Low Power Convolutional Neural Networks on a Chip

Yu Wang, Lixue Xia, Tianqi Tang, Boxun Li, Song Yao, Ming Cheng, Huazhong Yang
Dept. of E.E., Tsinghua National Laboratory for Information Science and Technology (TNList),
Tsinghua University, Beijing, China
e-mail: yu-wang@mail.tsinghua.edu.cn

***Abstract*—Deep learning, and especially Convolutional Neural Network (CNN), is among the most powerful and widely used techniques in computer vision. Applications range from image classification to object detection, segmentation, Optical Character Recognition (OCR), etc. At the same time, CNNs are both computationally intensive and memory intensive, making them difficult to be deployed on low power light-weight embedded systems. In this work, we introduce an on-chip convolutional neural network implementation for low-power embedded system. We point out that the high precision of weights limits the low-power CNN implementation on both FPGA and RRAM platform. A dynamic quantization method is introduced to reduce the precision while maintaining the same or comparable accuracy at the same time. Finally, the detailed designs of low-power FPGA-based CNN and RRAM-based CNN are provided and compared. The results show that FPGA-based design gets $2\times$ energy efficiency compared with GPU implementation, and the RRAM-based design can further obtain more than $40\times$ energy efficiency gains.**

I. Introduction

Image classification is a fundamental problem in computer vision area. In recent years, Convolutional Neural Network (CNN) has made great advances in image classification accuracy. In ImageNet Large-Scale Vision Recognition Challenge (ILSVRC) 2014 [1], GoogLeNet showed that CNN had great success in image classification by achieving the top-5 accuracy of 93.3% in classification task [2], which was significantly higher than other traditional image classification methods.

While achieving high performance, CNN-based methods demand much more computation and memory resources than traditional classification methods. For example, the VGG-16 model [3] needs 3.10×10^{10} operations to process one picture. As a result, most CNN-based algorithms and methods have to be processed on servers with plenty of resources. However, many applications on embedded systems demand capabilities of high-accuracy and real-time object recognition, such as auto-piloted car and robots. Therefore, an energy efficient method to implement CNN on embedded systems is highly demanded.

Field Programmable Gate Array (FPGA) and Application Specific Integrated Circuit (ASIC) are promising platforms for CNN accelerators, which can achieve considerable performance while significantly improving energy efficiency compared with commodity hardware. Many researchers have proposed various CNN accelerator designs [4]. However, weights in convolutional layers of CNN are used for multiple times in one inference process. Consequently, the overall performance is closely related to the efficiency of the memory system. If the number of memory access is too large, significant performance degradation and energy consumption increase will occur [5].

It is also a promising approach that using emerging metal-oxide resistive random-access memory (RRAM) to improve the energy efficiency of CNN for embedded applications. As two terminal devices, RRAM devices have ultra-integration density to support a large number of signal connections within a small footprint. Moreover, RRAM devices can be used to build resistive cross-point structure [6] named crossbar array, which can naturally transfer the weighted combination of input signals and realize the "matrix-vector multiplication" with incredible energy efficiency [7]. The convolution kernels in CNN can also be regarded as multiple vector-vector multiplications (or matrix-vector multiplication), which means that we can store the weights in the non-volatile RRAM cells and input the data into the crossbar to perform convolution kernel. In this way, RRAM-based design can reduce the energy consumption of computation and memory access. However, state-of-the-art RRAM devices can only store 8-bit weights [8], while the generally used CNN models need 16-bit or even 32-bit weights [3], which limits the implementation of RRAM-based CNN.

Therefore, the bottleneck of both FPGA-based CNN implementation and RRAM-based CNN implementation is the high-precision weights. For FPGA-based design, high-precision weights means we need high-precision multipliers, which costs lots of FPGA resources especially for large and deep CNNs. The large memory access also needs much energy consumption to load high-precision weights. For RRAM-based design, only 8-bit weights are available considering the limited bit-levels of RRAM devices caused by device variation [8].

