CMP-PIM: An Energy-Efficient Comparator-based Processing-In-Memory Neural Network Accelerator

Shaahin Angizi†, Zhezhi He†, Adnan Siraj Rakin and Deliang Fan
† These authors contributed equally.
Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL 32816
{angizi,elliot.he,adnanrakin}@knights.ucf.edu,dfan@ucf.edu

ABSTRACT

In this paper, an energy-efficient and high-speed comparator-based processing-in-memory accelerator (CMP-PIM) is proposed to efficiently execute a novel hardware-oriented comparator-based deep neural network called CMPNET. Inspired by local binary pattern feature extraction method combined with depthwise separable convolution, we first modify the existing Convolutional Neural Network (CNN) algorithm by replacing the computationally-intensive multiplications in convolution layers with more efficient and less complex comparison and addition. Then, we propose a CMP-PIM that employs parallel computational memory sub-array as a fundamental processing unit based on SOT-MRAM. We compare CMP-PIM accelerator performance on different data-sets with recent CNN accelerator designs. With the close inference accuracy on SVHN data-set, CMP-PIM can get ~ 94× and 3× better energy efficiency compared to CNN and Local Binary CNN (LBCNN), respectively. Besides, it achieves 4.3× speed-up compared to CNN-baseline with identical network configuration.

1 INTRODUCTION

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 even higher than human beings (94.9%). Following the trend, when going deeper in CNNs (e.g. ResNet employs 18-1001 layers), memory/computation resources and their communication have faced inevitable limitations. This has been interpreted as "CNN power and memory wall" [2], leading to development of different approaches to improve CNN efficiency at either algorithm or hardware level.

The most widely explored algorithmic approaches to address such issues of CNN involves using shallower models, quantizing parameters [3], compressing pre-trained networks, and network binarization [4]. It has been proven that convolution layers consume up to 90% of computation time of CNN with main purpose of feature extraction, which brings up a question to the community: Does there exist any other methods that can provide feature extraction with much less computational complexity while having similar CNN output accuracy? Following this trend, recently, some powerful texture descriptors like Gabor filter [5] and Local Binary Pattern (LBP) [6] have been employed to perform feature extraction in CNN, resulting in a similar output inference accuracy [7, 8].

In hardware domain, the separated 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 [9, 10]. To address these concerns, processing-in-memory (PIM) platforms built on non-volatile devices can be an alternative solution to integrate memory and logic, leading to an energy-efficient information processing platform [9, 11]. Resistive Random Access Memory (RRAM) [9, 11], Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM) [12] and recent Spin Orbit Torque Magnetic Random Access Memory (SOT-MRAM) [13] are very promising candidates to pave a novel path to realize such area and energy-efficient system supporting in-memory processing due to features like non-volatility, zero standby leakage, compatibility with CMOS fabrication process and excellent integration density.

In this work, inspired by LBP feature extraction method combined with depthwise separable convolution, we first modify the existing CNN algorithm by replacing the computationally-intensive multiplications in convolution layers with more efficient and less complex comparison and addition (CMPNET) in Section 2. Then, we present CMP-PIM as a comparator-based processing-in-memory accelerator based on SOT-MRAM array to accelerate CMPNET in Sections 3 and 4. Such processing-in-memory structure shows significant improvements in terms of energy consumption and delay. Accordingly, we evaluate CMP-PIM performance from both algorithm and hardware implementation perspectives in Section 5.

2 CMPNET

2.1 Local Binary Pattern

LBP, as a computationally-efficient feature descriptor with remarkable performance, has been extensively employed in various image processing applications [6]. It scans through the entire image similar as a convolutional layer, re-coding the image w.r.t the local information correlations. As shown in Fig. 1a, the LBP descriptor is formed by sequentially comparing the intensity of surrounding pixels with central pixel (referred to as Pivot) in the selected image patch. Neighbors with higher (/lower) intensity are assigned with binary value of '1'(/'0') and finally the bit stream is sequentially read and mapped to a decimal number as the feature value assigned to the central pixel. This LBP encoding operation of central pixel $C(x_c, y_c)$ and its reformulated expression can be mathematically described as [6, 7]:

$$LBP(C) = \sum_{n=0}^{d-2} cmp(i_n, i_c) \cdot 2^n = \sum_{n=1}^{d-2} \sigma((+1 \cdot i_n) + (-1 \cdot i_c)) \cdot 2^n \quad (1)$$

