ac.jp
b) E-mail: satos@eda.ict.e.titech.ac.jp
c) E-mail: nakahara@ict.e.titech.ac.jp

DOI: 10.1587/transinf.2017RCP0013

1.2 Convolutional Deep Neural Network (CNN)

Recently, for these systems, a convolutional deep neural network (CNN), which consists of the 2D convolutional layers and the fully connected neural network, is widely used. Since the CNN emulates the human vision, it has a high accuracy for an image recognition. Previous researches showed that the CNN outperforms conventional techniques. With the increase of the number of layers, the CNN can increase classification accuracy. Thus, a large-scale CNN is desired. To keep up with the real-time requirement of the embedded vision system, since the existing system using a CPU is too slow, the acceleration of the CNN is necessary [21]. Most software-based CNNs use the GPUs [2], [3], [7], [28], [29]. Unfortunately, since the GPU consumes much power, they are unsuitable for the embedded system [10]. Thus, FPGA-based CNNs are required for a low-power and a real-time embedded vision system. As for the classification accuracy, the CNN using a fixed-point representation has almost the same accuracy as one using a floating-point representation [12]. The FPGA can use a minimum precision which reduces the hardware resources and increases the clock frequency, while the GPU cannot do it. A previous work [10] reported that, as for the performance per power, the FPGA-based CNN is about 10 times more efficient than the GPU-based one.

1.3 Problems of the Conventional CNNs

Typically, the CNN consists of the convolutional layers and the fully connected layers. Figure 1 (a) shows that operations demanded in different layers, while that for (b) shows that the number of weights in different layers [32]. As shown these figures, in the convolutional layers, the multiply accumulation operation is a bottleneck, while in the fully connected layers, the memory accesses is a bottleneck. The binarized CNN [8] has been proposed to realize many multiply accumulation circuit on the FPGA, thus, the convolutional layer can be done with a high-speed operation. However, even if we apply the binarization to the fully connection layer, the amount of memory was still a bottleneck. In the paper, we propose the neuron pruning to reduce the memory size. Figure 2 (a) shows an example of edge pruning of the fully connected layer. In the conventional techniques, the randomly pruning techniques of edges have been proposed [1], [16]. However, in the hardware realization point

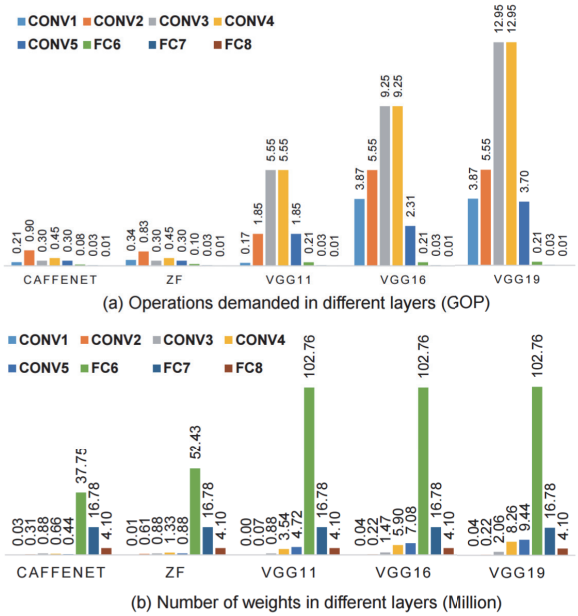

Fig. 1 The complexity distribution of state-of-the-art CNN models: (a) Distribution of operations by theoretical estimation; (b) Distribution of weight number [32].

of view, since the memory access of the sequential address is suitable, the random edge pruning may cause a performance degradation.

1.4 Proposed Method

The previous work showed that the neuron pruning for the floating point precision (non-binary) CNN can reduce the number of neurons. In the paper, we propose a neuron pruning instead of the edge pruning to the binarized CNN. Figure 2 (b) shows an example of neuron pruning of the fully connected layer. Since by pruning all the incoming and the outgoing edges of a neuron is equivalent to the neuron pruning, in general, the edge pruning can eliminate more edges than neuron pruning. However, even if the neuron pruning is applied, since it maintains the sequential memory access, it is suitable for the hardware realization. Since the proposed neuron pruning can eliminate almost edges, we can store all the remaining edges into the on-chip memory on the FPGA. In the paper, we propose the serial-input parallel-output circuit for the fully connected layer. To realize a high-performance circuit, it efficiently uses on-chip memories and DSP slices on the FPGA. In the experiment, we show that the FPGA based realization outperforms than the CPU and the GPU realizations.

1.5 Contributions of the Paper

The previous contributions [13] were as follows:

1. We proposed the threshold based neuron pruning techniques for the FPGA realization of the fully connected layer on the deep neural network. The proposed one is suitable to the on-chip realization of the FPGA. The experimental result showed that as for the 99% accuracy, it eliminated the number of neurons by 89.3% for the VGG-11 CNN.
2. We proposed the sequential-input parallel-output circuit for the fully connected layer. It efficiently uses on-chip memories and DSP slices on the FPGA. Since the proposed circuit can store all the weights of the fully connected layer, it can realize a wide band of the memory access. Our technique is a complementary to the conventional techniques that accelerate the convolutional layers for the FPGA. We expanded the applicability of the CNN using the FPGA.
3. We applied the neuron pruning for the fully connected layers on the VGG-11 CNN, then implemented them on the Digilent Inc. NetFPGA-1G-CML board. Our FPGA implementation outperformed the GPU and the CPU implementations.

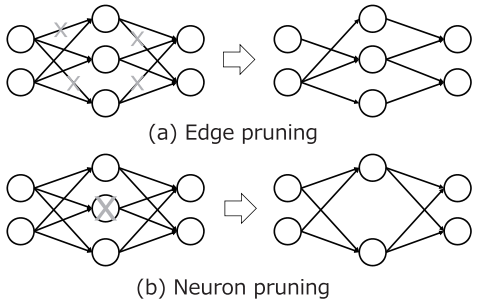

Fig. 2 Comparison of pruning techniques.