In this paper, we compare two kinds of low-power CNN designs based on FPGA and RRAM. The contributions of this paper include:

1) We use dynamic quantization method [9] to reduce the precision of weights and data in CNN while maintaining the recognition accuracy. The results show that the precision can be compressed into 8-bit fix-point data with less than 2% accuracy reduction.
2) We compare two kinds of CNN designs based on FPGA [9] and RRAM using the quantized network. The results show that FPGA-based design gets $2\times$ energy efficiency compared with Graphics Processing Unit (GPU) implementation, and the RRAM-based design can further obtain more than $40\times$ energy efficiency gains.

The rest of this paper is organized as follows: Section II provides the background knowledge of CNN. Section III discusses compression method, and the two kinds of designs are shown in Section IV and Section V. The experimental results are depicted in Section VI and Section VII concludes this work.

II. Preliminaries of CNN

A typical CNN consists of a number of convolutional (Conv) layers and fully-connected (FC) layers [2]. Generally, a convolutional layer can further contain three main functions as Fig. 2(a) shows: convolution kernel, non-linear neuron, and spatial pooling. Therefore, there are four basic functions in CNN.

Convolution Kernel can be expressed as in Eq.1:

$$g(x,y,z)=\sum_{i=0}^{C-1}\sum_{j=0}^{C-1}\sum_{k=1}^{I}f(x+i,y+j,k)\cdot c_z(i,j,k)=\vec{f}\cdot\vec{c}_z \quad (1)$$

where the vector $\vec{f}$ and $\vec{g}$ respectively represent the input and output feature map in the form of 3D matrix; $\vec{c}_z$ is one convolution kernel

978-1-4799-5341-7/16/$31.00 ©2016 IEEE

with the size of $C \times C$; and I is the channel number of the convolution kernel and the input feature map.

Non-linear Neuron is an one-by-one function attached after the convolution kernel. Rectified Linear Unit (ReLU) function is the most common one, i.e. $h = \text{ReLU}(g) = \max(g, 0)$. Previous work [10] shows that the training convergence can be much faster in the network with ReLU neurons than that with other neurons.

Spatial Pooling is cascaded after the non-linear neurons. It merges the neighbor area of input feature map in order to reduce the data amount and keep the local invariance. Generally, there are two kinds of pooling functions in CNN, namely mean pooling and max pooling. Mean pooling averages all the inputs while the max pooling chooses the maximum value. Previous work [10] shows max pooling achieves better performance than mean pooling in the same CNN application, especially in deep CNN.

Fully-Connected Layers are the final layers that all inputs and outputs are connected by weights like traditional Artificial Neural Network (ANN).

In our CNN implementations, we choose ReLU function as non-linear neurons and max pooling as the method of spatial pooling.

III. Dynamic Quantization Method [9]

Using short fixed-point numbers instead of long floating-point numbers can reduce the computation resource consumption of FPGA platform and reduce the bandwidth requirements. RRAM-based design also needs to reduce the weight precision considering the device fabrication limit. Therefore, a quantization phase is introduced to convert 32-bit or even longer floating-point numbers of CNN model into 16-bit or 8-bit fixed-point numbers.

Specifically, the value of a fixed-point number can be expressed as:

$$n = \sum_{i=0}^{bw-1} B_i \cdot 2^{-f_l} \cdot 2^i \tag{2}$$

where bw is the bit width of the number and f_l is the fractional length which can be negative. However, directly using the same methods for quantization in different layers may reduce lots of recognition accuracy [9]. Therefore, a dynamic quantization method is used where f_l is static in the same layer while dynamic for different layers. First, we find the optimal f_l for **weight**s in one layer by a brute-force manner for optimization because the solution space is small:

$$f_{l,weight} = \underset{f_l}{\text{argmin}} \sum |W_{float} - W_{fix}(bw, f_l)| \tag{3}$$

where W is a weight and $W_{fix}(bw, f_l)$ represents the fixed-point format of W under the given bw and f_l. Then, an **data** quantization phase finds the optimal f_l of features between two layers by greedy comparing the the fixed-point CNN model and the floating-point CNN model:

$$f_{l,data} = \underset{f_l}{\text{argmin}} \sum |x^+_{float} - x^+_{fix}(bw, f_l)| \tag{4}$$

where x^+ represents the result of a layer that $x^+ = A \cdot x$. The effect of above quantization method is shown in Section VI, and more details can be find in [9].