DAC '18, San Francisco, CA, USA

DOI: 10.1145/3195970.3196009

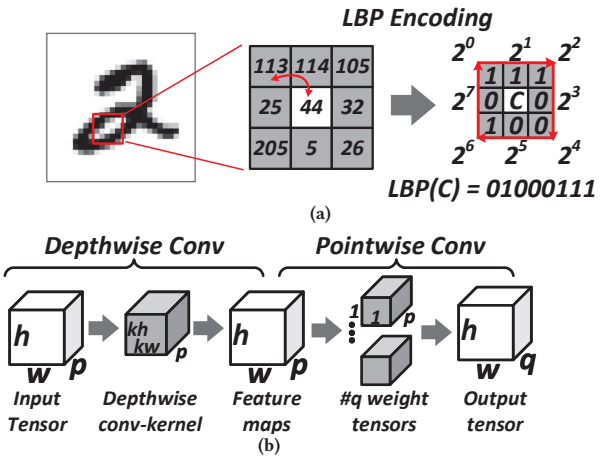


Figure 1: (a) Standard LBP encoding with 3 ×3 descriptor size. (b) Data flow diagram of depthwise separable convolution.

where d is the dimension of the LBP, i_n and i_c represent the intensity of n^{th} neighboring- and central-pixel, respectively. $cmp(i_n, i_p) = 1$ when $i_n \geq i_c$, otherwise outputs 0. $\sigma()$ is step function, where $\sigma(*) = 1$ if $* \geq 0$, otherwise $\sigma(*) = 0$. This equation clearly shows that the comparison of LBP can be equivalently performed by a subtraction and step function. Furthermore, the subtraction can be substituted by convolution with a *fixed ternary kernel*, which is defined in this work as a convolution kernel with only one '+1' and one '-1', while the rest weight elements are zeros. An example of convolution with the fixed ternary kernel is described in the following equation:

$$\overbrace{\begin{bmatrix} i_{n0} & i_{n1} & i_{n2} \\ i_{n3} & i_c & i_{n4} \\ i_{n5} & i_{n6} & i_{n7} \end{bmatrix}}^{\text{image patch}} \cdot \overbrace{\begin{bmatrix} 0 & +1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}}^{\text{fixed ternary kernel}} = i_{n1} - i_c \tag{2}$$

We define the sparsity of the convolution kernel as (#non-zero weights/#total weights). In our case, the sparsity of such fixed ternary kernel is $2/(kh * kw)$, where kh and kw are the height and width of kernel, respectively.

2.2 Depthwise Separable Convolution

Recently, the depthwise separable convolution [14] has been used in many advanced neural networks such as MobileNet [15] and Xception [16] to replace the standard convolutional layer, targeting to reduce CNN computational cost. As a factorized form of normal convolution, the depthwise separable convolution consists of two parts: depth-wise convolution and 1x1 convolution (a.k.a. pointwise convolution), as depicted in Fig. 1b. It is intriguing to notice that, there exists many similarities between the structure of LBP and depthwise separable convolution. First, both the comparison in LBP and the depthwise convolution extract features based on the local information. Second, both the weighted summation in LBP and pointwise convolution create new representations through linearly combining the extracted features. Based on this observation, we propose to further integrate LBP feature extraction method into depthwise separable convolution, using the introduced fixed ternary kernel for depthwise convolution and inserting a binarized activation function between depthwise convolution and pointwise convolution. It could greatly reduces the computational cost and could be easily implemented in hardware as will be described later.

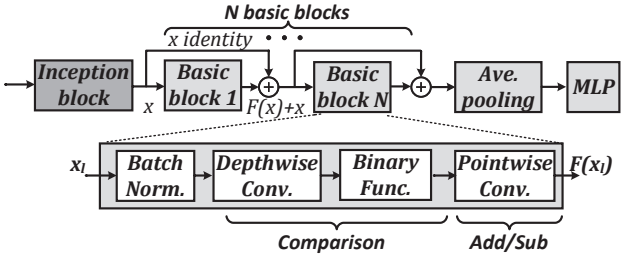


Figure 2: Block diagram of CMPNET and its basic block.

2.3 CMPNET: Comparator-based Network

As shown in Fig. 2, the proposed structure of CMPNET is built upon the Residual Network (ResNet) [17], which composes an initial *inception block* (3 × 3 spatial convolution [1], Batch-normalization and ReLU), *N basic blocks*, one *Average Pooling* layer and *Multi-Layer perceptron block (MLP)*. As the vital part of CMPNET, the basic block is constructed by a series of operations, including batch normalization, depthwise convolution with fixed sparse kernel, binarized activation function and pointwise convolution.

