# ReNA: A Reconfigurable Neural-Network Accelerator for AI Edge Computing

Yasuhiro Nakahara*, Motoki Amagasaki*, Qian Zhao†, Masato Kiyama*, Masahiro Iida*

*Graduate School of Science and Technology,
†Graduate School of Computer Science and Systems Engineering
*Kumamoto University, †Kyushu Institute of Technology
*Kumamoto, Japan 860-8555, †Fukuoka, Japan 804-0015
nakahara@st.cs.kumamoto-u.ac.jp, amagasaki@cs.kumamoto-u.ac.jp, cho@ai.kyutech.ac.jp,
masato@cs.kumamoto-u.ac.jp, iida@cs.kumamoto-u.ac.jp

**Low power consumption is important in edge artificial intelligence (AI) chips, where power supply is limited. Therefore, we propose a reconfigurable neural-network accelerator (ReNA), an AI chip that can process convolutional and fully connected layers with the same structure by reconfiguring the circuit. We also develop tools for preevaluation of performance when a deep neural-network (DNN) model is implemented on ReNA. From this, we establish an implementation flow for DNN models on ReNA. Evaluating the performance of VGG16, we achieve 1.30 TOPS/W.**

## I. Introduction

In artificial intelligence (AI) edge computing, it is important to process deep neural networks (DNNs) at low power and low latency. Existing devices such as central processing units (CPUs) and graphics processing units (GPUs) are used on the edge side, but AI chips are nearly the only option for applications where power is extremely limited [1]. Many AI chips designed as application-specific integrated circuits for processing convolutional neural networks (CNNs) have been researched and developed in recent years [2]. These AI chips are highly power efficient because they are specifically designed for CNN structures. Some CNN accelerators focus only on convolutional layers, because fully connected layers, which require many parameters, are often not used in CNNs. However, a fully connected layer is required for performing regression. Therefore, CNN accelerators must also support fully connected layers.

Further, chip design alone is insufficient for actually using edge AI chips. It is important to quickly implement user models, so we need to construct a path from architectural exploration to implementation of CNN models.

In this paper, we propose a reconfigurable CNN accelerator with the following features:

- Our accelerator has a reconfigurable wiring structure. By changing connections between neighboring processing blocks, both convolution and fully connected operations can be efficiently performed on unique processing blocks.
- Many weights become zero due to edge pruning, allowing low power consumption by calculating only nonzero components in the fully connected layer.
- The numerical precision is 8-bit fixed-point operations, based on accurate estimates with a quantization library [3, 4].
- Predictable processing speed and power when using the simulator.

Low-power and low-latency CNN processing is thus realized by adopting a processing structure that is optimal for the inference model type. In addition, performance evaluation tools support rapid model implementation.

## II. Related works

Many AI chips for CNN are designed with a focus on the convolutional layer, which incurs significant computation. Eyeriss v1 [5] and Envison [6] are examples of AI chips for the convolutional layer. These AI chips process DNNs at high parallelism and low power by using data multiple times in the circuit. However, when these AI chips process fully connected layers, the transfer of huge weights becomes a bottleneck that deteriorates processing speed. ReNA transfers only nonzero weights to reduce weight transfer times, speeding up the processing of all coupled layers by transforming many weights to zero by pruning. Eyeriss v2 [7] is an AI chip capable of accelerating both the convolutional layer and the fully connected layer by pruning for both. However, this pruning requires retraining, which incurs high costs in terms of equipment and time. In contrast, pruning only the fully connected layer can transform many weights to zero without retraining. The peak throughputs of Eyeriss v2 and ReNA are 153.6 and 573.4 GOPS, respectively, showing that ReNA can

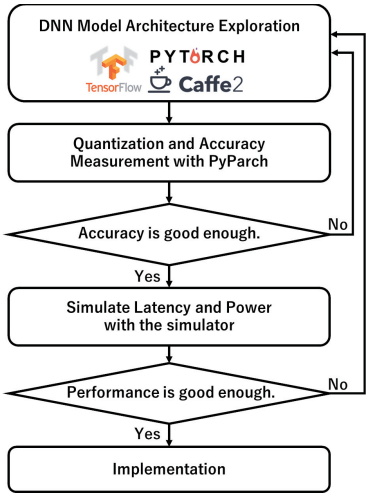


Fig. 1. Implementation flowchart.