IV. FPGA-based Design

A. Challenges

Though FPGA-based CNN accelerator can realize high parallelism by placing a large number of processing elements, the limited memory space and bandwidth still prevent further performance improvement. On the one hand, state-of-the-art CNN models for large scale visual recognition are much larger than early CNN models that are designed for simple tasks. In this case, CNN accelerators such as ShiDianNao [4] that store weights on chip are hard to support today's CNN models. Consequently, state-of-the-art CNN models can only be stored in external memory. On the other hand, storing CNN weights in external memory means extra efforts should be paid to load those weights. In this manner, the bandwidth may limit the accelerator performance on large-size layers.

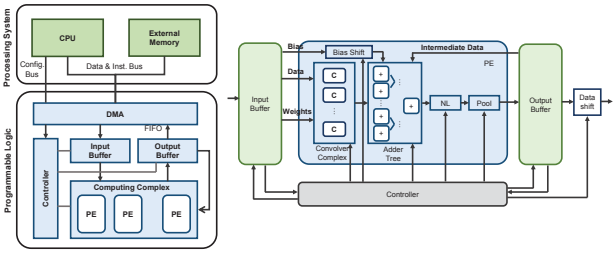

Fig. 1. The FPGA-based CNN accelerator: (a) the overall architecture; (b) the processing element.

A full CNN model consists of both convolutional layers and FC layers. Convolutional layers are generally computation-centric: they contain relatively much fewer weights but need a great deal of operations. FC layers are memory-centric: they usually contain hundreds of million weights, and each weight is used for only one time. Consequently, for FC layers, much time is spent on loading weights from the external memory. Therefore, it is necessary to go deeper with the FPGA platform on alleviating the memory and bandwidth problem.

B. Accelerating Large-Scale CNN on FPGA [9]

To accelerate large-scale CNNs, we use CPU+FPGA heterogeneous architecture to implement a CNN accelerator, as Figure 1(a) shows. The whole system can be divided into two parts: the Programmable Logic (PL) and the Processing System (PS).

PL is the FPGA chip, on which we place the *Computing Complex*, *On-chip Buffers*, *Controller*, and Direct Memory Accesses (*DMAs*). The Computing Complex consists of Processing Elements (PEs) which take charge of the majority of computation tasks in CNN, including convolution kernels, pooling, and FC. On-chip buffers, including input buffer and output buffer, prepare data to be consumed by PEs and store the generated results. Controller fetches instructions from the external memory and decodes them to orchestrate all the modules except DMAs on the PL. DMAs are working for transferring data and instructions between the external memory on the PS side and On-chip Buffers on the PL side.

PS consists of the external memory and a CPU. All the CNN model parameters, data, and instructions are stored in the external memory. Processors run bare-metal programs and help to orchestrate the whole inference phase by configuring the DMAs. Some non-linear functions such as the Softmax function are realized on CPU to ensure the flexibility.

The complete inference process of an image with the CNN accelerator consists of three steps that are executed in sequence: *data preparation*, *data processing*, and *result output*. In data preparation phase, image data, model data, and control data are stored in the external memory. In the data processing phase, CPU host starts to configure DMAs and the configured DMA loads data and instructions to the controller, triggers a computation process on PL. In the result

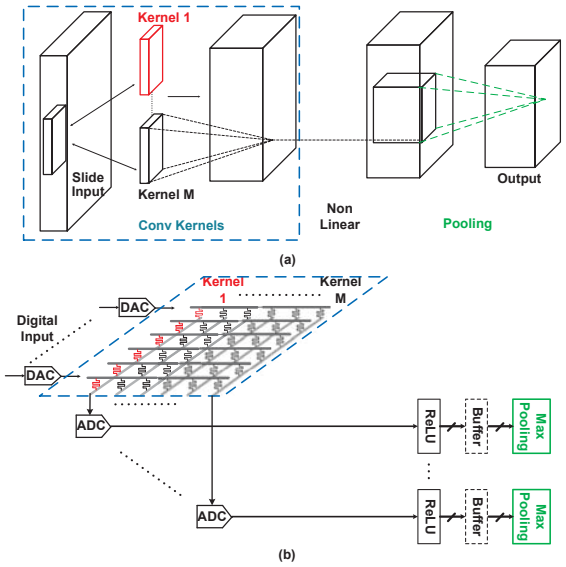

Fig. 2. (a). Functions in a convolutional layer. (b). Structure of the RRAM-based convolutional layer. Each RRAM row processes a specific convolution kernel.

