An FPGA Realization of a Deep Convolutional Neural Network Using a Threshold Neuron Pruning

Tomoya Fujii[1], Simpei Sato[1], Hiroki Nakahara[1](✉), and Masato Motomura[2]

[1] Tokyo Institute of Technology, Meguro, Japan
nakahara.h.ad@m.titech.ac.jp
[2] Hokkaido University, Sapporo, Japan

Abstract. 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. In this paper, we propose a neuron pruning technique which eliminates almost part of the weight memory. In that case, the weight memory is realized by an on-chip memory on the FPGA. Thus, it achieves a high speed memory access. In this paper, we propose a sequential-input parallel-output fully connection layer circuit. 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 89.3% with keeping the 99% accuracy. We implemented the fully connected layers on the Digilent Inc. NetFPGA-1G-CML board. Comparison with the CPU (ARM Cortex A15 processor) and the GPU (Jetson TK1 Kepler), as for a delay time, the FPGA was 219.0 times faster than the CPU and 12.5 times faster than the GPU. Also, a performance per power efficiency was 125.28 times better than CPU and 17.88 times better than GPU.

1 Introduction

1.1 Convolutional Deep Neural Network (CNN)

Recently, for embedded computer 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. For example, a human face recognition [22], a human and object detection [12], a human pose estimation [25], a string recognition in a scene [13], a road traffic sign recognition [6], a sport scene recognition [16], a human action recognitions [8,15], are reported. These 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

S. Wong et al. (Eds.): ARC 2017, LNCS 10216, pp. 268–280, 2017.
DOI: 10.1007/978-3-319-56258-2_23

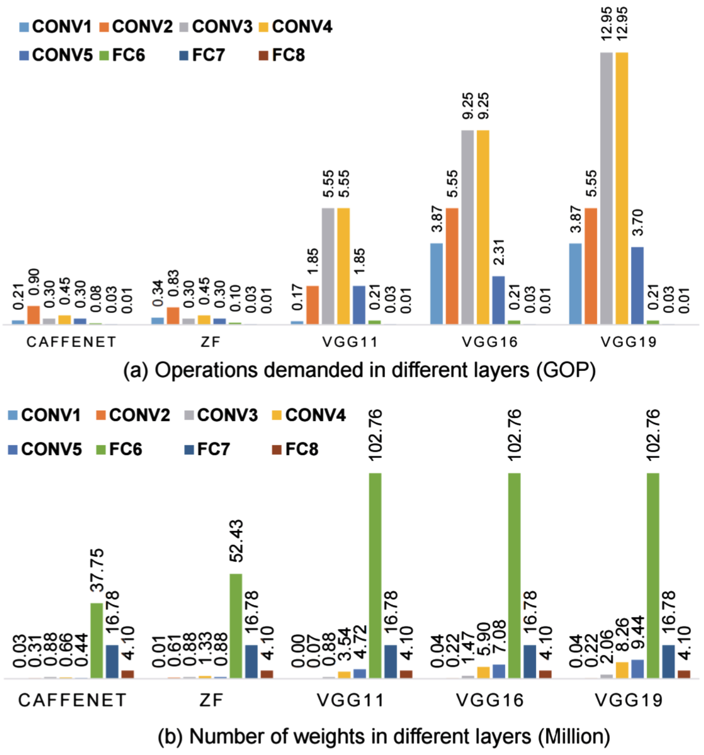

Fig. 1. The complexity distribution of state-of-the-art CNN models: (a) distribution of operations by theoretical estimation; (b) distribution of weight number [26].

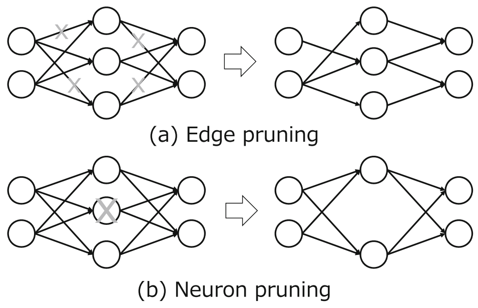

Fig. 2. Comparison of pruning techniques.

requirement of the embedded vision system, since the existing system using a CPU is too slow, the acceleration of the CNN is necessary [17]. Most software-based CNNs use the GPUs [2,3,7,23,24]. Unfortunately, since the GPU consumes much power, they are unsuitable for the embedded system [9]. 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 [11]. 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 [9] reported that, as for the performance per power, the FPGA-based CNN is about 10 times more efficient than the GPU-based one.