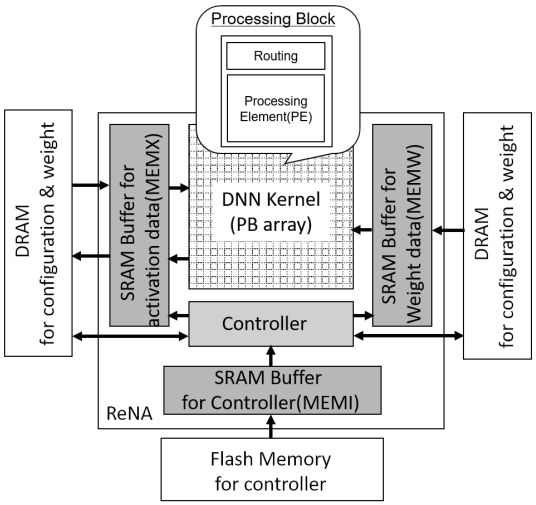


Fig. 2. Overview of the CNN accelerator.

process DNNs faster without retraining. Specialized AI chips for binary networks in which parameters are quantized to 1 bit are also widely studied[8, 9, 10, 11]. However, these approaches require retraining as well. Also, these AI chips refer only to the architecture, not to the implementation, of a DNN model running on them. Running a DNN model on an AI chip requires establishing an implementation flow, so in addition to the chip, we developed tools to support this implementation. Pyparch, a dedicated DNN library, allows users to measure correct accuracy when implementing a DNN model on ReNA. It is possible to measure accuracy when pruning is applied, thus ensuring accuracy. Dedicated simulators can estimate the performance of various DNN models on ReNA in advance, allowing users to quickly estimate performance when implementing a DNN model on ReNA.

## III. Proposed CNN accelerator

### A. Overview

When implementing a DNN model on an AI chip, it is necessary to verify whether the DNN model meets performance requirements. Therefore, we developed tools that allow preliminary evaluations of accuracy, processing speed, and power consumption. Figure 1 shows a flowchart of the process from architecture exploration of CNN models to their implementation. First, we make a CNN model using DNN frameworks. The DNN framework used can be anything that can export CNN models in the Open Neural Network Exchange (ONNX) format. Next, we quantize the CNN model and measure its accuracy after quantization using PyParch, a quantization library based on Pytorch that can accurately simulate post-quantization accuracy. If the simulated accuracy meets a criterion, we then simulate latency and power using a dedicated simulator. If latency and accuracy too meet their criteria, we proceed to implementation. By constructing a path to implementation, we can quickly implement a DNN model that satisfies requirements for accuracy and latency.

### B. ReNA architecture

#### B.1. Architecture overview

Figure 2 shows a block diagram for ReNA. ReNA has three SRAM types, those for the input (MEMX), weight (MEMW), and controller (MEMI). PB ARRAY is composed of $64 \times 64$ processing blocks (PBs) with multiply–accumulate (MAC) circuits. Each PB is connected only to its top, bottom, left, and right neighbor PBs (Figure 3). A torus network connects the upper PB to the lower PB, and connections between PBs are reconfigurable by the configuration register. The controller supports microcode instructions and generates control signals for different processing layers. The PB array size and memory size are determined so that AlexNet [12] can be implemented.

#### B.2. Processing block

Figure 3 shows the PB circuit structure and PB connections. The PB has two functions for activation data. One function receives activations from the four connected neighbor PBs, so the PB receives activations from its full direction set. Using these lines, the PB gets new activations for each calculation, while supplying activations for other PBs. This allows data sharing between PBs, reducing the amount of data transfer from SRAM. The direction of data reception can be dynamically reconfigured for

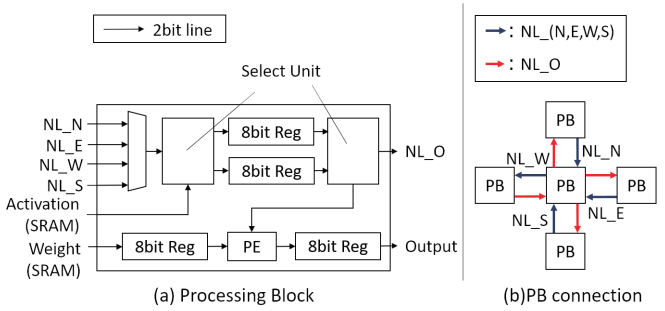


Fig. 3. Processing block circuit.