We have found that the incorporation of convolution with fixed ternary kernel and step function is mathematically equivalent to the comparison in Equation. 1,2. For the depthwise convolution in CMPNET, we use a randomly generated group of fixed ternary convolution kernels that will not be updated during training. Considering such fixed ternary kernel group is in the size of $kh \times kw \times p$, for each kernel-slice ($kh \times kw$), we randomly select two elements and set them as +1 and -1, respectively. When such kernels perform convolution over the chosen patch of the image, a subtraction is equivalently calculated between two selected pixels. Then, combining with the following binarized activation function, the computations of two layers (i.e. depthwise convolution and binarized activation function) can be converted into comparisons.

Unlike depthwise fixed ternary convolution kernels, the weights of pointwise convolution layer are fully learnable during training. More importantly, due to the inserted binarized activation function, the input tensor of pointwise convolution layer only contains +1 and -1. Thus, the pointwise convolution can be implemented with only addition/subtraction operations. Therefore, in CMPNET, for both comparison and pointwise convolution operations, all the Multiplication and Accumulation (MAC) operations within the convolution layers are converted into computationally-efficient comparison and addition/subtraction. In summary, the response of basic block in CMPNET can be described as:

$$\begin{aligned} x_{l+1}^t &= \sum_{s=1}^{p} \sigma'\left(K_l^s * BN(x_l^s)\right) \cdot \alpha_{l,s}^t \\ &= \sum_{s=1}^{p} cmp'\left(BN(x_{l,a}^s), BN(x_{l,b}^s)\right) \cdot \alpha_{l,i}^t \end{aligned} \tag{3}$$

where $s \in [p]$ and $t \in [q]$ denote the input channel and output channel, respectively. K_i is generated sparse binary convolution kernel. l is the index of basic block in CMPNET, while p is the

[1]To avoid ambiguity, all the unspecified convolution layer are normal spatial convolution layer hereinafter. All the convolution kernel in this work has no bias term. All the convolution operations will not reduce the tensor size through zero padding.

number of input channels of l_{th} basic block. $\alpha_{l,i}$ is the real value weight learned from training. $BN()$ is batch-normalization. $\sigma'()$ is the modified step function, where $\sigma'(*) = +1$ if $* \geq 0$, otherwise $\sigma'(*) = -1$. $cmp'(m,n) = +1$ when $m \geq n$, otherwise outputs -1. a and b are the index the selected two pixels in one image channel.

We list the computational cost and memory cost of convolution layer in both traditional CNN and proposed CMPNET as in Table 1. It can be seen that CMPNET (with fixed ternary kernel and binarized activation function) significantly reduces the hardware cost in both computation and memory. Moreover, in order to successfully map such algorithm to hardware, we adopt the quantization method introduced in [3]. The detailed CMPNET algorithm accuracy in pattern recognition will be presented in Section 5.1.

Table 1: Hardware cost of CNN and CMPNET.

	Computational cost		Memory cost
	Mul–$O(N^2)$	Add/Sub/Cmp–$O(N)$	
CNN	$p \cdot q \cdot h \cdot w \cdot kh \cdot kw$	$p \cdot q \cdot h \cdot w \cdot kh \cdot kw$	$p \cdot q \cdot kh \cdot kw$
CMP-NET	-	$p \cdot h \cdot w \cdot 2 + p \cdot q \cdot h \cdot w$	$p \cdot kh \cdot kw + p \cdot q$
$\frac{CMPNET}{CNN}$	0	$\frac{2+q}{q \cdot kh \cdot kw} \approx \frac{1}{kh \cdot kw}$	$\frac{1}{q} + \frac{1}{kh \cdot kw}$

3 IN-MEMORY PROCESSING PLATFORM

Fig. 3a shows the presented PIM sub-array architecture based on Spin-Orbit Torque Magnetic Random Access Memory (SOT-MRAM). This architecture could work in dual mode that perform both memory read-write and AND/OR/XOR logic operations. SOT-MRAM device is a composite structure of Spin Hall Metal (SHM) and Magnetic Tunnel Junction (MTJ). The resistance of MTJ with parallel magnetization in both magnetic layers (data-'0') is lower than that of MTJ with anti-parallel magnetization (data-'1'). 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 the following operations:

1) Memory Write: To write a data bit in any of the SOT-MRAM cells (e.g. m2 in Fig. 3a), write current should be injected through the SHM (Tungsten, $\beta - W$ [18]) of SOT-MRAM. Therefore, WWL2 should be activated by the Row Decoder where SL2 is grounded. Now, in order to write '1'(/'0'), the voltage driver (V1) connected to WBL1 is set to positive (/negative) write voltage. This allows sufficient charge current flows from V1 to ground (/ground to V1) leading to change of MTJ resistance.