1.2 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 [26]. 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. In the paper, we focus on the solving the latter part, that is, we propose the memory reduction techniques to realize them on-chip memories on the FPGA. Figure 2(a) shows an example of edge pruning of the fully connected layer. The memory access of the fully connected layer is a sequentially reading of the weights that are indexed to the corresponding edges. Thus, by pruning edges, the amount of memory can be reduced. In the conventional techniques, the randomly pruning techniques of edges have been proposed [1,14]. However, in the hardware realization point of view, since the memory access of the sequential address is suitable, the random edge pruning may cause a performance degradation.

1.3 Proposed Method

In the paper, we propose a neuron pruning instead of the edge pruning. 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 remainder 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 realized 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.

1.4 Contributions of the Paper

Contributions of the paper are 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.

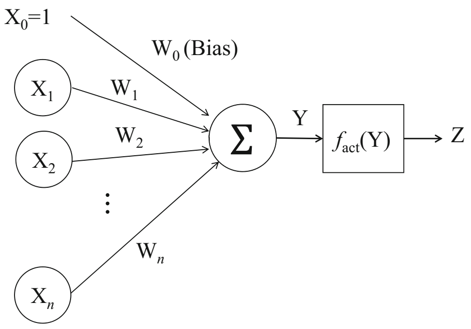

Fig. 3. Artificial neuron.

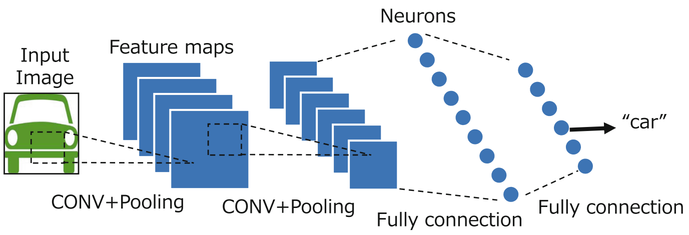

Fig. 4. Example of the convolutional neural network (CNN).

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 realized a wide band of the memory access. Our technique is a complementary to the conventional techniques that accelerates 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. Comparison with the CPU (ARM Cortex A15 processor) and the GPU (Jetson TK1 Kepler), as for a delay time, the FPGA was 219.0 times faster than the CPU and 12.5 times faster than the GPU. Also, a performance per power efficiency was 125.28 times better than CPU and 17.88 times better than GPU.

1.5 Organization of the Paper

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

2 Convolutional Deep Neural Network (CNN)

2.1 Artificial Neural Network

Let n be a bit precision, $x_i, y_i, w_i, z_i \in \{0, 1\}$ be binary variables, $X = (x_0, x_1, \ldots, x_n)$ be the input, $Y = (y_0, y_1, \ldots, y_n)$ be the internal variable, $W = (w_0, w_1, \ldots, w_n)$ be the weight, f_{act} be the activation function, and $Z = (z_0, z_1, \ldots, z_n)$ be the output. Note that, in this paper, a capital letter denotes an integer, while a small letter denotes a binary value. Figure 3 shows a circuit for **an artificial neuron (AN)**. The following expression shows an operation for the AN:

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

where X_0 is a constant one and W_0 denotes **a bias** which corrects the deviation of the given data. Typically, the activation function is realized by a sigmoid, a tanh, a ReLU [18], and so on. In the paper, we use the ReLU function which is suitable to a hardware realization. **A convolutional deep neural network (CNN)** has multiple **layers**. Figure 4 shows an example of the CNN. The typical layer consists of **a 2D convolutional layer**, **a pooling layer**, and **a classification layer**. Each layer consists of multiple **feature maps**. To recognize the input image, first, the feature map reacts corresponding subdivided training data by 2D convolutional layers with pooling layers. Then, the classifier selects the appropriate reactions from feature maps. Usually, the classifier is realized by the fully connected neural network. 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. Figure 5 shows the 2D convolution operation. It computes the output by shifting