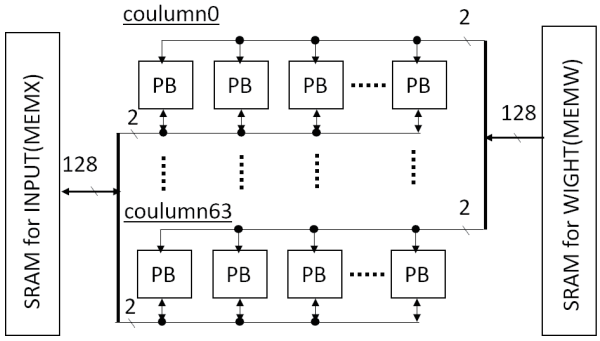


Fig. 4. Connection between SRAM and PB ARRAY.

each PB. This reconfiguration is performed layer-by-layer, and connections are changed based on layer hyperparameters.

The other function receives activations from SRAM during calculations. The PB has two registers for storing activations, one storing activations from PB connections and the other storing activations from SRAM. The select unit determines which register stores which activation. Thus, while one register supplies the MAC operator with operation activations, the other can supply the next activation from SRAM to the PB.

In addition, outputs can be transferred to SRAM along with calculations. Calculated data are stored in an 8-bit register after quantization to 8 bits and applying activation functions. This register operates independently of other processes and can transfer data to SRAM during calculations. Therefore, PB can independently perform activation transfers from SRAM, the operation, and the output transfer to SRAM, and so can always perform calculations.

### B.3. Data transfer

Figure 4 shows the connection between SRAM and PB ARRAY. SRAMs (MEMX and MEMW) and PB ARRAY are connected by a dedicated wire for each row. 8-bit input data and weights are serially transferred to PB ARRAY in 2-bit units. The size of the PB array is $64 \times 64$, and there are 64 PBs per column. The controller controls which PB receives the data. For activation, PBs receive data for each column. Each layer has a different way of receiving weight data. Sections B.4 and B.5 describe the specific method of transferring weights.

### B.4. Convolutional layer

Processing in the convolutional layer is expressed as

$$u_{ch_o,i,j} = \sum_{r=0}^{ch_i-1} \sum_{p=0}^{k-1} \sum_{q=0}^{k-1} w_{r,ch_o,p,q} \times x_{r,i+p,j+q} + b_{ch_o}, \quad (1)$$

where $ch_i$ is the number of channel inputs, $ch_o$ is the channel output, $i$ and $j$ are $x$- and $y$-axes of the input feature map, $k$ is the filter size, $w$ is the convolution coefficient, $u$ is the output feature map, and $b_{ch_o}$ is bias. In the convolutional layer, one datum is used to calculate multiple outputs. Therefore, ReNA reduces data transfer by sharing data with multiple PBs, allowing highly parallel operations with limited bandwidth.

Figure 5 shows processing in the convolutional layer for a $3 \times 3$ PB array. First, row activations for different channels are transferred to the PB array (Figure 5(a)). Note that the PB connection forms a circle. Next, transfers of weights and inputs between the PB and calculations are performed (Figure 5(b)). The same weights are transferred to the same rows in the PB array. Thus, different PBs can share the same weight and thereby reduce the cost of transmission. Each PB performs MAC operations using transferred weights and activations in the PB. At the same time, PB activations are transferred to the connected PBs. As Figure 5(c) shows, each PB is shifted one by one after the process in Figure 5(b). This procedure is repeated until the activation returns to the order (Figure 5(a)). This allows activations to be reused in the PB array for the length of the ring. This process is performed multiple times if the filter size $k$ is larger than 1 or if the input channels are larger than the array size. After the operation, outputs of one row of different channels are generated (Figure 5(d)). This process is repeated until all outputs have been computed. When the image size or the number of input feature-map channels exceeds the array size, the operation is split into multiple operations. Thus, it can process a layer of any hyperparameter. When the image size of the input feature map is smaller than the array size, multiple lines of output can be simultaneously computed. Therefore, ReNA can process input feature maps of various sizes with high parallelism.