2) 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_{AND}, EN_M, EN_{OR})= (0,1,0). Now, if the path resistance is higher (/lower) than R_M, (i.e. R_{AP} (/R_P)), then the output of the SA produces High (/Low) voltage indicating logic '1'(/'0').

3) Computing Mode: Every two bits stored in the identical column can be selected and sensed simultaneously as depicted in Fig. 3a, employing modified row decoder [10]. Then, the equivalent resistance of such parallelly connected SOT-MRAMs and their cascaded access transistors are compared with a programmable reference by SA. Through selecting different reference resistances (EN_{AND}, EN_M, EN_{OR}), the SA can perform basic in-memory Boolean functions (i.e. AND and OR). The XOR logic can be realized with

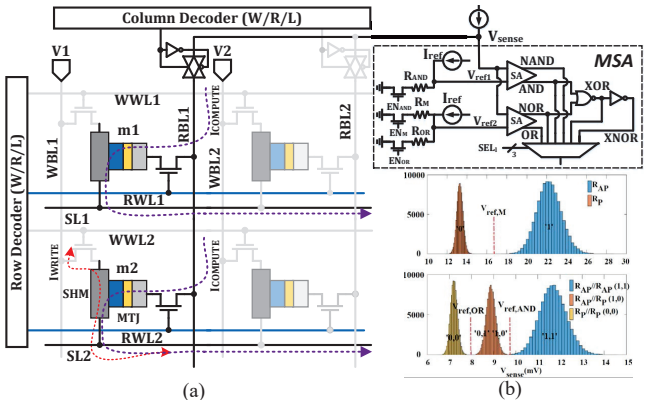


Figure 3: (a) In-memory processing sub-array based on SOT-MRAM, (b) Monte Carlo simulation result of V_{sense}.

two SAs (AND and NOR logic) and CMOS NOR gate using Modified SA (MSA). As shown in Fig. 3a, the operation of such sense circuit is determined by the control signals (EN_{AND}, EN_M, EN_{OR}), while the desired result is acquired by the select signal (SEL) of the output multiplexer. It is noteworthy that only one SA is used during AND/OR/memory read operation, in order to reduce the power consumption of sensing. To validate the variation tolerance of sense circuit, we have performed Monte-Carlo simulation with 100000 trials. A σ = 5% variation is added on the Resistance-Area product (RA_P), and a σ = 10% process variation is added on the TMR. The simulation result of sense voltage (V_{sense}) distributions in Fig. 3b shows the sense margin of in-memory computing. In this work, to avoid read failure (overlapping of V_{sense} distribution), only two fan-in in-memory logic is used. Parallel computing/read is implemented by using one SA per bit-line.

4 CMP-PIM ACCELERATOR ARCHITECTURE

In this section, we show that our proposed CMPNET can achieve two significant objectives in hardware implementation: (1) Reducing the energy consumption of convolution layers through utilizing efficient comparator-based computing, and (2) Accelerating the inference task. As shown in Fig. 2, four main computational blocks of CMPNET are first convolutional block (*Inception*), *Basic Block*, *Pooling* and *MLP*. The architectural diagram of the proposed CMP-PIM accelerator is shown in Fig. 4a consisting of Image and Kernel Banks, SOT-MRAM-based computational sub-arrays and a Digital Processing Unit (DPU) including three ancillary units (i.e. Quantizer, Batch Normalization and Activation Function). This architecture can be adjusted by Ctrl unit to process entire CMPNET. Assume Input fmaps (I) and Kernels (W) are initially stored in Image Banks and Kernel Banks of memory, respectively. As depicted in Fig. 4a, inputs can 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 computational sub-arrays (Fig. 4b). For realization of bit-wise operations, the proposed computational sub-array discussed in Section 3 is readily utilized such that ultra-efficient and parallel in-memory AND-XOR operations required for different blocks can be handled. We present in-memory magnitude comparator and in-memory convolver to handle main operations in CMPNET.