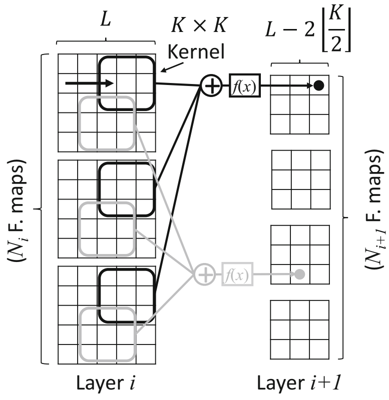

Fig. 5. Convolutional operation.

Table 1. Specifications for the VGG-11 [21].

Layer #	Type	Kernel size	# Feat. maps
1	Conv + maxpool	3	64
2	Conv + maxpool	3	128
3	Conv	3	256
4	Conv + maxpool	3	256
5	Conv	3	512
6	Conv + maxpool	3	512
7	Conv	3	512
8	Conv + maxpool	3	512
9	FC	1	4096
10	FC	1	4096
11	FC	1	1000

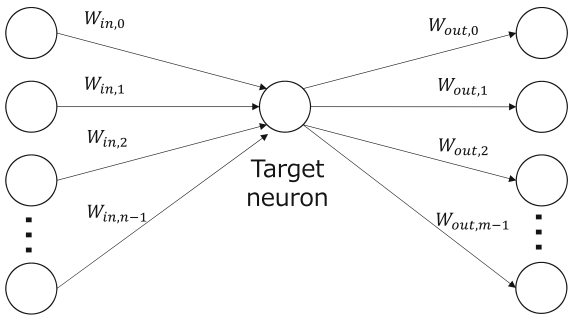

Fig. 6. Model of a neuron pruning.

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:

$$Y_{i+1,x,y} = \sum_{k=0}^{N_i-1} (\sum_{m=0}^{K-1} \sum_{n=0}^{K-1} X_{k,x+m,y+n} W_{k,m,n}) \tag{1}$$
$$Z_{i+1,x,y} = f_{act}(Y_{i+1,x,y}).$$

In the 2D convolutional operation, Z is mapped to (x, y) at the output feature map $i + 1$. In the fully connected layer, $L_i = 1$ and $K_i = 1$. By inserting the non-linear and low-imaging operations into the convolution layers, we can reduce the number of computations in the convolution layers, while we can obtain the movement invariance. We call this **a pooling operation**, which can be realized by a simple circuit. In this paper, we implement the max-pooling operation. Its operation can be realized by a comparator for selecting the maximum value in the kernel. It is much smaller than the 2D convolution operation circuit.

2.2 VGG-11 CNN

Table 1 shows specifications for the VGG-11 benchmark CNN [21], which is widely used in the computer vision system. The VGG-11 consists of 11 layers. The basic layers consist of multiple 2D convolution (Conv) layers with $K = 3$ and max-pooling (maxpool) layers, while the rear layers consist of fully connected (FC) layers. First, it receives a normalized 32×32 image, which consists of 8-bit RGB color data.

Almost CNN researches have been proposed to improve the performance, power consumption for the convolutional layer only on the CNN [4,20,27]. Only a few work [26] tried to improve both the convolutional layer and the fully connected layer. In this paper, we consider a high-speed and a low-power circuit for the fully connected layer with a neuron pruning technique. Our technique can be applied to the previous work.