output phase, once receiving the interrupt, the CPU host applies Softmax function to the results from PEs, and outputs the final results to UART port.

In addition, considering that the bandwidth problem in computing FC layers is mainly restricted by the bandwidth, we do not design specific hardware to accelerate FC layers, but reuse the Convolver Complex in one of the PEs to do the computation for FC layers. To fully utilize the bandwidth of the external memory with the current PL structure is the key point.

V. RRAM Design

A. RRAM-based Convolution Kernels

An RRAM device is a passive two-port element with multiple variable resistance states, and multiple RRAM devices can be used to build the crossbar structure as shown in Fig. 2(b). If the weights are stored by the conductivity of the RRAM devices and the data are reflected by the input voltage signals, the RRAM crossbar is able to perform analog convolution kernels. The relationship between the input and output signals can be expressed as in Eq. 5 [7]:

$$i_{out,k} = \sum_{j=1}^{N} g_{k,j} \cdot v_{in,j} \tag{5}$$

where $\vec{v}_{in}$ is the input voltage (denoted by $j = 1, 2, ..., N$), $\vec{i}_{out}$ is the output current denoted by $(k = 1, 2, ..., M)$, and $g_{k,j}$ is conductivity of the RRAM device representing the matrix data. Taking advantage of the non-volatile characteristics and the above natural "multiplication and merging" function of the crossbar structure, the RRAM crossbar can implement the convolution kernels in analog mode with high speed, small area, and low power.

For example, for the first convolutional layer of VGG-16 [3], there are 64 kernels in $3 \times 3 \times 3$ size. In RRAM-based implementation, we can use two 27×64 RRAM crossbars to store all the 64 kernels, where each RRAM row processes a specific convolution kernel and the positive weights and negative weights are stored separately. In this way, each convolution operation can be implemented in two RRAM crossbar rows, which can reduce the power and area consumption taking advantage of the strength of RRAM-based matrix-vector multiplication, as shown in Fig. 2(b). When the channel number of input feature map is large, it is better to separate the convolution kernel into multiple crossbars to make the network scale meet the requirement of the physical limitation of RRAM crossbar. After calculating the inner product of each channel, these intermediate results are added together to get final result of the convolution kernel.

The computation of FC layers is the same as traditional ANN, while the implementation of RRAM-based ANN has been already discussed by previous work [7], [11]. Moreover, in the five generally used CNN models (AlexNet [12], ZF [13], VGG-11, VGG-16, and VGG-19 [3]), the FC layers only take less than 2% of the whole computation in CNN. Therefore in this paper, we just explore the implementation of convolutional layer part on RRAM-based platform, which has not been well researched before.

B. Peripheral Functions in RRAM-based Convolutional Layer

To implement the whole convolutional layer, it is necessary to make a discussion on how to implement the ReLU function, max pooling function and other interface circuits after convolution kernels. However, since RRAM is used as an analog computing device, there are some challenges for implementing CNN on RRAM device: Reliability and Scalability.

Reliability: The ReLU function selects the larger one from the input signal and the reference signal (usually set as 0) while the max pooling function selects the largest one among the input nearby positions of the feature map. Both ReLU and max pooling functions can be regarded as obtaining the maximum value among multiple input signals. Since each comparator can only compare two inputs, the maximum value for multiple inputs needs to be selected by cascaded two-input "comparison and selection" functions. In each "comparison and selection" function, the two inputs are compared to generate a select signal showing the comparison result. Then the select signal drives selectors like transmission gates to output the analog maximum value. However, the above structure means the analog signals need to drive both the comparator and transmission gate, which can introduce noise caused by analog circuit variation and voltage amplitude loss, and the amplitude loss will further increases dramatically when computing the maximum value for the whole 4 signals in a typical 2×2 max pooling function. This problem causes critical impact because the output signal contains multi-bit analog levels in RRAM-based convolution.