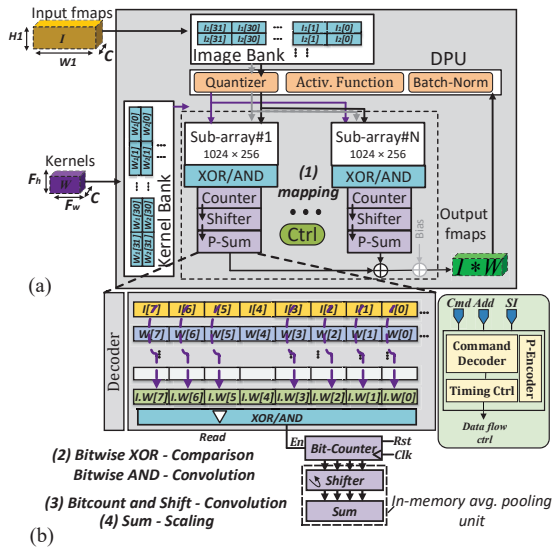


Figure 4: (a) General overview of the proposed CMP-PIM accelerator, (b) Computational sub-array.

4.1 In-Memory Magnitude Comparator

Comparison layer is the most critical component of the accelerator, as it is responsible for the most iterative layer which takes up the vast majority of the runtime of CMPNET. The unit must keep high throughput and resource efficiency while handling different input widths at run-time. While there are several designs for in-memory equality comparator in literature [19, 20], to the best of our knowledge, this work is the first proposing a magnitude comparator using in-memory bit-wise operations.

The initial idea here is to use in-memory XOR to perform bit-wise equality comparison from MSB to LSB. As shown in Fig. 5a, Pivot (P) and Input fmaps (F_j where $j = 1, 2, 3, \ldots$) are stored in consecutive memory rows. The operation begins with bit-wise comparison (i.e. XOR) of MSBs of P and one of the fmaps (F_1 in Fig. 5a) and continues towards LSBs. The result of i^{th} bit comparison ($P_i \oplus F_{j,i}$) is used as a determining factor for CMP-PIM's Ctrl unit to take next step. As tabulated in Ctrl decision and operation table (Table 2), when XOR result is "0" (indicating that two bits are equal), next less significant bit in two memory rows are selected for comparison and this process stops when XOR result is "1" (inequality). When inequality is detected, P_i content in memory array is read by Ctrl unit. Now, if P_i ="1"(/"0"), it denotes P is greater (/less) than F_j.

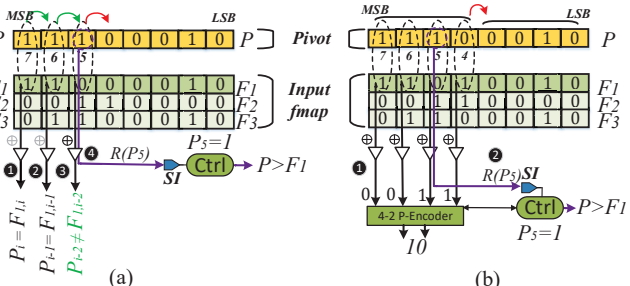


Figure 5: Realization of in-memory magnitude comparator in CMP-PIM: (a) 1-bit, (b) 4-bit.

Inspired by the proposed 1-bit magnitude comparator, we initialize the hardware with an optimization algorithm to further boost its efficiency by performing n-bit in-memory comparison. According to required bit-width for parallel comparison ($N_c \geqslant 2$), both pivot and input fmap can be split into P_r ($= \frac{\#bits}{N_c}$) portions. We take Fig. 5b as an instance to intuitively illustrate this process. As depicted, a 4-bit ($N_c = 4$ as an example) in-memory comparator can be efficiently designed leveraging intrinsic parallel bit-wise XOR operation by assigning multiple modified sense amplifier to each computational sub-arrays of CMP-PIM accelerator [10]. In this scheme, every four-bit (here P_r =2 considering 8-bit quantization for P and F), starting with MSB of P and F, can be compared concurrently. We further devise a 4-to-2 Priority Encoder in Ctrl unit so that it can detect and encode the first inequality position in two bit streams. Based on Fig. 5b, bit-wise XOR result ("0011") is sent to P-Encoder to encode bit position of the first inequality (here, "10" corresponding to 3^{rd} bit position).

Table 2: Control decision and operation of Fig. 5.

