IMCE: Energy-Efficient Bit-Wise In-Memory Convolution Engine for Deep Neural Network

Shaahin Angizi, Zhezhi He, Farhana Parveen, and Deliang Fan
Department of Electrical and Computer Engineering
University of Central Florida, Orlando, FL 32816
Email:angizi@knights.ucf.edu, dfan@ucf.edu

Abstract— In this paper, we pave a novel way towards the concept of bit-wise In-Memory Convolution Engine (IMCE) that could implement the dominant convolution computation of Deep Convolutional Neural Networks (CNN) within memory. IMCE employs parallel computational memory sub-array as a fundamental unit based on our proposed Spin Orbit Torque Magnetic Random Access Memory (SOT-MRAM) design. Then, we propose an accelerator system architecture based on IMCE to efficiently process low bit-width CNNs. This architecture can be leveraged to greatly reduce energy consumption dealing with convolutional layers and also accelerate CNN inference. The device to architecture co-simulation results show that the proposed system architecture can process low bit-width AlexNet on ImageNet data-set favorably with 785.25μJ/img, which consumes $\sim$3$\times$ less energy than that of recent RRAM based counterpart. Besides, the chip area is $\sim$4$\times$ smaller.

I. Introduction

Deep Convolutional Neural Network (CNN) has achieved world-wide attention due to outstanding performance in image recognition over large scale data-set such as ImageNet [1]. For instance, ResNet shows a prominent recognition accuracy of 96.43%, which is higher than human beings (94.9%). Following the trend, when going deeper in CNNs (e.g. ResNet employs 18-1001 layers), memory/computational resources and their communication have faced inevitable limitations. This can be interpreted as "CNN power and memory wall" [2], leading to the development of different approaches to improve CNN efficiency at either algorithm or hardware level.

Estimation of CNN using shallower models, quantizing parameters [3,4], compressing pre-trained networks, and network binarization [5–7] are the most widely explored algorithmic approaches. Recent research efforts have significantly reduced both model size and computing complexity by using low bit-width weights, activations, and gradients [3, 4]. For example, Zhou et al. [3] have shown that low bit-width convolution kernels achieved from their quantization method can accelerate both training and inference with almost comparable prediction accuracy as 32-bit counterparts on ImageNet data-set.

In hardware design domain, the isolated memory and computing units (GPU or CPU) interconnected via buses has faced serious challenges, such as long memory access latency, significant congestion at I/Os, limited memory bandwidth, and huge leakage power consumption for the neural network acceleration [8, 9]. To address these concerns, in-memory processing platforms built on non-volatile devices can be an alternative solution to integrate memory and logic, leading to an energy-efficient information processing platform [8,10]. Resistive Random Access Memory (RRAM) [8, 10], Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM) [11] and recent Spin Orbit Torque Magnetic Random Access Memory (SOT-MRAM) [12] are very promising candidates to pave a novel path to realize such area and energy-efficient systems supporting in-memory processing due to features like non-volatility, zero standby leakage, compatibility with CMOS fabrication process and excellent integration density.

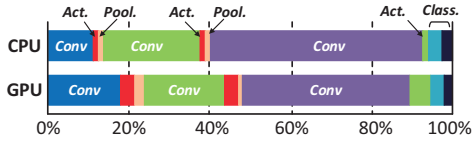


Fig. 1. Execution time of a sample CNN for scene labeling on CPU and GPU [1].

A CNN basically consists of multiple stacking layers, namely convolution, activation, and pooling. As depicted in Fig. 1 [1], convolutional layer always takes most fraction of execute time and computational sources in both GPU and CPU implementations. This motivates us to propose the first optimized bit-wise In-Memory Convolution Engine (IMCE) based on SOT-MRAM implementing an energy and area-efficient convolutional accelerator for low bit-width CNN.

The main contributions of this work are summarized as follows: (1) We first develop a computational sub-array architecture based on SOT-MRAM, which could be used as both non-volatile memory and reconfigurable in-memory logic; (2) We propose a CNN accelerator system architecture capable of implementing low bit-width convolution and pooling operations. The design of bit-wise IMCE along with other components are developed within the proposed system architecture; (3) We present detailed hardware mapping and task distribution of each computational layer of low bit-width CNNs into the proposed system; (4) We perform extensive experiments on our proposed CNN accelerator, such as inference accuracy, memory storage, area, and energy consumption, to show the favorable performance.