The previous work only applied the neuron pruning to the floating point precision CNN, while in the paper, we applied the neuron pruning to the binarized CNN. As far as we know, this is the first report. Additionally, in the paper, new contributions are as follows:

1. We reduced the number of neurons using the neuron pruning technique to the binarized fully connection layer, whose memory size was the bottleneck.
2. To further reduction, we applied the neuron pruning with retraining to the binarized deep neural network. With this technique, we successfully reduced the number of neurons for the fully connection layer.
3. The previous work only implemented the fully connection layers, while in the paper, we implemented all layers of the binarized CNN on the Xilinx Inc. Zedboard. Compared with the ARM Cortex-A57, it was 1773.0 times faster, it dissipated 3.1 times lower power, and its performance per power efficiency was 5781.3 times better. Also, compared with the Maxwell GPU, it was 11.1 times faster, it dissipated 7.7 times lower power, and its performance per power efficiency was 84.1 times better. Thus, we showed that the binarized CNN on the FPGA is suitable for the embedded system.

This paper is the updated version of the previous pub-

lication [13].

1.6 Organization of the Paper

The rest of the paper is organized as follows: Sect. 2 introduces the binarized convolutional deep neural network (CNN); Sect. 3 introduces the neuron pruning in the fully connected (FC) layer on the CNN; Sect. 4 shows the proposed architecture; Sect. 5 shows the experimental results; and Sect. 6 concludes the paper.

2. Binarized Convolutional Deep Neural Network (CNN)

Figure 3 shows a typical convolutional deep neural network (CNN), which consists of sequential layers including a convolutional layer, a pooling layer, and a fully connected (FC) layer. In the paper, we assume that the pretrained CNN is given, and the goal is to realize only inference with high-performance and small hardware. In this section, we briefly introduce a binarized CNN. Then, we will show a specific VGG-11 for the CIFAR-10 image classification task [5], which is our target.

2.1 Convolutional Layer

Both the convolutional and FC layers are variations of an artificial neural network (ANN). Figure 4 shows a model of an ANN, which is calculated as follows:

$$Z = f_{act}(\sum_{i=0}^{n} W_i X_i), \tag{1}$$

where Z is the output, X_i is the input, W_i is the weight, and n is the number of input. Note that, when $i = 0$, we assume that $X_0 = 1$. In that case, W_0 becomes a constant value, and it is called a bias, which adjusts the output of a neuron to keep a recognition accuracy. f_{act} denotes the activation function, such as the rectified linear unit (ReLU), the hyperbolic tangent function, and the sigmoid function.

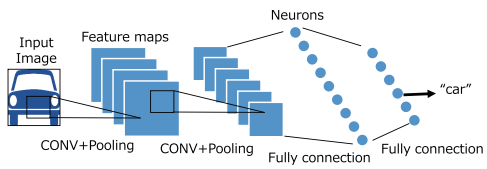

Fig. 3 Typical convolutional deep neural network (CNN).

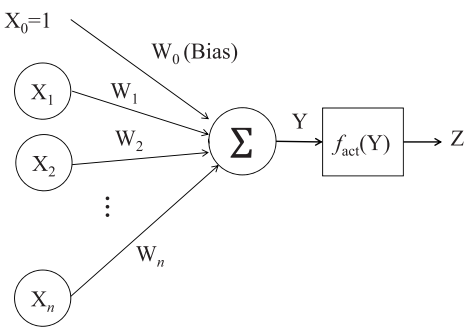

Fig. 4 Model of an artificial neural network.

As shown in Fig. 5, a 2D convolutional layer in the CNN is similarly to the FC layer. It applies the ANN operation to the $K \times K$ size kernel on the feature map, where K denotes the kernel size. It greatly reduces the number of parameters involved, allows local features, and avoid the over-fitting. Let l be the layer index. The output $Z_{l,r,c}$ of the i^{th} convolutional layer, which takes an input N_i images (feature maps) of dimension $K \times K$ at location (r, c), is calculated as follows:

$$Z_{l,r,c} = f_{act}(\sum_{s=0}^{N_i} \sum_{j=0}^{K} \sum_{l=0}^{K} W_{i,j,l,s} X_{i-1,s,r+j,c+k}), \tag{2}$$

where $K \times K$ are the dimensions of the kernel for the convolution operation.

2.2 Binarized CNN

Courbariaux et al. developed two types of the binarized CNN [8]. The first version is only weight binarized, while the later version is both weights and inputs are binarized. Similar works [20], [25], [35] consider full binarized CNN, however, their binarized one drops the recognition accuracy compared with floating-point precision CNNs. For instance, for the best-case ImageNet top-1 accuracies of 43% for full binarization and 53% for partial binarization.

Courbariaux et al. did not drop the accuracy for the binarized CNN, since they used a batch normalization technique, which reduces the information lost during low-precision by linearly shifting and scaling the dataset distribution to keep zero mean and unit variance. Thus, it covers binarization error compared to an arbitrary input distribution. It reported the considerable accuracy on the MNIST, SVHN, and CIFAR-10 tasks. However, since the normalization is also necessary during an inference, it becomes a bottleneck and requires additional hardware. Since it is only

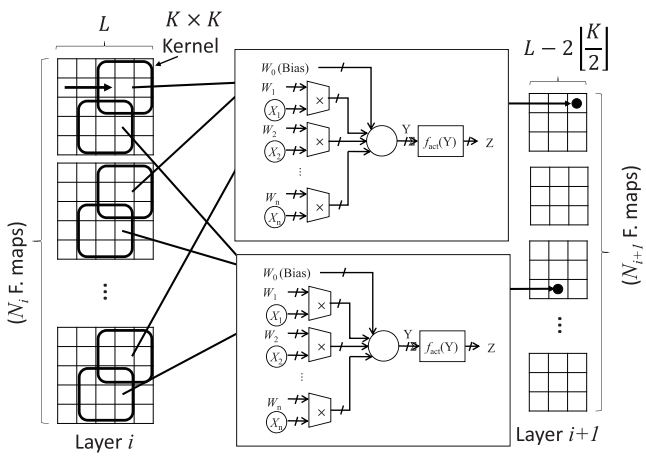

Fig. 5 Example of a convolutional layer.