P_i	$F_{j,i}$	$\oplus$	Ctrl Decision	Operation
0	0	0	Continue and Compare	$P_{i-1} \oplus F_{j,i-1}$
0	1	1	Stop and Read	$Read(P_i)$
1	0	1	Stop and Read	$Read(P_i)$
1	1	0	Continue and Compare	$P_{i-1} \oplus F_{j,i-1}$

The rest of operations is similar to that of 1-bit magnitude comparator. Thus, comparator can efficiently compare input feature maps with pivot to aggregate feature values characterizing the local texture of an image. Clearly, number of P_r portions, which is inversely proportional to N_c, directly impacts CMP-PIM accelerator performance. The optimization of P_r will be analyzed later. After comparison operations, the outputs need to be scaled by 1×1 convolutional kernels, which can be efficiently implemented within accelerator using Sum unit since no multiplication is needed as fully discussed in previous section. Meanwhile, Sum unit combined with DPU could be used to implement pooling layer.

4.2 In-Memory Bit-Wise Convolver

Besides comparison layer as the basic block, there are some other layers in CMPNET, such as first convolutional layer (directly taking image as inputs, not replaced by comparison layer) and MLP block. Note that, MLP layers can be equivalently implemented by convolution operations using 1×1 kernels [3]. Thus, the rest layers could be implemented all by convolution computation by exploiting *logic AND*, *bitcount*, and *bitshift* as rapid and parallelizable operations [3]. Assume I is a sequence of M-bit input integers (3-bit as an example 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

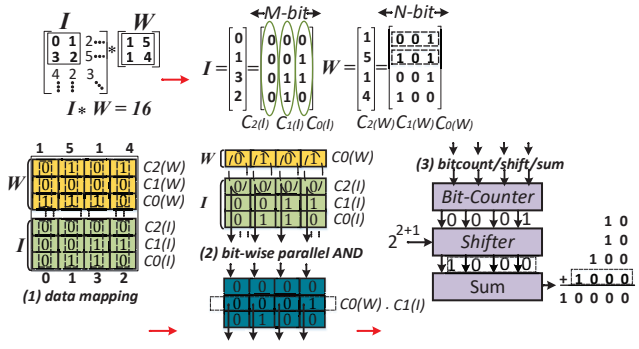


Figure 6: Realization of in-memory convolver in CMP-PIM.

$m = [0, M-1]$, 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:

$$I * W = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} 2^{m+n} bitcount(and(C_n(W), C_m(I))) \quad (4)$$

As shown in data mapping step in 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, computational sub-array can perform bit-wise parallel AND operation of $C_n(W)$ and $C_m(I)$ as depicted in Fig. 6. 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.

5 PERFORMANCE EVALUATION

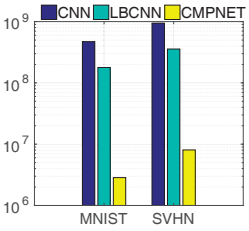
5.1 Accuracy

1) Experiment Setup: To demonstrate that CMPNET greatly saves computation resources while preserving good inference accuracy in comparison with CNN (as a baseline) and LBCNN, CMPNET is trained to perform popular pattern recognition task. For impartial comparison, all three networks have identical hyper-parameters in number of basic blocks, number of hidden neurons, etc. Two data-sets, MNIST and SVHN, are used to evaluate the performance of both algorithm accuracy and hardware implementation. Beyond that, the simulation is performed on Torch [21], a Matlab-like deep learning framework, with single GPU (Nvidia 1080Ti) configuration. We modify the code based on LBCNN[2]. For CMPNET, we set the fixed ternary kernel size ($kh \times kw$) as 7×7, basic block input-and output channels (p, q) as 256, the number of basic blocks is 5 for MNIST and 10 for SVHN, and 512 hidden neurons.

2) Experiment Result: According to the simulation results reported in Fig. 7a, the accuracy of our proposed CMPNET is very close to LBCNN in both full precision (32-bit) and quantized (8-bit for all weight parameters in convolution and MLP layers) versions. It shows the methodology that uses comparison layer to replace convolution and ReLU function causes very small accuracy degradation, but resulting in great hardware cost reduction. For MNIST and SVHN data-sets, both CMPNET and LBCNN have shown lower accuracy in comparison to CNN baseline. This accuracy gap could be narrowed through fine tuning the parameters, such as increasing the number of input and output channels, enlarging the dimension of convolution kernel or increasing the depth of convolution neural network [22, 23]. We then show the corresponding computational cost of convolutional layers in these three different neural network architectures in Fig. 7b. CMPNET shows almost two orders lower cost compared to CNN, which matches with the theoretical analysis in Table 1. For the rest of evaluation, we follow 8-bit quantized CMPNET.

