DIMA: A Depthwise CNN In-Memory Accelerator

Shaahin Angizi, Zhezhi He and Deliang Fan
Department of Electrical and Computer Engineering, University of Central Florida, Orlando, FL 32816
{angizi,elliot.he}@knights.ucf.edu,dfan@ucf.edu

ABSTRACT

In this work, we first propose a deep depthwise Convolutional Neural Network (CNN) structure, called Add-Net, which uses binarized depthwise separable convolution to replace conventional spatial-convolution. In Add-Net, the computationally expensive convolution operations (i.e. Multiplication and Accumulation) are converted into hardware-friendly Addition operations. We meticulously investigate and analyze the Add-Net's performance (i.e. accuracy, parameter size and computational cost) in object recognition application compared to traditional baseline CNN using the most popular large scale ImageNet dataset. Accordingly, we propose a Depthwise CNN In-Memory Accelerator (*DIMA*) based on SOT-MRAM computational sub-arrays to efficiently accelerate Add-Net within non-volatile MRAM. Our device-to-architecture co-simulation results show that, with almost the same inference accuracy to the baseline CNN on different data-sets, *DIMA* can obtain ~1.4× better energy-efficiency and 15.7× speedup compared to ASICs, and, ~1.6× better energy-efficiency and 5.6× speedup over the best processing-in-DRAM accelerators.

1 INTRODUCTION

Deep Convolutional Neural Network (CNN) has achieved great success due to outstanding performance in image recognition over large scale data-sets such as ImageNet [1]. Following current trend, when going deeper in CNNs (e.g. ResNet employs 18-1001 layers), memory/computational resources and their communication have faced inevitable limitations. This has been 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. Model pruning [3], parameters quantization [4, 5] and rank refactorization [6] are the most widely-explored algorithmic approaches to mainly mitigate above challenges. Meanwhile, it has been proven that convolutional layers consume up to ~90% [1] of execution time and computational energy of whole CNN in both CPUs and GPUs, with the main purpose of feature extraction.

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, huge data communication energy and large leakage power consumption for storing network parameters in volatile memory [7]. To address these concerns, Processing-in-Memory (PIM) CNN accelerators, as a potentially viable way to address memory wall challenge, have been widely explored [7–9]. The key concept behind PIM is to embed logic units within memory to process data by leveraging the inherent parallel computing mechanism and exploiting large internal memory bandwidth. It could lead to remarkable saving in off-chip data communication energy and latency. An ideal PIM architecture should be capable of performing bulk bit-wise operations used in a wide spectrum of applications [10]. The proposals for exploiting SRAM-based PIM architectures can be found in recent literature [11]. However, PIM in context of main memory (DRAM- [8]) provides more benefits in recent years owing to the larger memory capacity and off-chip data communication reduction as opposed to SRAM-based PIM. However, the existing DRAM-based PIM architectures encounter several inevitable drawbacks, such as high refresh/leakage power, multi-cycle logic operations, operand data overwritten, operand locality, etc.

The PIM architectures have recently become even more popular when integrating with emerging Non-Volatile Memory (NVM) technologies, such as Resistive RAM (ReRAM) [7]. ReRAM offers more packing density (~ 2−4×) than DRAM, and hence appears to be competitive alternatives to DRAM. However, it still suffers from slower and more power hungry writing operations than DRAM [12]. Spin-Transfer Torque Magnetic Random Access Memory (STT-MRAM) [13] and Spin-Orbit Torque Magnetic Random Access Memory (SOT-MRAM) [9] are other promising high performance candidates for both the last level cache and the main memory, due to their low switching energy, non-volatility, superior endurance, compatibility with CMOS technology, etc. Meanwhile, MRAM technology is undergoing the process of commercialization [14]. Hence, PIM in the context of different NVMs, without sacrificing memory capacity, can open a new way to realize efficient PIM paradigms [7, 10].