Table 1 Truth table for a binarized $(-1/+1)$ multiplication $y = w \times x$.

w	x	$w \times x$
−1	−1	+1
−1	+1	−1
+1	−1	−1
+1	+1	+1

Table 2 Truth table for a binarized (0/1) multiplication $y = \overline{w \oplus x}$.

w	x	$\overline{w \oplus x}$
0	0	1
0	1	0
1	0	0
1	1	1

necessary to calculate the sign bit of the activation function, a normalization operation is done by integered bias [31]. Thus, in the hardware implementation, we use the integered bias instead of the binarized one.

Table 1 shows a truth table for a binarized $(-1/+1)$ multiplication, while Table 2 shows that for (0/1) assigned binarized one. In that case, the multiplication is performed by the XNOR circuit. Thus, in the binarized convolutional operation is calculated by

$$z_{l,r,c} = f_{sgn}(\sum_{s=0}^{N_i}\sum_{j=0}^{K}\sum_{l=0}^{K}\overline{w_{i,j,l,s} \oplus x_{i-1,s,r+j,c+k}}), \quad (3)$$

where $f_{sgn}(Y)$ denotes a binarized activation function as follows:

$$f_{sgn}(Y) = \begin{cases} 1 & (if\ Y \geq 0) \\ 0 & (otherwise) \end{cases} \quad (4)$$

In a similar way, in the binarized FC layer is calculated by

$$z = f_{sgn}(\sum_{l=0}^{n}\overline{w_l \oplus x_l}), \quad (5)$$

Suppose that $Y = \sum_{l=0}^{n}\overline{w_l \oplus x_l}$, Expr. (5) denotes a majority function, such as $f_{major}(Y)$. The majority function produces "1" when more than half of the inputs are 1, otherwise, it produces "0" when more than half the inputs are 0.

2.3 VGG-11 CNN for the CIFAR-10 Image Classification Task

The CIFAR-10 dataset [5] consists of 60,000 color images of 32×32 pixels, and the images are categorized into 10 classes (i.e., airplane, truck, cat, horse, etc.) and labels have already given. Table 3 shows specifications for the original VGG-11 benchmark CNN [26], which is widely used in the computer vision system, and it contains layers. The basic layers consist of multiple 2D convolution layers with $K = 3$ and max-pooling layers, while the rear layers consist of fully connected neural networks. First, it receives a normalized 32×32 image, which consists of 8-bit RGB color data.

Table 3 Specifications for the original VGG-11 [26].

Layer	Output Dim.	Input # Fmaps	Output Fmaps
Conv1	32× 32	3	64
Conv2	32× 32	64	64
Max Pool	16× 16	64	64
Conv3	16× 16	64	128
Conv4	16× 16	128	128
Conv5	16× 16	128	128
Max Pool	8× 8	128	128
Conv6	8× 8	128	256
Conv7	8× 8	256	256
Conv8	8× 8	256	256
Max Pool	4× 4	256	256
FC1	1× 1	4096	4096
FC2	1× 1	4096	4096
FC3	1× 1	4096	10

3. Threshold Neuron Pruning

3.1 Definition

In the paper, we propose the threshold neuron pruning instead of the edge pruning. Figure 6 shows that a model for the neuron pruning. Suppose that a target neuron is connected to n incoming edges with weight $W_{in,k}$ and m outgoing edges with weight $W_{out,k}$, where k denotes the index variable. If all the incoming edges and the outgoing ones of a neuron are eliminated, it means the neuron pruning itself. Experimentally, the edge pruning eliminates more edges than the neuron pruning. In Sect. 5.3, we show the experimental result. However, since the edge pruning randomly eliminates edges, it is not suitable for the hardware realization, which requires sequential memory access. On the other hand, since the neuron pruning eliminates all the incoming and outgoing edges, it maintains the sequentially memory access of weights. Thus, it is suitable for the hardware realization.

First, we define the neuron pruning.

Definition 3.1: A neuron pruning eliminates all the incoming and outgoing edges for a neuron.

In the paper, we propose a threshold neuron pruning.

Definition 3.2: A threshold neuron pruning performs the neuron pruning when the sum of the input weights or that of outputs is lower than the threshold.

There are various decisions of thresholds for the neuron pruning. In the paper, the threshold neuron pruning is performed, when one of the following conditions is satisfied:

1. $\sum_{k=1}^{n}|w_{in,k}| < \mu_i \times n$
2. $\sum_{k=1}^{m}|w_{out,k}| < \mu_o \times m$,

where $w_{in,k}$ denotes the k-th weight for the incoming edge, $w_{out,k}$ denotes the k-th weight for the outgoing one, μ_i denotes the threshold for the incoming edge, and μ_o denotes

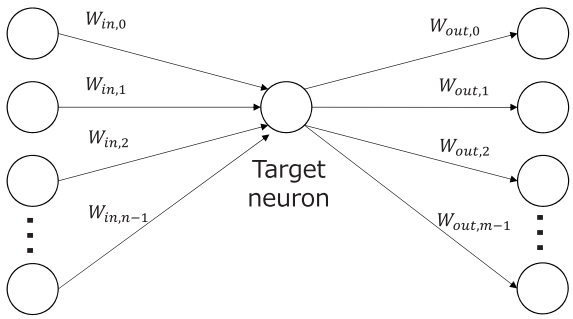

Fig. 6 Model of a neuron pruning.

that for the outgoing edge (Fig. 6). In this paper, different thresholds are used for incoming edges and outgoing ones.

3.2 Retraining for the Threshold Neuron Pruning

The neuron pruning is similar to the Synaptic pruning. The human brain has the process of pruning inherently. Five times synapses are pruned away from infant age to adulthood [16]. A similar rule can be applied to artificial neural networks. The neuron pruning proved to be a valid way to reduce the network complexity and overfitting [17]. This method works on modern neural networks as well. We begin by training the connectivity via the normal binarized network. Next, we prune the small-weight connections by the proposed neuron pruning. As a result, all connections with weights below a threshold are removed from the network. Finally, we retrain the network to learn the final weights for the remaining sparse connections. By applying pruning-retraining step by step, the number of neurons tends to be saturated. In the experiment, we will show this for the binarized fully connection layers.