Network	MNIST	SVHN
CNN	99.46%	95.01%
LBCNN	99.31%	94.02%
CMPNET (32-bit float)	99.4%	92.8 %
CMPNET (8-bit int)	99.4%	92.8 %

(a)

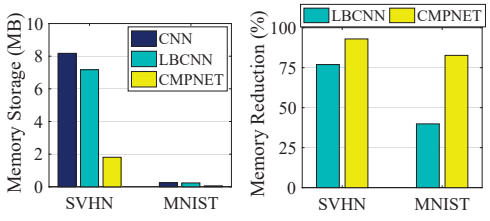


(b)

Figure 7: (a) Inference accuracy and (b) computational cost on MNIST and SVHN.

5.2 Memory Storage

The efficiency of entire CMPNET model in terms of memory storage required for processing MNIST and SVHN data-sets compared to different CNNs is shown in Fig. 8a. It can be seen that CMPNET can save large amount of memory compared to CNN and LBCNN. Furthermore, memory storage reduction of convolutional layers in LBCNN and CMPNET to CNN baseline is specifically reported in Fig. 8b. Based on the results, 92.97% and 82.69% reduction is achieved for SVHN and MNIST data-sets, respectively, compared to CNN baseline. This reduction mainly comes from reduced number of fixed ternary kernel and quantization as discussed in Sections 2.3 and 5.1, respectively.



(a) (b)

Figure 8: (a) Memory storage for processing SVHN and MNIST data-sets in CNN, LBCNN and CMPNET, (b) Memory storage reduction of convolutional layers in LBCNN and CMPNET to CNN baseline.

5.3 Energy and Delay

To perform energy and delay evaluation, the circuit level simulation is initially implemented in Cadence Spectre with NCSU 45nm CMOS PDK [24]. For modeling the SOT-MRAM cell, we incorporate the Landau-Lifshitz-Gilbert (LLG) equation to model the free layer magnetization dynamics and Non-Equilibrium Green's Function (NEGF) to calculate the SOT-MRAM resistance range (R_P,R_{AP}) with similar device parameters as in [25]. Then, we simulate the real-time resistance in the memory read/write paths w.r.t to the memory configuration. Accordingly, we massively modified the system level memory evaluation tool NVSim [26] to co-simulate with an in-house developed C++ code based on circuit level results. Control unit is designed through Verilog coding synthesized by Synopsys Design Compiler with 45nm CMOS Standard Cell libraries. Based

[2]LBCNN: https://github.com/juefeix/lbcnn.torch

on these cross-layer simulation results, CMP-PIM accelerator performance is evaluated. Setting N_c to 4, Fig. 9a shows the normalized log scale energy consumption of the proposed CMP-PIM accelerator running CMPNET under SVHN data-set compared to CNN and LBCNN designs. It can be seen that the proposed accelerator demonstrates up to ~ 94× and 3× better energy efficiency compared to CNN and LBCNN counterparts, respectively. Clearly, replacing the most computationally-expensive multiplication operations in convolutional layers of LBCNN and CNN with energy-efficient comparison operation (CMP) has yielded such significant improvement. To further explore the performance of CMPNET, Fig. 9b relatively compares inference delay per input image in above-mentioned designs. As shown, CMPNET achieves ~4.3× and 3× speed-up compared to CNN and LBCNN designs, respectively. Fig. 9c explores the trade-off between energy consumption and delay for inference in SVHN data-set w.r.t P_r. It can be seen that the larger P_r is, less delay and higher energy consumption are resulted. Based on this plot, we determine the optimum performance of CMP-PIM where N_c=4 and P_r=2.

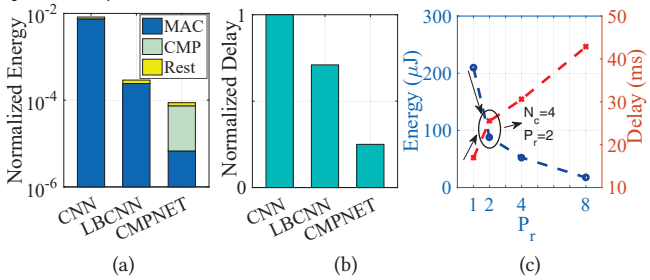


Figure 9: (a) Normalized log scale energy consumption and (b) Normalized Delay of CMP-PIM accelerator compared to CNN and LBCNN [7] under SVHN data-set. (c) Trade-off between energy and delay w.r.t P_r.