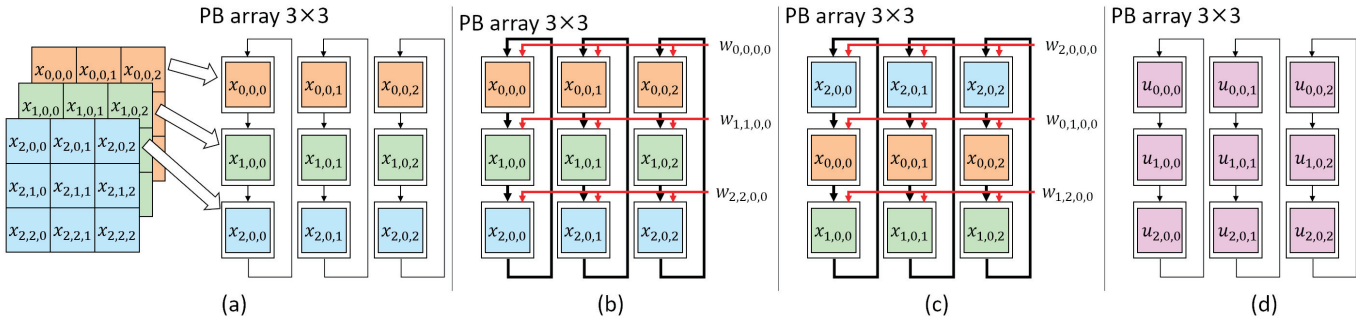


Fig. 5. Example of convolutional layer processing.

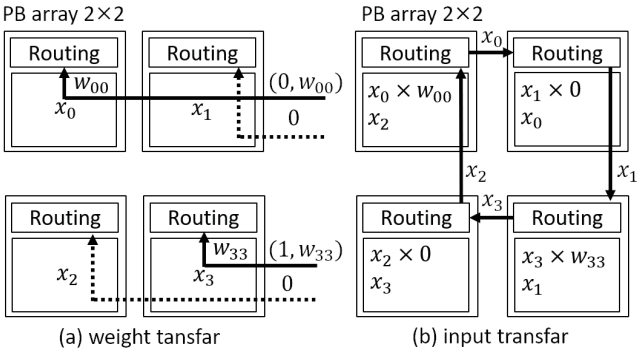


Fig. 6. Example of fully connected layer processing.

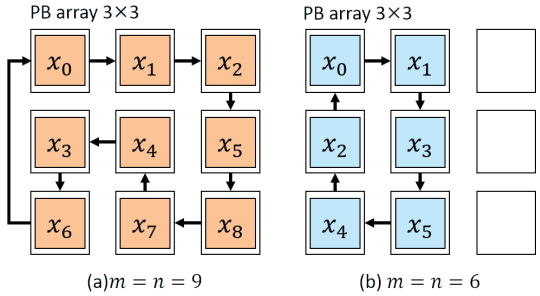


Fig. 7. Example of PB connections in a fully connected layer.

### B.5. Fully connected layer

Processing in a fully connected layer is represented as

$$u_n = \sum_{i=0}^{m-1} w_{i,n} \times x_i + b_n, \tag{2}$$

where $m$ is the number of input nodes, $n$ is the number of output nodes, $x$ is the input data, $w$ is the weight, $b$ is bias, and $u$ is the output for each node. The bottleneck in a fully connected layer is the transfer of enormous weights. Therefore, ReNA reduces the transfer time of weights by not transferring zeros. Note that many weights can be converted to zero by pruning weights, greatly reducing transfer times.

Figure 6 shows an example of data flow in a fully connected layer (with $m = n = 2$). First, activations are transferred in an arbitrary order. In the example, activations are transferred as shown in Figure 6(a). Next, the weights are transferred. A fully connected layer's weights have 6-bit addresses and 2-bit valid bits in addition to the weight value. The weight is transferred to the PBs specified by the address of each row in a single transfer. In the worst case, weight transfers are repeated for the number of rows of PBs performing the operation. As Figure 6(a) shows, zero weights are not transferred, so in this example weight transfers are cut in half. Next, ReNA transfers activations between the PBs and performs calculations. The PB connection connects the PB performing calculations to a daisy chain (Figure 6(b)). Each PB performs MAC operations using the transferred weight and PB activation. Activation in the PB is simultaneously transferred to connected PBs. This procedure is repeated until the activation returns to the first order (Figure 6(a)). Activation transfers from SRAM can thus be reduced, because activations can be shared within PB arrays. If $m$ and $n$ are less than the number of PBs, all operations can be performed in a single transfer.