4. Binarized CNN Architecture for the FPGA Implementation

4.1 Shared XNOR-MAC Circuit with Streaming Operation

Although we used the binarized MAC operation instead of the floating-point one, it consumes much hardware to realize the fully parallel XNOR-MAC operation. Since the typical CNN has the different number of feature maps in the layer, a heterogeneous streaming architecture requires many LUTs for a large size of XNOR operations.

In the paper, to realize the high-performance with less hardware, we proposed a shared XNOR-MAC circuit supporting a streaming operation as shown in Fig. 7. To reduce the memory access, we use the shift register to make a streaming data flow from the memory for the feature map. Also, it shares the different size of XNOR-MAC circuit into a single bitwise XNOR circuit followed by adder-trees, bias adder, and a write controller. The circuit reads the corresponding inputs from the shift register, then it applies to the bitwise binarized MAC operation. Next, it adds the pre-

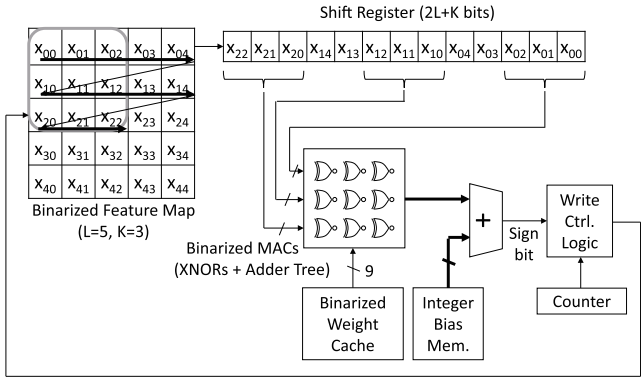

Fig. 7 Streaming binarized 2D convolutional circuit.

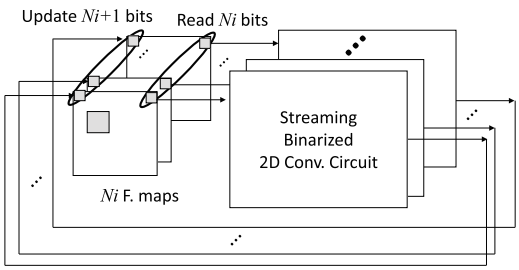

Fig. 8 Shared streaming binary 2D convolutional circuit.

computed bias, which is obtained by both the pre-trained bias and the batch normalization value. Since the kernel crosses the boundary of the feature map, we attach the write control logic to the output of the circuit.

To further increase the performance, we propose the shared streaming binary 2D convolutional circuit shown in Fig. 8. To flexibility access to all feature maps, multiple on-chip BRAMs are used to realize multi-port with wide band memory access speed. In contract, to read the weight, we use the off-chip memories, since the convolutional operation reads it at intervals for each feature map. Since we use the binarized CNN, the memory size is drastically reduced compared with non-binarized one. Since our CNN eliminates internal FC layers, the weight memories also eliminated.

4.2 Circuit for the Binarized Fully Connected Layers After Threshold Neuron Pruning

Figure 9 shows the serial-input parallel-output (SIPO) fully connected layer [11]. As shown in Fig. 9, it can reduce the memory bandwidth for the primary input. To realize the SIPO fully connected layer, it requires the sequentially multiply accumulation (MAC) circuit to emulates the artificial neuron shown in Fig. 4 sequentially. Figure 10 shows a sequential MAC circuit for the binarized neural network. It consists of the register and the XNOR gate to realized the binarized multiplication. Initially, it reset the value of the register to the bias value. Then, it updates the value for the neuron with performing the MAC operation sequentially. Finally, it sends the sign bit to the external output to realize the sign activation function. The MAC operation is realized

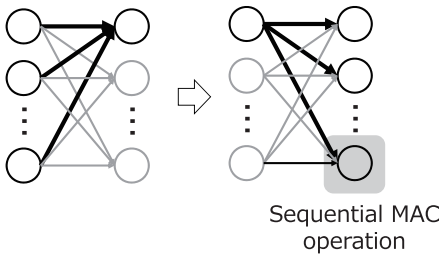

Fig. 9 Serial-input Parallel-output (SIPO) fully connected layer [11].

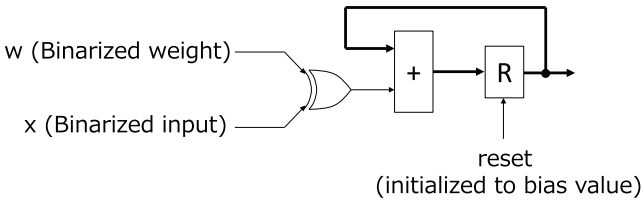

Fig. 10 Sequential multiply accumulation (MAC) circuit for the binarized neural network.

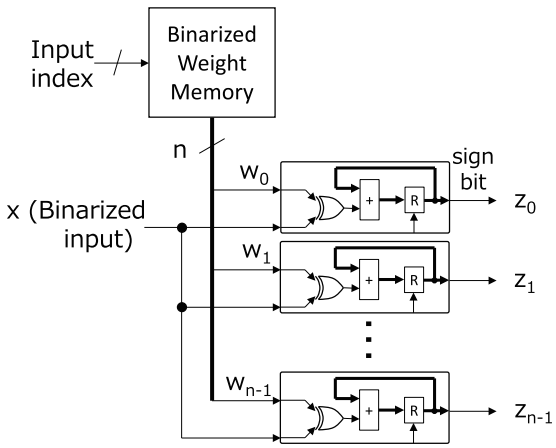

Fig. 11 Circuit for a SIPO binarized fully connected layer.

by the DSP slice on the FPGA. Figure 11 shows the circuit for the SIPO fully connected layer. In the circuit, **the binarized weight memory** stores the weight value, and it is read for corresponding input x_i. The sequential MAC circuit updates the value for neurons sequentially. Figure 12 shows the circuit for SIPO fully connected layers with the threshold neuron pruning. The most of the weights are eliminated by the neuron pruning, and only a few part of weights is packed in the weight memories. Since the FPGA can realize the appropriate size of the memory with the block RAMs (BRAM) and the distributed memories, it is suitable to realize the neuron pruning. All the weights for each layer are read, and the output neurons are updated at a time. After all the inputs are evaluated, it transfers the values for the output neurons to the shift register. Then, the next layer is evaluated by shifting the value of the shift register. When all the layers are evaluated, the values for the output neurons are sent to the external output.