II. Preliminaries on CNN

In this section, we briefly review the terminology of CNN and introduce low bit-width CNN and binary CNN (BCNN). CNN is a machine learning classifier which takes an image as

978-1-5090-0602-1/18/$31.00 ©2018 IEEE

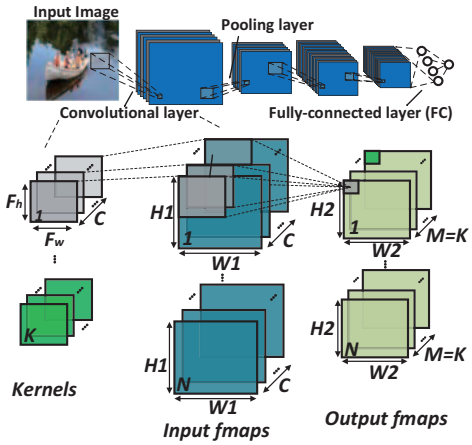
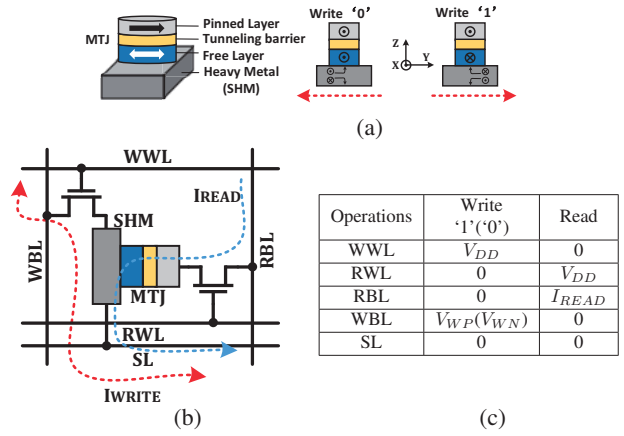


Fig. 2. Visualization of Inference (a.k.a forward propagation) in CNN.



Operations	Write '1'('0')	Read
WWL	V_{DD}	0
RWL	0	V_{DD}
RBL	0	I_{READ}
WBL	$V_{WP}(V_{WN})$	0
SL	0	0

Fig. 3. (a) SOT-MRAM device structure and Spin Hall Effect, (b) Schematic and (c) biasing conditions of SOT-MRAM bit-cell.

input and then computes the probabilities that image feature belong to a sort of output classes. Typically, a CNN consists of several convolutional layers and pooling layers followed by fully-connected layers (FC) as depicted in Fig. 2. Note that, it has been proven that fully-connected layers could be equivalently implemented by convolutions [3, 5]. Fig. 2 also shows visualization of convolutional layer of CNN where each layer receives a set of features organized in multi-channel as input (**Input fmaps**). It applies **kernels** (filters) by performing high-dimensional convolutions and then produces the features (**Output fmaps**) for the next layer [13]. The dimensions of both fmaps (input/output) and kernels are 4-D (multiple 3-D structures) and a batch of input fmaps is typically processed by multiple 3-D kernels. After convolution, a non-linear activation function, such as ReLU, will be applied to the results. Considering the shape parameters listed in Table I, the computation of one convolutional layer can be defined as follow:

$$O[n][k][x][y]= ReLU(B[k]+\sum_{i=0}^{F_h-1}\sum_{j=0}^{F_w-1}\sum_{z=0}^{C-1} I[n][z][U_x+i][U_y+j] W[k][z][i][j]), 0\le n<N, 0\le k<K, 0\le x<W2, 0\le y<H2; \quad (1)$$

where O, B, I, and W are the matrices representing output fmaps, Bias, input fmaps, and kernels, respectively. W2/H2 dimensions can be achieved as $W2 = (W1 - F_w + 2P)/S + 1$ and $H2 = (H1 - F_h + 2P)/S + 1$.

TABLE I
SHAPE PARAMETERS OF A CONVOLUTIONAL LAYER

Shape Parameter	Description
input fmaps dimension	$W1 \times H1 \times C$
3-D fmaps batch size (input/output)	N
no. of 3-D kernels	K
spatial extent of kernels	$F_w \times F_h \times C$
stride	S
no. of zero padding	P
output fmaps dimension	$W2 \times H2 \times M$

CNN functions in two different modes: (1) training mode in which the configuration values of layers are calculated by training the network on pre-classified training images, and (2) inference mode where new test images are examined. In both modes, according to Eq-(1), multiply-accumulate (MAC) is the key and most computationally expensive arithmetic operation [2]. To eliminate the need for massive MAC operations and memory usage, researchers have come up with various quantized/binary CNNs by forcing the inputs/weights/gradients to be quantized/binary specifically in forward propagation. DoReFa-NET shows acceptable accuracy over SVHN and ImageNet data-sets under different low bit-width configuration after applying its quantization method [3]. In an extreme quantization, BinaryConnect [6] trains deep neural networks with binary weights and shows near state-of-the-art results on MNIST and CIFAR-10 data-sets. BinaryNet [14] proposes an extension to BinaryConnect by binarizing both wights and activations. XNOR-NET [5] offers simple and accurate BCNNs and achieves almost similar results with full-precision AlexNet on ImageNet. Performing bit-wise convolution between the binary inputs and weights in forward path is shown in [5, 6] thanks to the following formula which computes the dot-product of two bit vector I and W using *bitcount* and *and* operations.