In this work, we focus on massively reducing the computational complexity and parameter size in convolutional layers of CNN while preserving similar inference accuracy. It leads to an efficient hardware-friendly deep neural network architecture with binarized depthwise separable convolution (i.e. a refactorized form of normal convolutional layer) referred to as Add-Net. Our contributions can be summarized as: (1) For the first time, we evaluate the proposed Add-NET performance in objection recognition application using various hallmark datasets, including MNIST, SVHN, CIFAR10 and ImageNet. (2) We propose a Depthwise CNN In-Memory Accelerator (*DIMA*) to accelerate Add-Net within non-volatile SOT-MRAM. (3) We then perform detailed analysis about the effect of internal hyper-parameter configurations, which shows the trade-off between neural network accuracy and hardware resources (e.g. energy, memory storage/access and area).

ICCAD '18, November 5–8, 2018, San Diego, CA, USA

ACM ISBN 978-1-4503-5950-4/18/11…$15.00
https://doi.org/10.1145/3240765.3240799

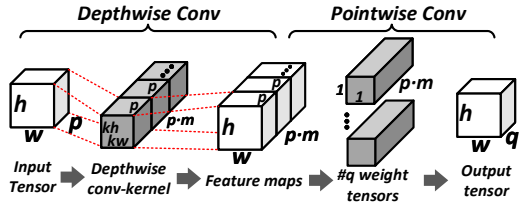

Figure 1: Data flow for depthwise separable convolutional layer. The default channel multiplier m is 1 in this figure.

2 BINARIZED SEPARABLE CONVOLUTION

2.1 Depthwise Separable Convolution

Recently, depthwise separable convolution [15] has been widely used in many state-of-the-art deep neural networks, such as MobileNet [16] and Xception [17], which replaces the traditional convolutional layers to reduce CNN computational cost and memory usage. As a factorized form of conventional spatial-convolution, the depthwise separable convolution consists of two parts: depthwise convolution and 1×1 convolution (a.k.a. pointwise convolution). The conventional spatial-convolution mainly performs channel-wise feature extraction, then combining those features to generate new representations. Such two-step task could be separately handled by depthwise and pointwise convolutions.

The operation of depthwise separable convolution is described in the form of data flow in Fig. 1 considering the input tensor in the dimension of $h \times w \times p$, which denote height, width and channel, respectively. In the depthwise convolutional layer (Depthwise Conv), each channel of input tensor performs convolution with m kernels in the size of $kh \times kw$ correspondingly, which produces $p \cdot m$ feature maps. m is defined as *channel multiplier* herein. Those generated feature maps are concatenated along the depth dimension as a tensor in size of $h \times w \times (p \cdot m)$[1], which is taken as the input to pointwise convolutional layer (Pointwise Conv). Contrary to the distinctive depthwise convolution, pointwise layer is just a normal spatial-convolutional layer with 1×1 convolution kernel size. Thus, it only linearly combines the $p \cdot m$ input feature maps to generate new representations with q output channels.

2.2 Add-Net

In previous works, such as BNN [18] and XNOR-NET [5], both convolution kernel and activation function are binarized, which converts the computationally-expensive convolution operation into bit-wise logic operations and bit-count. Such aggressive model compression method reduces the hardware resource utilization at the cost of performance degradation to some extent. In this work, we focus on the weight binarization while keeping the input tensor of each convolutional layer quantized in multi-bit (8-bit in 3). Accordingly, we construct a deep neural network with the proposed binarized depthwise separable convolution referring to the topology of ResNet [19], called Add-Net. The block diagram of Add-Net is depicted in Fig. 2, which sequentially consists of an *Inception block*[2] (3×3 spatial convolution, Batch-normalization and ReLU), *N-Basic block*, *Average pooling* and *Multi-Layer Perceptron*

[1]The default hyper-parameter configurations in convolutional layers are: kernel size= 3×3, stride = 1, padding = 1, no bias.

[2]Similar as previous works of binarized/quantized neural network [4, 5], we do not introduce binarization to the inception block.

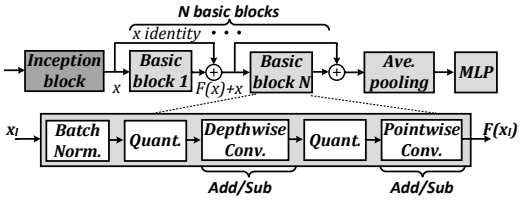

Figure 2: Block diagram of Add-Net.