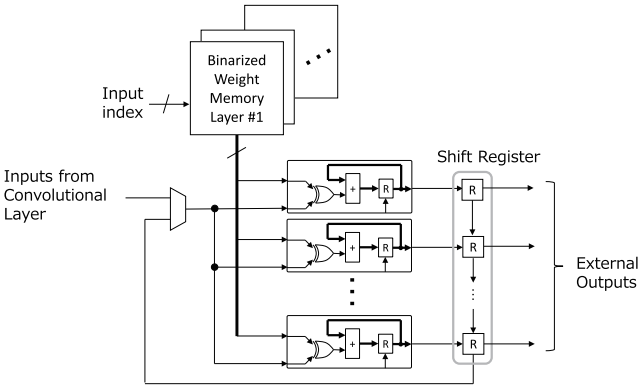

Fig. 12 Circuit for SIPO binarized fully connected layers with the threshold neuron pruning.

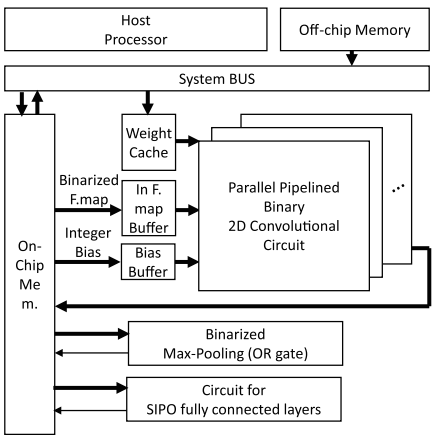

Fig. 13 Overall architecture.

4.3 Overall Architecture

Figure 13 shows the architecture for the proposed binarized CNN. The memory access circuit is almost the same as the conventional one, however, the memory part is realized by the on-chip block RAMs (BRAM). The proposed architecture has the shift registers and buffers to access indices for the corresponding kernel. In our implementation, we realize the binarized multiplier by an XNOR gate, while requires no DSP blocks for the convolutional operation. The on-chip memories (BRAMs) stores inputs, and outputs for all the feature maps, and the off-chip memories (DDR3SDRAM) stores weights. Thus, our architecture efficiently uses of the BRAMs while it saves additional memory and keeps the performance. Also, our implementation achieves higher computation speed than conventional one, since it performs a convolutional operation for 256 feature maps at a time.

5. Experimental Results

5.1 Threshold Neuron Pruning with Re-Training

We designed the CNN using a Chainer which is a deep neu-

Table 4 Number of neurons after the neuron pruning. Note that, we retain the 99% baseline accuracy.

Step	1	2	3	4	5	6	7	8	9	10	11	12	13
FC 1	4,096	2,259	1,578	1,458	1,457	1,457	1,457	1,454	1,454	1,454	1,454	1,454	1,454
FC 2	4,096	3,853	3,826	3,754	3,716	3,716	3,534	3,456	3,456	3,447	3,426	3,421	3,395
FC 3	4,096	3,438	1,149	1,059	498	373	193	102	89	57	54	51	37
FC 4	10	10	10	10	10	10	10	10	10	10	10	10	10
Total	12,298	9,560	6,563	6,281	5,681	5,556	5,194	5,022	5,009	4,968	4,944	4,936	4,896
Ratio	1.00	0.78	0.53	0.51	0.46	0.45	0.42	0.41	0.41	0.40	0.40	0.40	0.39

Table 5 Threshold values for each steps.

Step		1→2	2→3	3→4	4→5	5→6	6→7	7→8	8→9	9→10	10→11	11→12	12→13
FC 1	μ_i	0.3168	0.3375	0.3430	0.3470	0.3500	0.3508	0.3547	0.3500	0.3530	0.3530	0.3530	0.3525
	μ_o	0.3143	0.3278	0.3401	0.3547	0.3650	0.3650	0.3753	0.3750	0.3753	0.3751	0.3762	0.3769
FC 2	μ_i	0.2206	0.2412	0.2310	0.2430	0.2000	0.3117	0.2920	0.2200	0.2640	0.2686	0.2500	0.2848
	μ_o	0.2298	0.3551	0.3654	0.3988	0.4145	0.4270	0.4424	0.4463	0.4480	0.4465	0.4450	0.4461
FC 3	μ_i	0.0020	0.3900	0.1550	0.3410	0.1458	0.3950	0.4540	0.3500	0.5410	0.1700	0.2450	0.5600
	μ_o	—	—	—	—	—	—	—	—	—	—	—	—

Table 6 Comparison of binarized VGG-11 CNNs (Note that, the integer convolutional layer (IConv1) uses 1 bit weight and 8 bit input.).

Layer	Baseline				Neuron Pruning with Retraining			
	Output Dim.	Input # Fmaps	Output Fmaps	Weight [bits]	Output Dim.	Input # Fmaps	Output Fmaps	Weight [bits]
IConv1	32× 32	3	64	1.7K	32× 32	3	64	1.7K
BConv2	32× 32	64	64	36.8K	32× 32	64	64	36.8K
Max Pool	16× 16	64	64		16× 16	64	64	
BConv3	16× 16	64	128	73.7K	16× 16	64	128	73.7K
BConv4	16× 16	128	128	147.4K	16× 16	128	128	147.4K
Max Pool	8× 8	128	128		8× 8	128	128	
BConv5	8× 8	128	256	294.9K	8× 8	128	256	294.9K
BConv6	8× 8	256	256	589.8K	8× 8	256	256	589.8K
Max Pool	4× 4	256	256		4× 4	256	256	
BFC1	1× 1	4096	4096	16.7M	32× 32	1454	3395	4.4M
BFC2	1× 1	4096	4096	16.7M	32× 32	3395	37	125.6K
BFC3	1× 1	4096	10	40.9K	32× 32	37	10	370
(fc total)				(33.6M)				(4.5M)
Total				34.7M				5.7M

ral network framework [3], and the target task is the CIFAR-10 [5] which is an image recognition task. In the experiment, we set an appropriate threshold μ by manually, and applied the threshold neuron pruning for each fully connected layer. Then, we retrained the sparse fully connected layer. We repeated above processes step by step.