3 Threshold Neuron Pruning

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. Therefore, generally, the edge pruning eliminates more edges than the neuron pruning. However, since the edge pruning randomly eliminates edges, it is not suitable for the hardware realization, which requires sequentially memory

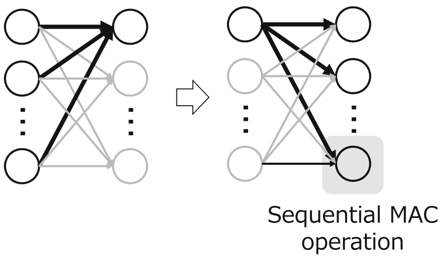

Fig. 7. Serial-input parallel-output (SIPO) fully connected layer [10].

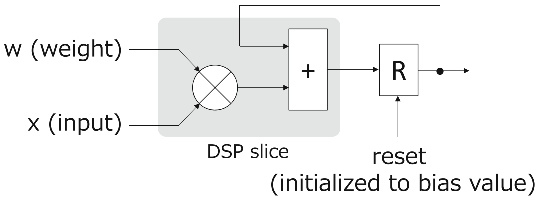

Fig. 8. Sequential multiply accumulation (MAC) circuit.

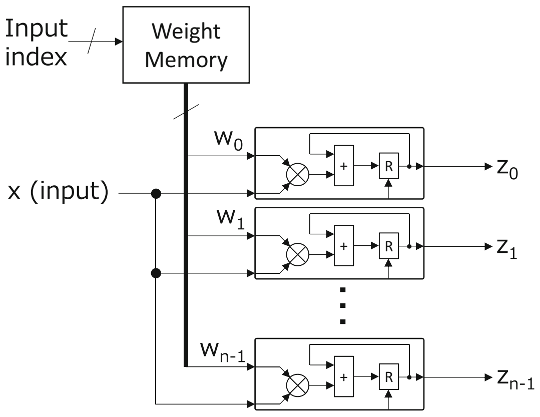

Fig. 9. Circuit for a SIPO fully connected layer.

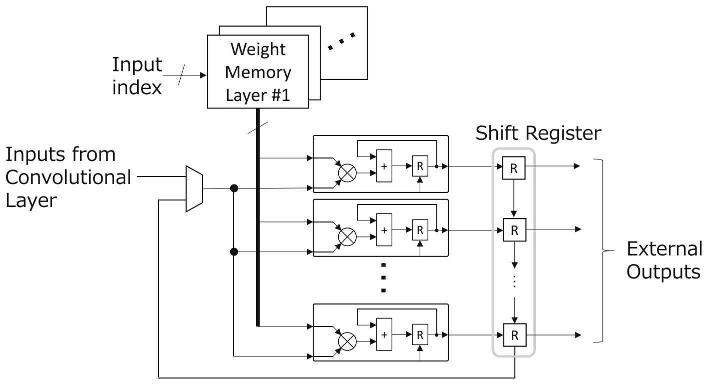

Fig. 10. Circuit for SIPO fully connected layers with the threshold neuron pruning.

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:

Table 2. Number of neurons corresponding to the accuracy (%).

Layer #	Original	99% accuracy	95% accuracy
1	4096	941	891
2	4096	354	74
3	4096	31	31
4	10	10	10
Total	12298	1336	1006
Ratio	1.000	0.107	0.082

Table 3. Number of 18 KB BRAMs corresponding to the accuracy (%).

Layer #	Original	99% accuracy	95% accuracy
1–2	7280	145	29
2–3	7280	5	1
3–4	18	1	1
Total	14578	151	31
Ratio	1.000	0.010	0.002

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

4 Circuit for the Fully Connected Layers After Threshold Neuron Pruning

Figure 7 shows the serial-input parallel-output (SIPO) fully connected layer [10]. As shown in Fig. 7, it can reduces 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. 3 sequentially. Figure 8 shows a sequential MAC circuit, which consists of the MAC unit and the register. Initially, it reset the value for the register to the bias value. Then, it updates the value for the neuron with performing the MAC operation sequentially. Finally, it sends the value to the external output. The MAC operation is realized by the DSP slice on the FPGA. Figure 9 shows the circuit for the SIPO fully connected layer. In the circuit, **the 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 10 shows the circuit for SIPO fully connected layers with the threshold neuron pruning. The most of weights is 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 transfer the values for the output neurons to the shift register. Then, the next layer is evaluated by shifting the value for the shift register. When all the layers are evaluated, the values for the output neurons are send to the external output.