ReNA can minimize processing times for fully connected layers by reconfiguring PB connections. Figure 7 shows PB connections for processing fully connected layers with different hyperparameters. In Figure 7(a) and (b), the shortest daisy chain is constructed by changing the PB connection, minimizing the number of calculations. If the PB connection in Figure 7(a) is used to process all the layers in Figure 7(b), the processing time worsens by about 1.5 times. Thus, the reconfigurable PB

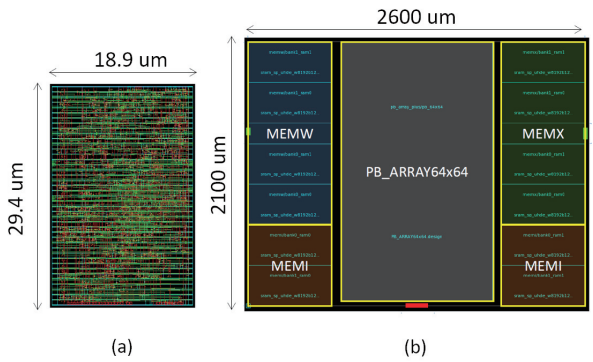


Fig. 8. ReNA layout capture.

TABLE I
Simulation results for processing speed and power.

| | | Mesured value | Simulated value |
|---|---|---|---|
| Latency [ms] | Conv | 2.487 | 2.154 |
| | FC | 12.265 | 12.265 |
| | Total | 14.752 | 14.779 (+0.18%) |
| Power [W] | Conv (1st layer) | 0.198 | 0.196 (-1.1%) |
| | FC (6th layer) | 0.098 | 0.116 (+11.8%) |

connection is an important element for processing fully connected layers.

### C. *Latency and power simulator*

#### C.1. ReNA design

We performed layout design with a TSMC 22-nm process standard cell. Figure 8 shows layout images for (a) a PB cell and (b) the external layout. MEMX, MEMW, and MEMI are each 524 KB. We used Synopsys PrimeTime P-2019.03-SP3 with the standard delay format to measure the delay of each component. The measurement results indicated a maximum frequency of 350 MHz.

#### C.2. Simulator

As described in Section A, preliminary evaluations of processing speed and power are important for selecting a DNN model, so we developed a simulator to estimate processing speed and power. This simulator estimates performance by reading DNN model parameters and weights from an ONNX file. Processing times can be estimated from the obtained DNN model hyperparameters. Power consumption is estimated from data acquired by a Synopsys PrimePower P-2019.03-SP3. In the convolutional layer, power is mostly consumed by the PB performing calculations. Therefore, the simulator calculates the number of PBs in each layer and estimates power consumption based on the acquired data. The simulator estimates power consumption other than that by the PB as a constant, because power consumption does not change nearly as much as do the DNN model's hyperparameters. As in the convolutional layer, power in the fully connected layer is mostly consumed by the PB that performs calculations. Unlike the convolutional layer, however, the number of PBs used for processing depends on the number of nonzero weights being transferred. The simulator thus estimates the number of PBs for performing calculations from the frequency at which nonzero weight data are transferred to the PBs. The number of nonzero weights is taken from the actual weight data. As in the convolutional layer, power consumption other than that by the PB is estimated as a constant. Table I shows estimated results for processing speed and power consumption when AlexNet is executed. The simulated and estimated results for latency are nearly identical. Power consumption in the convolutional layer, which accounts for most of the computation volume, is nearly identical to the results, and error in the fully connected layer is about 10%.

## IV. Evaluation

### A. *Conditions*

We evaluated ReNA processing speed and power consumption when running VGG16 [13]. Processing speed and power consumption were estimated using the simulator described in Section C.2. To evaluate the performance effects of weight pruning of fully connected layers, we also evaluated VGG16 with 70% of the weights pruned by a simple method in which all weights below a threshold are set to zero. The accuracy of VGG16 without and with pruning was 68.7% and 67.9%, respectively. Pruned models were not retrained.

### B. *Results*

We first evaluate the performance improvement by pruning. As the figure shows, pruning improved frames per second (FPS) and power efficiency by about 1.2 times without retraining. Retraining incurs costs in terms of time and equipment. Therefore, the ReNA method of processing fully connected layers is useful, because it improves performance without retraining.