Table 4 shows the number of neurons after n reduction steps, and Table 5 shows the threshold values for each reduction steps. Since we cannot eliminate the last layer, we did not consider μ_o for the out-going edge. Note that, in the neuron pruning, when we applied inappropriate threshold to neuron pruning, the accuracy drastically decreased. Thus, we carefully set the threshold value by hands. Generally, when the number of neurons decreases, then recognition accuracy also decreases. In the comparison, we measured the number of neurons for the original one to keep 99% baseline accuracy compared with the accuracy for the original one. By applying pruning-retraining step by step, the number of neurons tends to be saturated. In the experiment, step 13 may reach to the saturated value, thus, we stop the pruning process. From Table 4, the number of neurons decreased by 39.8%. In our experiment, for the VGG-11 CNN, we can realize the weight memory for the fully connected layer by the on-chip memory on the FPGA. In that case, since it reads weights with a width bandwidth memory access, it can operate the fully connected layer with a high-speed. Also, since it requires no extra off-chip memory, it reduces the power consumption and costs.

Compared with the previous work [13], which applies the neuron pruning only once, the proposed pruning-retraining process can reduce more neurons, and it can be realized without extra hardware.

5.2 Implementation Results

We implemented the binarized CNN for the VGG-11 on the Xilinx Inc. Zedboard, which has the Xilinx Zynq FPGA (XC7Z020, 53,200 Slices, 106,400 FFs, 280 18Kb BRAMs, 220 DSP48Es). We used the Xilinx Inc. SDSoC 2016.4 to generate the bitstream with timing constraint 143 MHz. Our implementation used 15,680 LUTs, 64 18Kb BRAMs, and no DSP48Es. Also, it satisfied the timing constraint for real-time applications. To obtain the delay time, we used 32×32 pixel image set from the CIFAR10 benchmark, then

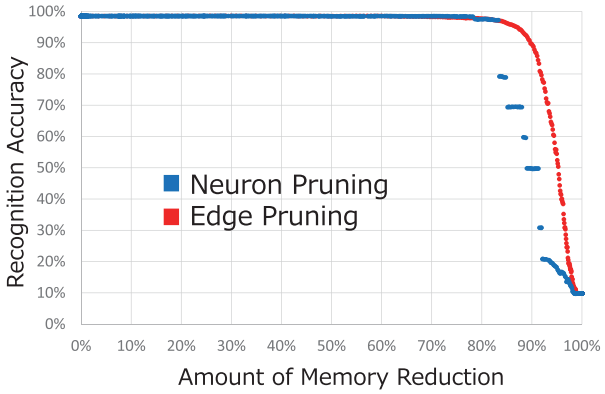

Fig. 14 Relationship between recognition accuracy and memory size reduction rate.

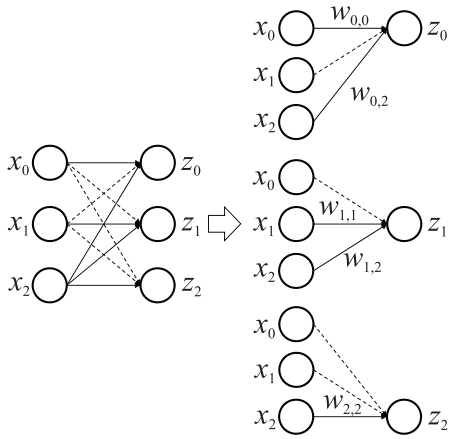

Fig. 15 Example of partition of edge pruning fully connection layer.

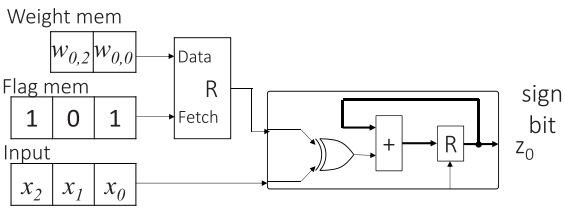

Fig. 16 Sequential MAC circuit for edge pruning.

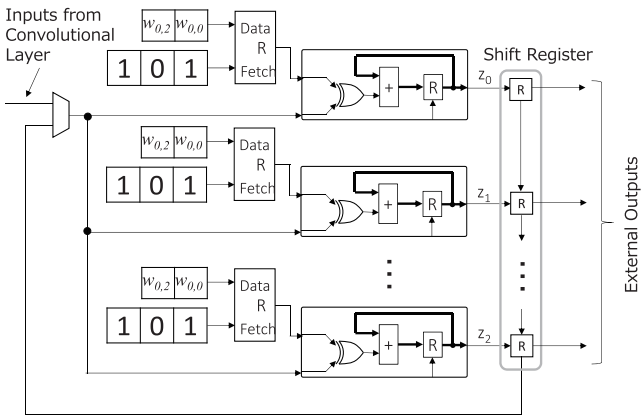

Fig. 17 Parallel realization of a sequential MAC circuit for edge pruning.

we measured a latency for one image. Since the measured delay time for our CNN was 2.456 msec, its performance was 407 (frames per second). We measured the total board power consumption: It was 2.2 W. Thus, the performance per power efficiency is 185.0 (FPS/Watt), the performance per LUT is 260.2×10^{-4} (FPS/LUT), and the performance per BRAM is 6.38 (FPS/BRAM), respectively.

5.3 Compared with an Edge Pruning Method

We compare the proposed neuron pruning with the edge pruning. We define the edge pruning.

Definition 5.3: An edge pruning performs the edge pruning when the weight is lower than the threshold value.

In the paper, we manually set the threshold value. We apply pruning-retraining step by step to the edge pruning. Figure 14 shows a relationship between recognition accuracy and memory size reduction rate, and Table 8 shows a comparison of the number of edges. From the experiment, the edge pruning can reduce more edge than the neuron pruning, however, it is almost the same value to keeping 99% of baseline accuracy.