5 Experimental Results

5.1 Threshold Neuron Pruning

We designed the CNN using a Chainer which is a deep neural 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.

Table 2 compared the number of neurons for each fully connected layer. Note that, generally, when the number of neurons decreases, then recognition accuracy also decreases. In the comparison, we measured the number of neurons for the original CNN, the 99% accuracy, and the 95% accuracy compared with the accuracy for the original one. From Table 2, as for the 99% accuracy, the number of neurons decreased by 89.3%, while as for the 95% accuracy, it decreased by 91.8%. Table 3 compared the number of 18 Kb BRAMs for each fully connected layer. From Table 3, as for the 99% accuracy, the number of BRAMs decreased by 99.0%, while as for the 95% accuracy, it decreased by 99.8%. Let n_i be the number of incoming edges for each layer, n_o be that of outgoing edges, and w be the bit precision (in the experiment, we used 8-bit). Since the amount of weight memory for each layer is $n_i n_o w \simeq O(n^2)$, the neuron pruning exponentially reduces the amount of memory. In our experiment, for the VGG-11 CNN, we can realized 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 band-width 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.

5.2 FPGA Implementation

We applied the threshold neuron pruning with the 99% accuracy. Then, we implemented the fully connected layers on the Digilent Inc. NetFPGA-1G-CML evaluation board (It has a Xilinx Inc. Kintex 7 XC7K325T FPGA: 50,950 slices, 890 18 Kb BRAMs, and 840 DSP slices). We used the Xilinx Inc. Vivado 2016.2 with timing constrain 100 MHz. Our implementation used 4,241 Slices, 151 18 Kb BRAMs, and 145 DSP slices. Also, it satisfied the timing constraint for real-time applications. The delay time for the fully connected layer was 29.0 usec.

We measured the power consumption without that for the power sources on the board: It was 7 W. Since the implemented fully connected layer operated with 29.0 usec delay time, its performance was 34482.7 (images/usec). Thus, the performance per power efficiency is 4926.10.

Table 4. Comparison with the CPU and the GPU.

Device	CPU	GPU	FPGA
Platform	Jetson TK1		NetFPGA 1G-CML
Device	Cortex A15@2.5 GHz	Kepler@950 MHz	Kintex 7@100 MHz
Delay time [usec]	6354	363	29
Performance [images/usec]	157.3	2754.8	34482.7
Power [W]	4	10	7
Performance/power	39.32	275.48	4926.10

5.3 Comparison with the CPU and the GPU

We applied the threshold neuron pruning fully connected layer, we compared with the CPU and the GPU. As for the CPU, we used the ARM Corp. Cortex A15 running at 2.5 GHz on the nVidia Corp. Jetson TK1 evaluation board, while that for the GPU, we used the nVidia Corp. Kepler running at 950 MHz, which has 192 CUDA cores on the same board. As for software based one, we used Ubuntu 14.04 LTS as an operating system, and used framework was the Chainer. Table 4 compared the FPGA realization with the CPU and the GPU ones. From Table 4, the delay time for the CPU was 6,354 usec, and its power consumption was 4 W, while the delay time for the GPU was 363 usec, and its power consumption was 10 W. Note that, we measured the power consumption excepting for the standby power consumption. The experimental results showed that as for the delay time, the FPGA realization was 291.0 times faster than the CPU one, and it was 12.5 times faster than the GPU one. As for the performance per power efficiency, the FPGA realization was 125.28 times better than the CPU, and it was 17.88 times better than the GPU one.

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% accuracy, the number of neurons decreased by 76.4%, while as for the 95% accuracy, it decreased by 91.7%. That is, as for the 95% accuracy, the number of BRAMs decreased by 96.2%, while as for the 95% accuracy, it decreased by 99.7%. We implemented the neuron pruning fully connected layer on the Digilent Inc. NetFPGA-1G-CML FPGA board, and compared with the ARM Cortex A15 processor and the Kepler GPU. As for a delay time, the FPGA was 219.0 times faster than the CPU and 12.5 times faster than the GPU. Also, a performance per power efficiency was 125.28 times better than CPU and 17.88 times better than GPU.