Scalability: The analog intermediate results are difficult to be directly stored. When the depth of CNN and the size of pictures increases, there are huge amounts of intermediate results need to be buffered because the amount of channels increases as Fig. 2(a) shows. Without any buffer, for a 224×244 input picture with 3×3 bi-polar kernel in VGG-19 [3], we need to use at least 98568 (=$222 \times 222 \times 2$) RRAM crossbars only to process the first convolutional layer without any buffer and reuse, and all the 10^{11} RRAM cells of all layers need to work simultaneously until the final results are obtained. Therefore, directly implementing all the functions in analog circuits without buffers wastes lots of area and energy, which is not practical for large and deep CNNs considering both the data scale and the driving ability of the crossbar input.

The above shortages of analog signal processing lead to an approach that uses digital circuits to implement the functions between RRAM convolution crossbars, because digital data are insensitive to amplitude loss and are easy to be stored in latches or registers. As Fig. 2(b) shows, analog signals coming from RRAM crossbar are converted to digital signals through analog-to-digital converters

(ADCs), and then we can use digital circuits to implement the ReLU function and max pooling function. Also, a buffer can be introduced after the ReLU module to store the temporary results before max pooling. Specifically, considering that the weights of kernels in each layer are the same for different feature maps and different pictures, so we can reuse the crossbar in the same layer for multiple picture convolution operations in different feature maps. Therefore, the digital buffer provide a strong scalability for large images compared with traditional analog design.

VI. Case Study on VGG-16

A. Experimental Setup

The generally used VGG-16 model [3] for ImageNet dataset is demonstrated as a case study to evaluate our designs. The FPGA-based design uses Xilinx Znyq ZC706 board, which consists of the Xilinx Kinex-7 FPGA, dual ARM Cortex-A9 Processor whose frequency is 800 MHz, and 1 GB DDR3 memory offering 4.2GB/s bandwidth. For RRAM-based design, the area and power data of analog peripheral circuits and RRAM devices and are taken from [14]–[16]. We also show the performance of VGG-16 on Nvidia K40 GPU platform (2880 CUDA cores with 12GB GDDR5 384-bit memory) for comparison. The energy cost of obtaining picture data from memory is taken from [5]. For the accuracy and robustness emulation, an RRAM device model packed in Verilog-A [17] is used to build up the SPICE-level crossbar array. In addition, for FPGA, GPU, and ASIC design, the energy efficiency is calculated by the quotient of the platform speed (giga operations per second, GOP/s) and the power consumption (W), while for RRAM it is calculated by the quotient of problem complexity (GOP) and the whole energy cost (J) to process a specific picture.

B. Experimental Results

In Table I, we compare our designs with GPU implementation and previous ASIC work [4]. Taking advantage of the quantization and structure design, both FPGA design and RRAM design consume less energy than GPU. The 16-bit data and weight quantization for FPGA only reduces only 0.1% accuracy compared with 32-bit ideal VGG-16 model, while the 8-bit precision for RRAM reduces less than 2% accuracy. In addition, the ASIC design [4] can only process small CNN models for the limited on-chip memory and communication bandwidth. Specifically, the 16-bit weights need total 264MB memory and the intermediate data need 63.4MB memory, which is more than 1000 times larger than the SRAM size in [4]. If we take the difference of network scale into account, the energy efficiency of using thousands of chips in [4] is much lower and our methods. Also, the accuracy of [18] is not provided because this platform cannot directly support VGG-16 model. More detailed results about FPGA-based design are available in [9].

It should be noted that the RRAM-based design seems not such energy efficient in the results. The reason is that compared with the high efficiency of RRAM crossbar, the interfaces such as ADCs and digital-to-analog converters (DACs) contribute to large portion (about 95.2%) of the energy consumption of the whole system. The overhead introduced by ADCs/DACs even increases when implementing large and deep CNNs because the convolution kernels need to be performed in multiple RRAM crossbars.