$$I * W = bitcount(and(I, W))^1 \quad (2)$$

III. In-Memory Processing Platform

A. SOT-MRAM

Fig. 3a shows a Spin-Orbit Torque Magnetic Random Access Memory (SOT-MRAM) device structure with the composite structure of spin Hall metal (SHM) and Magnetic Tunnel Junction (MTJ). Here the flow of charge current ($\pm$y) through the SHM (Tungsten, $\beta - W$ [15]) will cause accumulation of opposite directed spin on both surfaces of SHM due to spin Hall effect [16]. Thus, a spin current flowing in $\pm$z is generated and further produces spin-orbit torque (SOT) on the adjacent free magnetic layer, causing switch of magnetization. The bit-cell structure of 2T1R SOT-MRAM and its biasing conditions are shown in Fig. 3b and 3c, respectively.

In this work, the magnetization dynamics of the free ferromagnetic (FM) layer is modeled by LLG equation with STT term and SHE term [17]. Note that the ferromagnets in MTJ have In-plane Magnetic Anisotropy (IMA) in x-axis [16]. With the given thickness (1.2nm) of the tunneling layer (MgO), the Tunnel Magneto-Resistance (TMR) of the MTJ is $\sim$ 168.5%.

[1]When I and W are vectors $\in$\{-1,+1\}, this equation has a variant employing XNOR [3]:
$I * W = N - 2 \times bitcount(xnor(I, W))$

B. Computational Sub-array Architecture

The proposed SOT-MRAM sub-array architecture could work in dual mode that perform both memory read-write and AND/OR logic operations. Fig. 4a shows the architecture of 2×2 memory array. Each SOT-MRAM cell is associated with the Write Word Line (WWL), Read Word Line (RWL), Write Bit Line (WBL), Read Bit Line (RBL), and Source Line (SL) to perform typical memory operations. Moreover, in our design, any two cells in identical column (i.e. RBL) could be sensed simultaneously to implement an in-memory logic function. The peripheral decoders (active-high output) control the activation of current path through the array. Voltage drivers are used with the WBLs for providing the required write voltage. A voltage mode Sense Amplifier (SA) [16] is connected to the RBL for sensing the total resistance in the selected current path during Read or Computing mode.

Memory Write: To write a bit in any of the SOT-MRAM cells, for example in the cell of 1st row and 1st column, write current should be injected through the heavy metal substrate of SOT-MRAM. To activate this write current path, WWL1 should be activated by the Row Decoder and SL1 is grounded, while all the other word lines and source lines are kept deactivated. Now, in order to write '1' (/ '0'), the voltage driver (V1) connected with WBL1 is set to positive (/negative) write voltage. This allows sufficient charge current (120 μA) flows from V1 to ground (/ground to V1), leading to MTJ resistance in High-R_{AP} (/Low-R_P)

Memory Read: For typical memory read, a read current flows from the selected SOT-MRAM cell to ground, generating a sense voltage at the input of SA, which is compared with memory mode reference voltage ($V_{sense,P}<V_{ref}<V_{sense,AP}$). This reference voltage generation branch is selected by setting the Enable values $(EN_M, EN_{AND}, EN_{OR})$= (1,0,0). Now, if the path resistance is higher (/lower) than R_{ref}, i.e. R_{AP} (/R_P), then the output of the SA produces High (/Low) voltage indicating logic '1' (/ '0').

Computing Mode: In computing mode, every two bits stored in the identical column can be selected and sensed simultaneously as depicted in Fig. 4a. Note that, the row decoders are modified to support multi-line enable function, through combining two single-line enable decoder with their outputs connected to OR gates. Then, the equivalent resistance of such parallelly connected SOT-MRAMs and their cascaded access transistors are compared with a specific reference by SA. Through selecting different reference resistances $(EN_M, EN_{AND}, EN_{OR})$, the SA can perform basic in-memory Boolean functions (i.e. AND and OR). For AND operation, R_{ref} is set at the midpoint of $R_{AP}//R_P$ ('1','0') and $R_{AP}//R_{AP}$ ('1','1'). Thus only when both of the two selected SOT-MRAM bit-cells are in anti-parallel state (i.e. binary input: '1','1'), the output is high, whereas output is low. Similarly, for OR operation, R_{ref} is set at the midpoint of $R_P//R_P$ and $R_P//R_{AP}$. To validate the variation tolerance of sense circuit, we have performed Monte-Carlo simulation with 100000 trials. A $\sigma = 5\%$ variation is added on the Resistance-Area product (RA_P), and a $\sigma = 10\%$ process variation is added on the TMR. The simulation result of sense voltage (V_{sense}) distributions in Fig. 4b shows the sense margin of in-memory computing. It will be reduced by increasing the logic fan-in (i.e. number of parallel memory cells). Thus, to avoid read failure (overlapping of V_{sense} distribution), only two fan-in in-memory logic is used in this work. Note that parallel computing/read within sub-array is implemented by using one SA per bit-line with exact same mechanism.