We show the circuit for the edge pruning fully connection layer. We partition the edge pruning fully connection layer into single output neural networks (NNs) as shown in Fig. 15. Figure 16 shows a sequential MAC circuit for each NN. In the circuit, the binarized weight is loaded into the register, and the MAC operation is performed sequentially. Figure 17 shows a parallel realization of a sequential MAC circuit. It computes each single output NNs in parallel.

Let n be the number of inputs, m be the number of outputs, and $\sum w$ be the total number of edges for the edge pruning applied fully connection layer. From Fig. 16, it computes all single output NNs by n steps. As shown in Fig. 12, it also computes by n steps. Thus, both the circuit for the edge pruning and that for the neuron pruning take the same steps. Also, we quantitatively analysis the amount of memory. From Fig. 17, the flag memory requires nm bits, while the binarized weight memory requires $\sum w$ bits. Thus, the edge pruning circuit requires totally $nm + \sum w$ bits. On the other hand, from Fig. 12, it requires nm bits. Thus, as for the memory size, the circuit for the neuron pruning is smaller than that for the edge pruning. Table 9 compared of the circuit for pruning methods. From Table 9, the neuron pruning circuit is smaller than the edge pruning one. Above discussion, as for the circuit point of view, the neuron pruning is better than the edge pruning.

5.4 Compared with Conventional Binarized CNN Implementations

Table 7 compares binarized CNN implementations on the same FPGA. Although the FINN [31] implemented the VGG16 CNN on the Xilinx zcu102 board (Zynq Ultrascale+ MPSoC), to do fair comparison, we used the open source de-

Table 7 Comparison with other binarized CNN realizations on the FPGA.

Implementation (Year)	Zhao et al. (2017)	FINN (2017)	Ours
FPGA Board (FPGA)	Zedboard (XC7Z020)	PYNQ board (XC7Z020)	Zedboard (XC7Z020)
Clock (MHz)	143	166	143
# LUTs	46900	42823	15680
# 18Kb BRAMs	94	270	64
# DSP Blocks	3	32	0
Test Error	12.27%	19.9%	18.2%
Time [msec]	5.94	2.24	2.45
(FPS)	(168)	(445)	(408)
Power [W]	4.7	2.5	**2.2**
FPS/Watt	35.7	178.0	**185.0**
FPS/LUT	35.8×10^{-4}	103.9×10^{-4}	$\mathbf{260.2}\times10^{-4}$
FPS/BRAM	1.8	1.6	**6.38**

Table 8 Comparison of the number of edges.

Layer	Baseline	Neuron Pruning	Edge Pruning
FC 1	4096	1454	1312
FC 2	4096	3395	3256
FC 3	4096	37	36
FC 4	10	10	10

Table 9 Comparison of the circuit for a pruning method.

Method	Neuron Pruning	Edge Pruning
Clock (MHz)	143	143
# LUTs	15680	16320
# 18Kb BRAMs	64	96
# DSP Blocks	0	0
Test Error	18.2%	18.2%
Time [msec]	2.45	2.45
(FPS)	(408)	(408)
Power [W]	2.2	2.3

sign in Github to realize on the Xilinx PYNQ board which has the same FPGA used in other designs. From Table 7, compared with Zhao's implementation [34], the classification accuracy was almost the same, as for the performance per power efficiency (FPS/Watt), it is 5.18 times better, Compared with the FINN, the memory efficiency was 3.98 time better, and the performance per power efficiency is almost the same. Thus, our design achieves power efficiency CNN, since all the circuits are operated on chip primitives.

5.5 Comparison with the CPU and the GPU

We compared our binarized CNN with other embedded platforms. We used the NVidia Jetson TX1 board which has both the embedded CPU (ARM Cortex-A57) and the embedded GPU (Maxwell GPU). Following the benchmarking [23], the CPU and GPU run the VGG11 using Caffe [2] version 0.14. Also, we measured the total power consumption. Note that, in the experiment, to measure the latency, we set the number of batch size to one.

Table 10 compares our FPGA implementation with other platforms. Compared with the ARM Cortex-A57, it was 1773.0 times faster, it dissipated 3.1 times lower power, and its performance per power efficiency was 5781.3 times better. Also, compared with the Maxwell GPU, it was 11.1 times faster, it dissipated 7.7 times lower power, and its performance per power efficiency was 84.1 times better. Thus, the binarized CNN on the FPGA is suitable for the embedded system.

Table 10 Comparison with embedded platforms with respect to the VGG16 forwarding (Batch size is 1).

Platform	Embedded CPU	Embedded GPU	FPGA
Device	ARM Cortex-A57	Maxwell GPU	Zedboard FPGA
Clock Freq.	1.9 GHz	998 MHz	143 MHz
Memory	16GB eMMC Flash	4GB LPDDR4	5.0 Mb 18KbBRAM
Time [msec]	4210.0	27.23	**2.45**
(FPS)	(0.23)	(36.7)	**(408.1)**
Power [W]	7	17	2.2
Efficiency [FPS/W]	0.032	2.2	**185.0**

6. Conclusion

In the paper, we proposed the threshold neuron pruning which eliminates almost part of the weight memory, which was a bottleneck of the conventional realization. By applying the threshold neuron pruning, we could realize the weight memory by on-chip memory on the FPGA. Thus, it operated with a high-speed memory access. In the paper, we showed the SIPO fully connected layer circuit, which is efficiently access to on-chip memories on the FPGA. In the comparison, we measured the number of neurons for the original CNN, as for the 99% baseline accuracy, the number of neurons decreased by 39.8%. We implemented the neuron pruning CNN on the Xilinx Zedboard. Compared with the ARM Cortex-A57, it was 1773.0 times faster, it dissipated 3.1 times lower power, and its performance per power efficiency was 5781.3 times better. Also, compared with the Maxwell GPU, it was 11.1 times faster, it dissipated 7.7 times lower power, and its performance per power efficiency was 84.1 times better. Thus, the binarized CNN on the FPGA is suitable for the embedded system.