We next compared the ReNA with other AI chips. In terms of convolutional layer performance, power efficiency is second only to Envision, which is a dedicated AI chip for the convolutional layer. ReNA power efficiency outperformed Eyeriss v2, although exact comparisons are not

TABLE II
LATENCY AND POWER ESTIMATES.

| | | Eyeriss v1 | Envision | Eyeriss v2 | This work | |
|---|---|---|---|---|---|---|
| Process [nm] | | 65 | 28 | 65 | 22 | |
| Freq. [MHz] | | 200 | 200 | 200 | 350 | |
| Dataset | | ImageNet[14] | | | | |
| DNN model | | VGG16 | | Sparse AlexNet | VGG16 | VGG16 (70% pruning) |
| FPS | Conv | 0.69 | 1.7 | 342.4 | 8.9 | 8.9 |
| | Overall | - | - | 278.7 | 6.1 | 7.4 |
| Power [mW] | Conv | 236 | 26 | 461 | 185 | 185 |
| | Overall | - | - | 419 | 166 | 175 |
| Power efficiency [TOPS/W] | Conv | 0.09 | 2.02 | - | 1.49 | 1.49 |
| | Overall | - | - | 0.96 | 1.13 | 1.30 |

possible due to differences in process and model. In addition, all layers must be pruned to maximize Eyeriss performance, which requires costly retraining of the model. The sparse AlexNet used in the Eyeriss evaluation is a model with significant pruning of all layer weights. In contrast, ReNA does not require relearning for the evaluation, thus achieving high performance at low cost.

## V. Conclusion

We proposed an AI chip, ReNA, that efficiently computes both the convolutional layer and the fully connected layer. We also proposed a tool for evaluating performance of the DNN model on ReNA in advance. We evaluated the performance of VGG16 without retraining using the proposed tool and achieved 1.30 TOPS/W.

## References


[1] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, J. Kepner, "Survey and benchmarking of machine learning accelerators," *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, 2019.

[2] Xiaofei Wang, Yiwen Han, Victor C. M. Leung, Dusit Niyato, Xueqiang Yan, Xu Chen, "Convergence of Edge Computing and Deep Learning: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials*, Vol. 2, Issue 2, pp. 869-904, 2020.

[3] K. Masato, N. Yasuhiro, A. Motoki, I. Masahiro, "A Quantized Neural Network Library for Proper Implementation of Hardware Emulation," *Proc. of 4th International Workshop on GPU Computing and AI*, Vol. 1, pp. 136-140, 2019.

[4] K. Masato, N. Yasuhiro, A. Motoki, I. Masahiro, "Deep Learning Framework with Arbitrary Numerical Precision," *2019 IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pp. 81-86, 2019.

[5] Y.-H. Chen, T. Krishna, J. Emer, V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, pp. 127-138, 2016.

[6] Bert Moons, Roel Uytterhoeven, Wim Dehaene, Marian Verhelst, "14.5 Envision: A 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable Convolutional Neural Network processor in 28nm FDSOI," *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 60, pp. 246-247, 2017.

[7] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, Vivienne Sze, "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, Vol. 9, Issue 2, pp. 292-308, 2019.

[8] K. Ando, K. Ueyoshi, K. Orimo, H. Yonekawa, S. Sato, H. Nakahara, M. Ikebe, T. Asai, S. Takamaeda-Yamazaki, M. Kuroda,T. Motomura, "BRein Memory: A 13-Layer 4.2 K Neuron/0.8M Synapse Binary/Ternary Reconfigurable In-Memor Deep Neural Network Accelerator in 65nm CMOS," *2018 IEEE Symposium on VLSI Circuits*, pp. C24-C25, 2017.

[9] D. Bankman, L. Yang, B. Moons, M. Verhelst, and B. Murmann, "Analways-on 3.8 μ J/86% CIFAR-10 mixed-signal binary CNN processorwith all memory on chip in 28nm CMOS," *2018 IEEE International Solid - State Circuits Conference*, pp. 222-224, 2018.

[10] H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma, "A Mixed-SignalBinarized Convolutional-Neural-Network Accelerator Integrating denseWeight Storage and Multiplication for Reduced Data Movement," *2018 IEEE Symposium on VLSI Circuits*, pp. 141-142, 2018.

[11] Shihui Yin, Zhewei Jiang, Jae-Sun Seo, Mingoo Seok, "XNOR-SRAM: In-MemoryComputing SRAM Macro for Binary/Ternary Deep Neural Networks," *IEEE Journal of Solid-State Circuits*, Vol. 55, Issue 6, pp. 1733-1743, 2018.

[12] Alex Krizhevsky, Ilya Sutskever, Geoffrey E,Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Proc. of Neural Information Processing Systems (NIPS), pp. 1097-1105, 2012

[13] Karen Simonyan, Andrew Zisserman, "VERY DEEP CONVOLU-TIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNI-TION," *The International Conference on Learning Representations (ICLR)*, 2015.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei Fei, "Imagenet: A large-scale hierarchical image database," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248-255, 2009.