(MLP). As the key component in our proposed neural network, basic block includes batch normalization, quantization, depthwise convolution and pointwise convolution both with binarized weights. Similar as XNOR-NET [5], we place the batch normalization before the convolutional layer with binarized weight. In summary, the response of basic block can be described as:

$$x_{l+1}^{t} = \sum_{s=1}^{p \cdot m} Quant\left(W_l'^{s} * BN(x_l^s)\right) \cdot \alpha_{l,s}^{t} \tag{1}$$

where $s \in [p \cdot m]$ and $t \in [q]$ denote the input channel and output channel, respectively. W_l' is the learned depthwise convolution kernel with binarized weight. l is the index of basic block, while p is the number of input channels of l_{th} basic block. α_l is the learned weight of pointwise convolution. $BN()$ is the batch normalization function. $Quant()$ is the quantized activation function as in [4].

In this work, we choose binary weights for both depthwise and pointwise conv., so there are no multiplication in those convolutional layers. Moreover, in order to make the computation in the convolutional layers could be easily implemented by our proposed in-memory computing framework (discussed in section 3), we introduce the quantized activation function to divide the intermediate tensor into multi-level (default as 8-bit in this work). The formulas of binarization function $Bin()$ and multi-bit quantization function $Quant()$ in the forward path can be accordingly described as [4]:

$$Bin(r) = E(|w_l|) \cdot Sign(r) = \begin{cases} +E(|w_l|) & if\ r \geq 0 \\ -E(|w_l|) & otherwise \end{cases} \tag{2}$$

$$Quant(r) = \frac{2}{2^k - 1} round((2^k - 1)(\frac{tanh(r)}{2max(|tanh(r)|)} + 0.5)) - 1 \tag{3}$$

where $E(|w_l|)$ calculate the mean of the absolute value of weights in layer l as the layer-wise scaling factor, and such scaling factor is re-calculated for each input batch during the training process. r is the element either in intermediate tenor or weight tensor. k is the targeted number of bits for quantization. Beyond that, the Straight-Through Estimator (STE) [20] is applied to calculate gradient in backward path due to the quantization function owns zero derivatives almost everywhere, where the mathematical representation is:

$$\frac{\partial g}{\partial r} = \begin{cases} \frac{\partial g}{\partial r'} & if\ |r| \leq 1 \\ 0 & otherwise \end{cases} \tag{4}$$

where g is the loss, r and r' is the input and output of the quantization function, correspondingly. In the backward path, the gradient is clipped when the input of the quantization function is out of the range from -1 to +1. For quantizing/binarizing the weights in convolutional or fully-connected layers, we retain the weights in real value for the backward training. For the forward path, the weights are binarized using function in Eq. 2. As a result of this binarization, there are no multiplications for convolution operations

and the main computations are 8-bit *add/sub* operations. Note that, the scaling factor is shared by the whole network layer and will be implemented after binarized convolution computation.

In order to clearly show the benefits of Add-net, we analyze the hardware cost of standard spatial convolution and binarized depthwise separable convolution in terms of computational cost and parameter size. As depicted in Table 1, not only the multiplications in depthwise separable convolution operations are fully replaced by the hardware-efficient *add/sub*, but also the number of operations is reduced approximately by a factor of $\sim m/(kh \cdot kw)$. For parameter size, using the floating point depthwise separable convolution to replace the normal convolutional layer could achieve a compression rate of $\sim m/(kh \cdot kw)$. If further binarized, the model size could be compressed to $\sim m/(kh \cdot kw) * 32$.

Table 1: Hardware cost of standard convolution in CNN and Add-Net. n is the number bits of weights in pointwise layer.