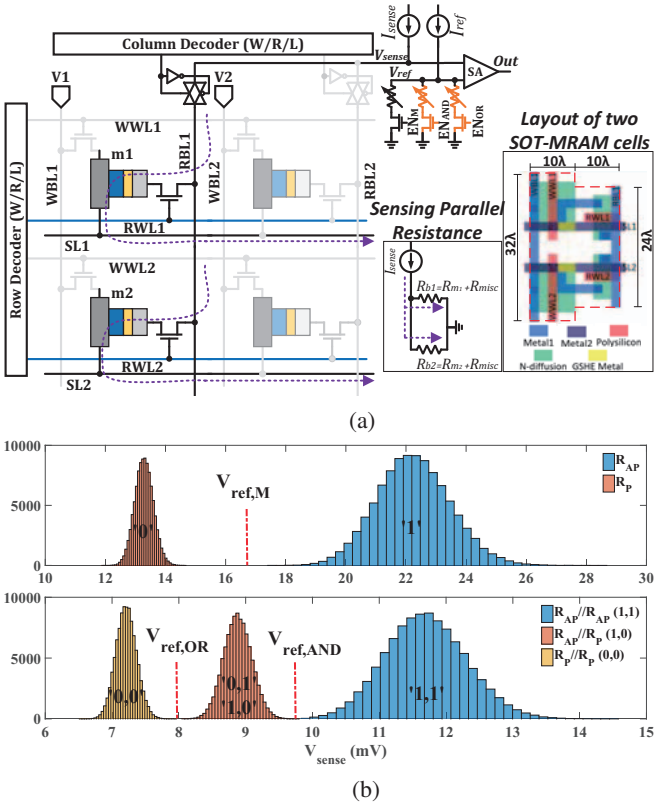


Fig. 4. (a) Proposed in-memory processing sub-array architecture based on SOT-MRAM. The layout of two adjacent SOT-MRAM cells is also indicated. (b) Monte Carlo simulation result of the sense voltage (V_{sense}) distribution.

IV. System Architecture of Accelerator

General overview of the proposed system architecture for performing low bit-width CNN is shown in Fig. 5a. This architecture mainly consists of Image Bank, Kernel Bank, bit-wise In-Memory Convolution Engine (IMCE), and Digital Processing Unit (DPU). As it can be seen in Fig. 1, convolutional layers contribute the largest fraction of computation time and complexity to CNNs. That is why in this work we mainly focus on convolutional layer. Assume Input fmaps (I) and Kernels (W) are initially stored in Image Banks and Kernel Banks of memory, respectively. As depicted in Fig. 5a, inputs need to be constantly quantized before mapping into computational sub-arrays. However, quantized shared kernels can be utilized for different inputs. This step is performed using DPU's Quantizer and then the results are mapped to IMCE's sub-arrays (Fig. 5b). For realization of bit-wise IMCE, the proposed computational sub-array in Section III is readily utilized such that ultra-efficient and parallel in-memory AND operations required for convolutions can be handled. The functionality of bit-wise IMCE and other components is elaborated in the following subsections.

A. Bit-Wise In-Memory Convolution Engine

The main idea behind employing bit-wise convolution is to exploit *logic AND*, *bitcount*, and *bitshift* as rapid and paral-

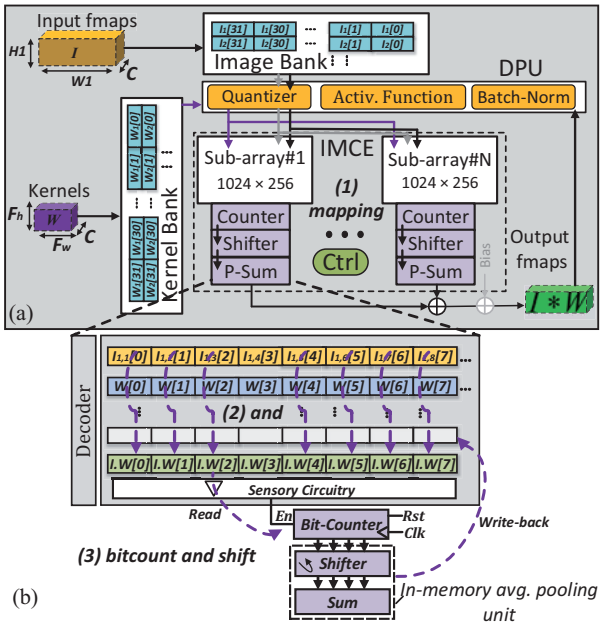


Fig. 5. (a) General overview of the proposed CNN accelerator with image bank, kernel bank, computational sub-arrays, and DPU, (b) Bit-wise IMCE's sub-array.