5.4 Hardware Mapping Comparison

In this subsection, we compare the hardware mapping results of similar CNN-PIM accelerators implemented by two promising resistive memories (i.e. RRAM [9] and SOT-MRAM herein) over two different data-sets in terms of energy and area under 45nm technology node. According to Table 3, among all the listed designs, CMP-PIM shows the best energy efficiency in two data-sets. It consumes 87.54 μJ/img for processing one image in SVHN data-set, which gets ~ 10× better energy efficiency compared to CNN-RRAM accelerator. In addition to energy efficiency of SOT-MRAM compared to RRAM, such significant improvement mainly comes from two sources: (1) RRAM design employs matrix splitting due to intrinsically limited bit levels of RRAM device so multiple sub-arrays should be simultaneously used in even simplest computations and (2) RRAM-based crossbar peripheral circuit such as buffers and DAC/ADC which contribute to more than 85% of energy consumption [9, 11].

Table 3: Performance estimation of CNN accelerators

Accelerator	SVHN		MNIST	
	Energy (μJ/img)	Area (mm^2)	Energy (μJ/img)	Area (mm^2)
CNN-RRAM [9]	850.42	0.09	18.39	0.054
BCNN-RRAM [9]	425.21	0.085	13.55	0.060
CMP-PIM- spin-CMOS	87.54	1.7	0.74	1.7

6 CONCLUSION

In this work, we proposed CMP-PIM as an energy-efficient and high-speed comparator-based processing-in-memory accelerator for neural network. CMP-PIM employed parallel computational memory sub-array as a fundamental processing unit based on SOT-MRAM design to process CMPNET. With almost similar inference accuracy on SVHN data-set, the CMP-PIM can get to ~ 94× and 3× better energy efficiency compared to CNN and Local Binary CNN (LBCNN), respectively. Besides, it achieves 4.3× speed-up compared to CNN-baseline design with identical network configuration.

ACKNOWLEDGEMENTS

This work is supported in part by the National Science Foundation under Grant No. 1740126 and Semiconductor Research Corporation nCORE

REFERENCES

[1] L. Cavigelli *et al.*, "Accelerating real-time embedded scene labeling with convolutional networks," in *DAC, 2015 52nd ACM/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] M. Rastegari *et al.*, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 525–542.
[5] T. P. Weldon *et al.*, "Efficient gabor filter design for texture segmentation," *Pattern recognition*, vol. 29, no. 12, pp. 2005–2015, 1996.
[6] T. Ahonen *et al.*, "Face description with local binary patterns: Application to face recognition," *IEEE TPAMI*, vol. 28, pp. 2037–2041, 2006.
[7] Juefei-Xu *et al.*, "Local binary convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 19–28.
[8] S. S. Sarwar *et al.*, "Gabor filter assisted energy efficient fast learning convolutional neural networks," *arXiv preprint arXiv:1705.04748*, 2017.
[9] T. Tang *et al.*, "Binary convolutional neural network on rram," in *22nd ASP-DAC*. IEEE, 2017, pp. 782–787.
[10] 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.
[11] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ISCA*. IEEE Press, 2016.
[12] Y. Kim *et al.*, "Write-optimized reliable design of stt mram," in *Proceedings of the 2012 ACM/IEEE ISLPED*. ACM, 2012.
[13] 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.
[14] L. Sifre and S. Mallat, "Rigid-motion scattering for image classification," Ph.D. dissertation, Citeseer, 2014.
[15] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.
[16] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *arXiv preprint arXiv:1610.02357*, 2016.
[17] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
[18] C.-F. Pai *et al.*, "Spin transfer torque devices utilizing the giant spin hall effect of tungsten," *Applied Physics Letters*, 2012.
[19] S. Aga *et al.*, "Compute caches," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 481–492.
[20] S. Jeloka *et al.*, "A 28 nm configurable memory (tcam/bcam/sram) using push-rule 6t bit cell enabling logic-in-memory," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 4, pp. 1009–1021, 2016.
[21] R. Collobert *et al.*, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, no. EPFL-CONF-192376, 2011.
[22] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[23] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.
[24] (2011) Ncsu eda freepdk45. [Online]. Available: http://www.eda.ncsu.edu/wiki/FreePDK45:Contents
[25] Z. He *et al.*, "High performance and energy-efficient in-memory computing architecture based on sot-mram," in *NANOARCH*. IEEE, 2017, pp. 97–102.
[26] 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.