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. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," Computer Research Repository (CoRR), Dec., 2015. https://arxiv.org/ftp/arxiv/papers/1512/1512.08571.pdf
[2] Caffe: Deep learning framework, http://caffe.berkeleyvision.org/
[3] Chainer: A powerful, flexible, and intuitive framework of neural networks, http://chainer.org/
[4] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A

dynamically configurable coprocessor for convolutional neural networks," Annual Int'l Symp. on Computer Architecture (ISCA), pp.247–257, 2010.
[5] The CIFAR-10 data set, http://www.cs.toronto.edu/~kriz/cifar.html
[6] D.C. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," In Proc. CVPR, 2012.
[7] CUDA-Convent2: Fast convolutional neural network in C++/CUDA, https://code.google.com/p/cuda-convnet2/
[8] M. Courbariaux, I. Hubara, D. Soudry, R.E. Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1," Computer Research Repository (CoRR), March 2016, http://arxiv.org/pdf/1602.02830v3.pdf
[9] J. Donahue, L.A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," In Proc. CVPR, 2015.
[10] A. Dundar, J. Jin, V. Gokhale, B. Martini, and E. Culurciello, "Memory access optimized routing scheme for deep networks on a mobile coprocessor," HPEC2014, pp.1–6, 2014.
[11] C. Farabet, C. Poulet, J.Y. Han, and Y. LeCun, "CNP: An FPGA-based processor for convolutional networks," FPL2009, pp.32–37, 2009.
[12] C. Farabet, B. Martini, P. Akselrod, S. Talay, Y. LeCun, and E. Culurciello, "Hardware accelerated convolutional neural networks for synthetic vision systems," ISCAS2010, pp.257–260, 2010.
[13] T. Fujii, S. Sato, H. Nakahara, and M. Motomura, "An FPGA Realization of a Deep Convolutional Neural Network using a Threshold Neuron Pruning," Int'l Symp. on Applied Reconfigurable Computing (ARC2017), pp.268–290, 2017.
[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," In Proc. CVPR, 2014.
[15] I.J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and Vi. Shet, "Multi-digit number recognition from street view imagery using deep convolutional neural networks," arXiv prprint arXiv: 1312.6082, 2013.
[16] S. Han, H. Mao, and W.J. Dally, "Deep Compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," ICLR2016, pp.1–14, 2016.
[17] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R.R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," arXiv:1207.0580, 2012.
[18] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," IEEE Trans. Pattern Anal. Mach. Intell., vol.35, no.1, pp.221–231, 2013.
[19] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," In Proc. CVPR, pp.1725–1732, 2014.
[20] M. Kim and P. Smaragdis, "Bitwise neural networks," CoRR, abs/1601.06071, 2016.
[21] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proc. IEEE, vol.86, no.11, pp.2278–2324, 1998.
[22] V. Nair and G.E. Hinton, "Rectified linear units improve restricted Boltzmann machines," ICML, pp.807–814, 2010.
[23] https://github.com/charlyng/Embedded-Deep-Learning/tree/master/Benchmark-Performance
[24] M. Peemen, A.A.A. Setio, B. Mesman, and H. Corporaal, "Memory-centric accelerator design for convolutional neural networks," ICCD2013, pp.13–19, 2013.
[25] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," https://arxiv.org/pdf/1603.05279.pdf
[26] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," ICLR2015, pp.1–14, 2015.
[27] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," In Proc. CVPR, pp.1701–1708, 2014.
[28] Theano, http://deeplearning.net/software/theano/
[29] Torch: A scientific computing framework for LUTJIT, http://torch.ch/
[30] A. Toshev and C. Szegedy, "Deeppose: Human pose estimatiion via deep neural networks," In Proc. CVPR, 2014.
[31] Y. Umuroglu, N.J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," ISFPGA, 2017. Source code for the Xilinx PYNQ board: https://github.com/Xilinx/BNN-PYNQ
[32] 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," FPGA2016, pp.26–35, 2016.
[33] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," FPGA2015, pp.161–170, 2015.
[34] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, "Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs," ISFPGA, pp.15–24, 2017.
[35] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients," http://arxiv.org/pdf/1606.06160v2.pdf

Tomoya Fujii received the B.S. degrees from Tokyo Institute of Technology (Tokyo Tech), in 2016. Now, he is a master stduent at Tokyo Institute of Technology, Japan. His current research interests include reconfigurable architecture and deep learning.

Shimpei Sato received the B.S., M.S., and Ph. D. degrees in engineering from Tokyo Institute of Technology (Tokyo Tech), in 2007, 2009, and 2014. He is currently an Assistant Professor with the Department of Information and Communications Engineering of Tokyo Tech. From 2014 to 2016, he worked in High performance computing area as a post doctoral researcher, where he investigated an application performance analysis/tuning method. His current research interests include approximate computing realization by architecture design and circuit design and their applications.

Hiroki Nakahara received the B.E., M.E., and Ph.D. degrees in computer science from Kyushu Institute of Technology, Fukuoka, Japan, in 2003, 2005, and 2007, respectively. He has held research/faculty positions at Kyushu Institute of Technology, Iizuka, Japan, Kagoshima University, Kagoshima, Japan, and Ehime University, Ehime, Japan. Now, he is an associate professor at Tokyo Institute of Technology, Japan. He was the Workshop Chairman for the International Workshop on Post-Binary ULSI Systems (ULSIWS) in 2014, 2015, 2016 and 2017, respectively. He served the Program Chairman for the International Symposium on 8th Highly-Efficient Accelerators and Reconfigurable Technologies (HEART) in 2017. He received the 8th IEEE/ACM MEMOCODE Design Contest 1st Place Award in 2010, the SASIMI Outstanding Paper Award in 2010, IPSJ Yamashita SIG Research Award in 2011, the 11st FIT Funai Best Paper Award in 2012, the 7th IEEE MCSoC-13 Best Paper Award in 2013, and the ISMVL2013 Kenneth C. Smith Early Career Award in 2014, respectively. His research interests include logic synthesis, reconfigurable architecture, digital signal processing, embedded systems, and machine learning. He is a member of the IEEE, the ACM, and the IEICE.