lelizable operations to accelerate the MACs in convolutional layers. As these operations are polynomial in the product of bit-width multiplicands, reducing the bit-width brings remarkable improvements in computation complexity dominating the run-time of Neural Networks specially in resource-constrained environments. The 1-bit convolution depicted in Eq-(2) is generalized to compute the dot-product and consequently convolution of k-bit fixed point integers [3]. Here we show an example to elaborate the mapping method and operations of IMCE.

Assume I is a sequence of M-bit input integers (3-bit, as shown in Fig. 6) located in input fmap covered by sliding kernel of W, such that $I_i \in I$ is an M-bit vector representing a fixed-point integer. Now, we index the bits of each I_i element from LSB to MSB with $m = [0, M-1]$, such that $m = 0$ and $m = M-1$ are corresponding to LSB and MSB, respectively. Accordingly, we represent a second sequence denoted as $C_m(I)$ including the combination of m^{th} bit of all I_i elements (shown by colored elliptic). For instance, $C_0(I)$ vector consists of LSBs of all I_i elements "0110".

Considering W as a sequence of N-bit weight integers (3-bit, herein) located in sliding kernel with index of $n = [0, N-1]$, the second sequence can be similarly generated like $C_n(W)$. Now, by considering the set of all m^{th} value sequences, the I can be represented like $I = \sum_{m=0}^{M-1} 2^m C_m(I)$. Likewise, W can be represented like $W = \sum_{n=0}^{N-1} 2^n C_n(W)$. In this way, the convolution between I and W can be defined as follow:

$$I * W = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} 2^{m+n} bitcount(and(C_n(W), C_m(I))) \quad (3)$$

To efficiently load the resultant data from Quantizer unit to IMCE, I and W should be reorganized. As shown in data organization and mapping step of Fig. 6, $C_2(W)$-$C_0(W)$ are consequently mapped to the designated sub-array. Accordingly, $C_2(I) - C_0(I)$ are mapped in the following memory rows in the same way. Now, IMCE can perform bit-wise parallel AND

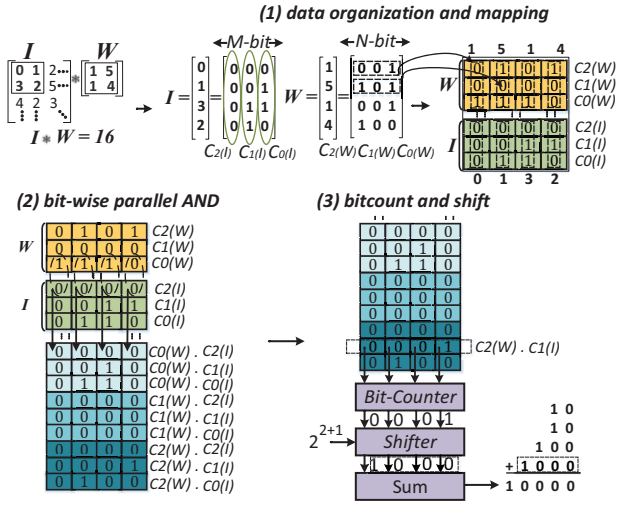


Fig. 6. Mapping method and operations of bit-wise IMCE.

operation of $C_n(W)$ and $C_m(I)$ as depicted in Fig. 6. Considering the proposed SOT-MRAM sub-array architecture in Fig. 4a, by activating two specific RWLs in each subarray, the shared WBLs can be sensed to produce the result of *AND* operation. This data organization ensures that at least $(F_w \times F_h \times C)$ binary operations can be concurrently processed in one cycle within one active sub-array.

The results of parallel AND operations stored within sub-array will be accordingly processed using Bit-Counter. Bit-Counter readily counts the number of "1"s within each resultant vector and passes it to the Shifter unit. As depicted in Fig. 6, "0001", as result of Bit-Counter is left-shifted by 3-bit ($\times 2^{2+1}$) to "1000". Eventually, Sum unit adds the Shifter unit's outputs to produce the output fmaps.