TABLE I
Comparison of our FPGA Design, RRAM Design, GPU Implementation, and Previous ASIC Design [4].

	FPGA	RRAM	ASIC [4]	GPU
Problem Complexity (GOP)	30.76	30.76	0.0013	30.76
Weight	264MB	132MB	118KB	528MB
Data	63MB	32MB	45KB	127MB
Precision	16-bit fixed	8-bit fixed	16-bit fixed	32-bit float
Top-1 Accuracy	68.02%	66.58%	-	68.10%
Top-5 Accuracy	87.94%	87.38%	-	88.00%
Energy Efficiency (GOP/J)	14.22	462.67	606.25	7.14

VII. Conclusions

This paper provides a detailed discussion about the low-power CNN implementation on a chip for embedded devices. We point out that the high precision of CNN weights limits the implementation both FPGA design and RRAM design. We use a dynamic quantization method to compress precision of model while maintaining the recognition accuracy of CNN. We further discuss the design of FPGA-based CNN and the RRAM-based CNN, and compare the implementation results of our two designs. The experimental results show that both our FPGA design and RRAM design obtain large energy efficiency improvement compared with GPU implementation. In the future, we will further reduce the precision of intermediate data to reduce the multiplication cost for FPGA design and the energy consumptions of ADCs/DACs for RRAM design. The RRAM-based design proposed in this paper will be integrated in our simulator for RRAM-based computation *MNSIM* [19].

References

[1] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge (ilsvrc)," 2014.

[2] C. Szegedy *et al.*, "Going deeper with convolutions," *arXiv preprint arXiv:1409.4842*, 2014.

[3] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[4] Z. Du *et al.*, "Shidiannao: Shifting vision processing closer to the sensor," in *ISCA*, 2015, pp. 92–104. [Online]. Available: http://doi.acm.org/10.1145/2749469.2750389

[5] S. Han *et al.*, "Learning both weights and connections for efficient neural networks," *arXiv preprint arXiv:1506.02626*, 2015.

[6] C. Xu *et al.*, "Design implications of memristor-based rram cross-point structures," in *DATE*, 2011, pp. 1–6.

[7] M. Hu *et al.*, "Hardware realization of bsb recall function using memristor crossbar arrays," in *DAC*. ACM, 2012, pp. 498–503.

[8] L. Gao *et al.*, "A high resolution nonvolatile analog memory ionic devices," in *NVMW*, 2013, pp. paper–57.

[9] J. Qiu *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," to appear in FPGA 2016.

[10] G. E. Dahl *et al.*, "Improving deep neural networks for lvcsr using rectified linear units and dropout," in *ICASSP*, 2013, pp. 8609–8613.

[11] B. Li *et al.*, "Memristor-based approximated computation," in *ISLPED*. IEEE Press, 2013, pp. 242–247.

[12] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[13] M. D. Zeiler *et al.*, "Visualizing and understanding convolutional networks," in *ECCV*. Springer, 2014, pp. 818–833.

[14] R. St Amant *et al.*, "General-purpose code acceleration with limited-precision analog computation," in *ISCA*, 2014, pp. 505–516.

[15] W.-H. Tseng *et al.*, "A 960ms/s dac with 80db sfdr in 20nm cmos for multi-mode baseband wireless transmitter," in *VLSI Circuits Digest of Technical Papers*, 2014, pp. 1–2.

[16] J. Proesel *et al.*, "An 8-bit 1.5 gs/s flash adc using post-manufacturing statistical selection," in *CICC*, 2010, pp. 1–4.

[17] S. Yu *et al.*, "A low energy oxide-based electronic synaptic device for neuromorphic visual systems with tolerance to device variation," *Advanced Materials*, vol. 25, no. 12, pp. 1774–1779, 2013.

[18] V. Gokhale *et al.*, "A 240 g-ops/s mobile coprocessor for deep neural networks," in *CVPRW*, 2014, pp. 696–701.

[19] L. Xia *et al.*, "Mnsim: Simulation platform for memristor-based neuromorphic computing system," to appear in DATE 2016.