The future project is to apply the pruning technique to the binarized CNN [19].

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. Anwar, S., Hwang, K., Sung, W.: Structured pruning of deep convolutional neural networks. Computer Research Repository (CoRR), December 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. Chakradhar, S., Sankaradas, M., Jakkula, V., Cadambi, S.: A dynamically configurable coprocessor for convolutional neural networks. In: Annual International Symposium on Computer Architecture (ISCA), pp. 247–257 (2010)
5. The CIFAR-10 data set. http://www.cs.toronto.edu/kriz/cifar.html
6. Ciresan, D.C., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. In: Proceedings of CVPR (2012)
7. CUDA-Convent2: Fast convolutional neural network in C++/CUDA. https://code.google.com/p/cuda-convnet2/
8. Donahue, J., Hendricks, L.A., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., Darrell, T.: Long-term recurrent convolutional networks for visual recognition and description. In: Proceedings of CVPR (2015)
9. Dundar, A., Jin, J., Gokhale, V., Martini, B., Culurciello, E.: Memory access optimized routing scheme for deep networks on a mobile coprocessor. In: HPEC 2014, pp. 1–6 (2014)
10. Farabet, C., Poulet, C., Han, J.Y., LeCun, Y.: CNP: an FPGA-based processor for convolutional networks. In: FPL 2009, pp. 32–37 (2009)
11. Farabet, C., Martini, B., Akselrod, P., Talay, S., LeCun, Y., Culurciello, E.: Hardware accelerated convolutional neural networks for synthetic vision systems. In: ISCAS 2010, pp. 257–260 (2010)
12. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of CVPR (2014)
13. Goodfellow, I.J., Bulatov, Y., Ibarz, J., Arnoud, S., Shet, V.: Multi-digit number recognition from street view imagery using deep convolutional neural networks (2013). arXiv preprint: arXiv:1312.6082

14. Han, S., Mao, H., Dally, W.J.: Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. In: ICLR 2016 (2016)
15. Ji, S., Xu, W., Yang, M., Yu, K.: 3D convolutional neural networks for human action recognition. IEEE Trans. Pattern Anal. Mach. Intell. **35**(1), 221–231 (2013)
16. Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., Li, F.: Large-scale video classification with convolutional neural networks. In: Proceedings of CVPR, pp. 1725–1732 (2014)
17. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
18. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: ICML, pp. 807–814 (2010)
19. Nakahara, H., Yonekawa, H., Sasao, T., Iwamoto, H., Motomura, M.: A memory-based realization of a binarized deep convolutional neural network. In: The International Conference on Field-Programmable Technology (FPT 2016), pp. 273–276 (2016)
20. Peemen, M., Setio, A.A.A., Mesman, B., Corporaal, H.: Memory-centric accelerator design for convolutional neural networks. In: ICCD 2013, pp. 13–19 (2013)
21. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: ICLR 2015 (2015)
22. Taigman, Y., Yang, M., Ranzato, M., Wolf, L.: DeepFace: closing the gap to human-level performance in face verification. In: Proceedings of CVPR, pp. 1701–1708 (2014)
23. Theano. http://deeplearning.net/software/theano/
24. Torch: A scientific computing framework for LUTJIT. http://torch.ch/
25. Toshev, A., Szegedy, C.: DeepPose: human pose estimatiion via deep neural networks. In: Proceedings of CVPR (2014)
26. Qiu, J., Wang, J., Yao, S., Guo, K., Li, B., Zhou, E., Yu, J., Tang, T., Xu, N., Song, S., Wang, Y., Yang, H.: Going deeper with embedded FPGA platform for convolutional neural network. In: FPGA 2016, pp. 26–35 (2016)
27. Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B., Cong, J.: Optimizing FPGA-based accelerator design for deep convolutional neural networks. In: FPGA 2015, pp. 161–170 (2015)