The output fmaps coming from convolutional layer can be later processed for down-sampling using average pooling. Average pooling operation is performed using Sum and Shifter units, respectively, by summing up the output fmap's tensors and dividing (shifting) into rectangular pooling region size.

B. DPU's Components

Quantizer: This unit quantizes a real number input $r_i \in [0, 1]$ to a k-bit number output $r_o \in [0, 1]$ using quantization function [3]:

$$r_o = \frac{1}{2^k - 1} round((2^k - 1) r_i) \quad (4)$$

Batch-Norm. Batch Normalization layer [7] alleviates the information loss during quantization by normalizing the input batch to have zero mean and unit variance. The transformation can be written as:

$$I_o(R) = \frac{I_i(R) - \mu}{\sqrt{\sigma^2 + \varepsilon}} \gamma + \beta \quad (5)$$

where $I_o(R)$ and $I_i(R)$ denote the corresponding output and input pixels, respectively. σ and μ represent statistics achieved during training mode, γ and β are trained parameters, and ε is to avert round-off problem. During inference mode, all the parameters in aforementioned equation are stored in SOT-MRAM sub-arrays, so DPU efficiently fetches each pixel of input fmap and writes back the corresponding normalized pixel.

Activ. Function: The proper selection of activation function has a profound impact on network prediction accuracy specially in lower bit-width CNN. This unit can be reconfigured to perform two distinct activation functions (i.e. $\frac{tanh(x)+1}{2}$ and $sign(x)$) to provide highest accuracy.

V. Experiments

In this section, we explore the mapping results and configuration space of various bit-width of input/weight in terms of accuracy, memory storage, energy, and area over different data-sets.

A. Accuracy

Bit-width configuration: Six bit-width configurations of W:I (32:32, 1:1, 1:2, 1:3, 1:4, and 2:2) are considered for the accuracy evaluation. The 8-bit gradient is applied to all configurations except 32:32 as the baseline with 32-bit gradient.

Data-set: The SVHN data-set [18], as a real-world image data-set consisting of photos of house numbers in Google Street View images, is selected for evaluation. There are 73257 digits for training, 26032 digits for testing, and 531131 additional digits as extra training data. The cropped format of colored images (32×32) centered around each single digit is selected. Accordingly, the images are re-sized to 40×40 and fed to the model.

Model: A CNN with 6 (bit-wise) convolutional layers, 2 (average) pooling layers and 2 FC layers that cost about 80 FLOPs for a 40×40 image is adopted. It is noteworthy that we avoids quantization in the first and last layers to avert further prediction accuracy degradation [3, 5]. FC layers are equivalently implemented by convolutions.

Training: For training, we use existing open source algorithm presented by DoReFa-Net [3] where all the operations can be accelerated significantly using bit-wise convolution of fixed-point integers. Furthermore, we adopt batch normalization and different dropout techniques to accelerate and avoid over-fitting. The model is trained on TensorFlow [19] with 100 epochs where the lowest test error of epoch is reported. Finally ADAM learning rules is used for training with 0.001 as learning rate in different configurations.

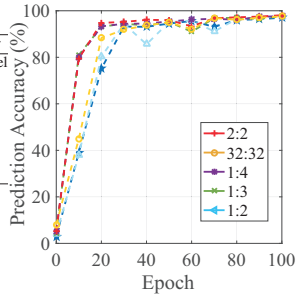
Results: Fig. 7a tabulates the test error results and relative complexity of mentioned model under various bit-width configurations. Complexity of inference and training are achieved using $W \times I$ and $W \times I + W \times G$, respectively. Generally, experiments replicate the conclusion drawn by [3] that weights, inputs and gradients are progressively more sensitive to bit-width changes. Fig. 7b depicts the prediction accuracy curve vs. number of epoch in different configurations.

B. Storage Requirements

Six bit-width configurations of W:I (32:32, 1:1, 1:2, 1:3, 1:4, and 2:2) are considered for the evaluation of memory storage of CNN model discussed earlier. The breakdown of memory storage is shown in Fig. 8a under different bit-width configurations. As shown, lower bit-width the CNN model is, less memory storage is required. To further explore the memory storage for larger data-sets, Fig. 8b depicts required memory storage for three bit-width configurations of W:I (64:64, 32:32, and 1:1) for AlexNet model running Image-Net data-set mapped to the proposed system architecture. It can be seen that 1:1 bit-width configuration requires 39.7MB memory which is ∼12× and ∼6× less compared to its double precision and single precision CNN implementations, respectively.

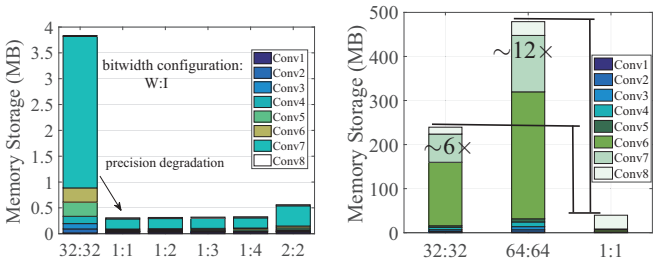
Config.		Complexity		Test Error
W	I	Inference	Training	CNN Model
32	32	-(†)	-	2.4%
1	1	1	9	3.1%
1	2	2	10	2.6%
1	3	3	11	2.4%
1	4	4	12	2.4%
2	2	4	20	1.8%

(†) The computation complexity of 32:32 is not shown, since it is not computationally efficient to perform bit-wise convolution of 32:32 configuration [3] and it is already reported in previous works.



(a) (b)

Fig. 7. (a) Test error of CNN model for processing SVHN in different bit-width configurations, (b) Evolution of prediction accuracy vs. epoch.

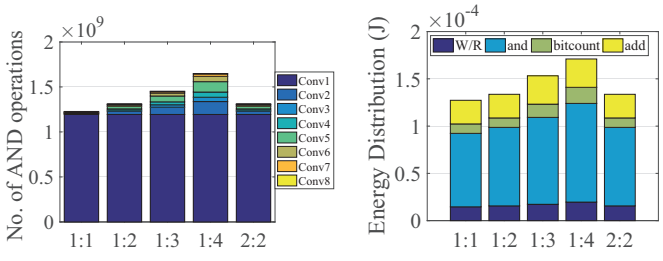


(a) (b)

Fig. 8. Memory storage (inputs/weights) required by (a) CNN model for processing SVHN, (b) AlexNet architecture for processing ImageNet.

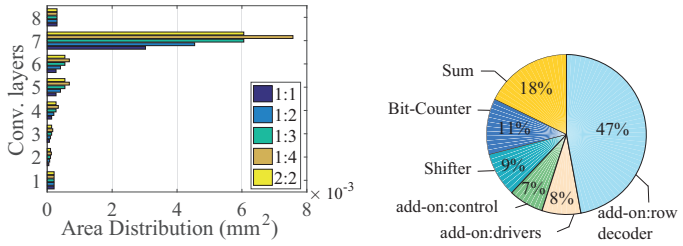
C. Energy Consumption Estimation

In this subsection, the energy consumption of discussed CNN model considering different bit-width configuration is estimated. To perform the experiment, the circuit level simulation is initially implemented in Cadence Spectre with NCSU 45nm CMOS PDK [20]. SOT-MRAM resistive model of Fig. 3a is used in the circuit simulation. The MTJ resistance (R_{MTJ}) is obtained from the NEGF approach [21], while the heavy metal resistance (R_{SHM}) is calculated based on the resistivity and device dimension. Accordingly, we massively modified the system level memory evaluation tool NVSim [22] to co-simulate with an in-house developed C++ code based on circuit level results. Clearly, AND operations performed by bit-wise IMCE dominate the energy consumption of CNN's models. Therefore, we first report number of requisite ANDs of CNN model for processing a single image of SVHN data-set. Fig. 9a shows the break-down of performed AND operations in convolutional layers using bit-wise IMCE. Correspondingly, we show the energy distribution of CNN divided into W/R, and, $bitcount$, and add operations under different bit-width configurations. Note that the energy consumption of Shifter and Bit-Counter are plotted together under bitcount in Fig. 9b.



(a) (b)

Fig. 9. (a) Break-down of no. AND operations in bit-wise IMCE for processing SVHN data-set, (b) Energy distribution under varying configurations.



(a) (b)

Fig. 10. (a) Area distribution of conv. layers mapped to IMCE for processing a single image of SVHN, (b) Breakdown of area overhead of IMCE.

D. Area Estimation

We have profiled the area distribution of different convolutional layers of CNN model for processing a single image of SVHN data-set in distinct configurations (1:1, 1:2, 1:3, 1:4, 2:2) in Fig. 10a. To further explore the area occupation of IMCE, Fig. 10b shows the break-down of area overhead resulted from add-on hardware to sub-arrays. Our experiments show that totally IMCE imposes 15.4% area overhead to original memory chip. It can be seen that modified row decoder and Sum unit contribute more than 60% of area overhead.

E. Hardware Mapping Comparison

In this subsection, we compare the hardware mapping results of CNN accelerators implemented by two promising resistive memories (i.e. RRAM [8] and SOT-MRAM herein) over three different data-sets in terms of energy and area under 45nm technology node. Table II shows that the proposed accelerator exploiting bit-wise IMCE can process BCNN over different data-sets very efficiently. For instance, it processes binary-weight AlexNet [5] for ImageNet favorably with 785.25μJ/img where $\sim$3$\times$ and 4$\times$ lower energy and area are achieved, respectively, compared to RRAM-based design. In addition to area/energy efficiency of SOT-MRAM compared to RRAM, such significant improvements mainly come from two sources: (1) RRAM design employs matrix splitting due to intrinsically limited bit levels of RRAM device so multiple sub-arrays should be occupied, (2) RRAM-based crossbar peripheral circuit's overhead such as buffers and DAC/ADC which contribute more than 85% of area and energy consumption [8, 10]. Note that the energy reported in Table II is the convolution computation energy of all layers as an accelerator.

TABLE II
PERFORMANCE ESTIMATION OF CNN AND BCNNS ACCELERATORS

Designs	ImageNet		SVHN		MNIST	
	Energy (μJ/img)	Area (mm^2)	Energy (μJ/img)	Area (mm^2)	Energy (μJ/img)	Area (mm^2)
CNN-RRAM [8]	5444.85	21.25	850.42	0.09	18.39	0.054
BCNN-RRAM [8]	2275.34	9.19	425.21	0.085	13.55	0.060
BCNN-SOT-MRAM	785.25	2.12	135.26	0.01	0.92	0.009

VI. CONCLUSION

In this paper, we pave a way towards novel concept of bit-wise In-Memory Convolution Engine (IMCE) that could implement the dominant convolution computations of either CNN or binary CNN within computational sub-arrays. The hardware mapping results show that the proposed accelerator based on IMCE can process the binary-weight AlexNet [5] on ImageNet dataset favorably with 785.25μJ/img where $\sim$ 3$\times$ and 4$\times$ lower energy and area are achieved, respectively, compared to RRAM-based counterpart.

ACKNOWLEDGEMENTS

This material is based upon work supported in part by the National Science Foundation under Grant No.1740126.

REFERENCES

[1] L. Cavigelli *et al.*, "Accelerating real-time embedded scene labeling with convolutional networks," in *DAC, 2015 52nd ACM/EDAC/IEEE*. IEEE, 2015, pp. 1–6.

[2] R. Andri *et al.*, "Yodann: An architecture for ultra-low power binary-weight cnn acceleration," *IEEE TCAD*, 2017.

[3] S. Zhou *et al.*, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[4] Courbariaux *et al.*, "Training deep neural networks with low precision multiplications," *arXiv preprint arXiv:1412.7024*, 2014.

[5] M. Rastegari *et al.*, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.

[6] M. Courbariaux *et al.*, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems*, 2015, pp. 3123–3131.

[7] R. Zhao *et al.*, "Accelerating binarized convolutional neural networks with software-programmable fpgas," in *Proceedings of the 2017 ACM/SIGDA FPGA*. ACM, 2017, pp. 15–24.

[8] T. Tang *et al.*, "Binary convolutional neural network on rram," in *22nd ASP-DAC*. IEEE, 2017, pp. 782–787.

[9] S. Li *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *2016 53nd DAC*. IEEE, 2016.

[10] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ISCA*, vol. 43, 2016.

[11] Y. Kim *et al.*, "Write-optimized reliable design of stt mram," in *Proceedings of the 2012 ACM/IEEE ISLPED*. ACM, 2012.

[12] G. Prenat *et al.*, "Beyond stt-mram, spin orbit torque ram sot-mram for high speed and high reliability applications," in *Spintronics-based Computing*. Springer, 2015.

[13] Y.-H. Chen *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, 2017.

[14] C. Matthieu et al., "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or-1," *arXiv arXiv:1602.02830*, 2016.

[15] C.-F. Pai *et al.*, "Spin transfer torque devices utilizing the giant spin hall effect of tungsten," *Applied Physics Letters*, 2012.

[16] X. Fong *et al.*, "Spin-transfer torque devices for logic and memory: Prospects and perspectives," *IEEE TCAD*, vol. 35, 2016.

[17] Z. He *et al.*, "A low power current-mode flash adc with spin hall effect based multi-threshold comparator," in *ISLPED*. ACM, 2016.

[18] Y. Netzer *et al.*, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on deep learning and unsupervised feature learning*, vol. 2011, no. 2, 2011, p. 5.

[19] M. Abadi *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[20] (2011) Ncsu eda freepdk45. [Online]. Available: http://www.eda.ncsu.edu/wiki/FreePDK45:Contents

[21] G. Panagopoulos *et al.*, "A framework for simulating hybrid mtj/cmos circuits: Atoms to system approach," in *DATE*, 2012.

[22] X. Dong *et al.*, "Nvsim: A circuit-level performance, energy, and area model for emerging non-volatile memory," in *Emerging Memory Technologies*. Springer, 2014, pp. 15–50.