	Computation Cost		Memory Cost
	Mul–$O(N^2)$	Add/Sub–$O(N)$	
CNN	$h \cdot w \cdot kh \cdot kw \cdot p \cdot q$	$h \cdot w \cdot kh \cdot kw \cdot p \cdot q$	$kh \cdot kw \cdot p \cdot q \cdot 32$
This work	–	$h \cdot w \cdot kh \cdot kw \cdot p \cdot m$ $+h \cdot w \cdot p \cdot m \cdot q$	$kh \cdot kw \cdot p \cdot m$ $+p \cdot m \cdot q$
$\frac{This\ work}{CNN}$	0	$\frac{m}{q} + \frac{m}{kh \cdot kw}$	$\frac{m}{q \cdot 32} + \frac{m}{kh \cdot kw \cdot 32}$

3 DIMA ARCHITECTURE

In the following sections, we show that our proposed binarized depthwise separable CNN (Add-Net) can achieve four significant objectives in hardware implementation: (1) Reducing the energy consumption of convolutional layers through utilizing efficient *add/sub*-based computing; (2) Reducing the memory (i.e. network parameter) storage and access required for feature extraction; (3) Reducing the computation area overhead and (4) Accelerating inference task within memory. The architectural diagram of the proposed Depthwise CNN In-Memory Accelerator (*DIMA*) is shown in Fig. 3a 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 Add-Net. Assume Input fmaps (I) and Kernels (W) are initially stored in Image Bank and Kernel Bank of memory, respectively. Except for the inception block, inputs/kernels need to be constantly quantized (to n-bit) before mapping into computational sub-arrays which are designed to handle the computational load of *DIMA* employing in-memory computing. However, quantized shared kernels can be utilized for different inputs in the pointwise convolutional layer. This operation is basically performed using DPU's Quantizer (see DPU in Fig. 3a) and then results are mapped to the parallel sub-arrays.

3.1 Computational Sub-array

Fig. 3b shows the presented in-memory computing sub-array architecture which is accordingly implemented by SOT-MRAM in Fig. 3c (A). This architecture mainly consists of Write Driver (WD) (B), Memory Row Decoder (MRD) (C), Memory Column Decoder (MCD), Sense Amplifier (SA) (D), multiplexers (MDMUX, GMUX) and FA/FS unit and can be adjusted by Ctrl unit (E) to work in dual mode that perform both memory write/read and in-memory logic operations (using two distinct methods). 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 located in computational sub-arrays 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 operations as follow:

1) Memory Write/Read: To *write* a data bit in any of the SOT-MRAM cells (e.g. M1 in Fig. 3c (A)), write current should be injected through the SHM (Tungsten, $\beta - W$ [21]) of SOT-MRAM. Therefore, WWL1 should be activated by the MRD where SL1 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. 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 (D), 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). 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').

2) Computing Mode: The proposed computational sub-array is designed to perform the computation between in-memory operands using two distinct methods referred to as 2-row activation and 3-column activation. The main ideas behind developing 2-row and 3-column activation methods are to perform bulk bit-wise in-memory AND operation and in-memory addition/subtraction, respectively.

In the 2-row activation method, every two bits stored in the identical column can be selected and sensed simultaneously employing modified MRD [10], as depicted in Fig. 3c (A). Then, the equivalent resistance of such parallel 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). For AND operation, R_{ref} is set at the midpoint of $R_{AP}//R_P$ ('1','0') and $R_{AP}//R_{AP}$ ('1','1'). As an example, considering the data organization shown in Fig. 3b where A and D operands correspond to M1 and M4 memory cells in Fig. 3c (A), respectively, 2-row activation method generates AD after SA. 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 (V_{sense}) distributions showed the sufficient sense margin of in-memory computing. In this work, to avoid read failure, only two fan-in in-memory logic is used. Parallel computing/read is implemented by using one SA per bit-line.

In the 3-column activation method, we have devised a Mode demultiplexer (MDMUX) right after SAs to switch between memory mode and this new computing method. As can be seen in block-level sub-array architecture (Fig. 3b), the output of each SA is routed to MDMUX. According to the mode selector, output data can be routed to either GMUX or FA/FS unit. The key idea behind exploiting a CMOS FA/FS unit is to realize a fast in-memory full adder (/subtractor) after SAs to efficiently process the data avoiding inevitable

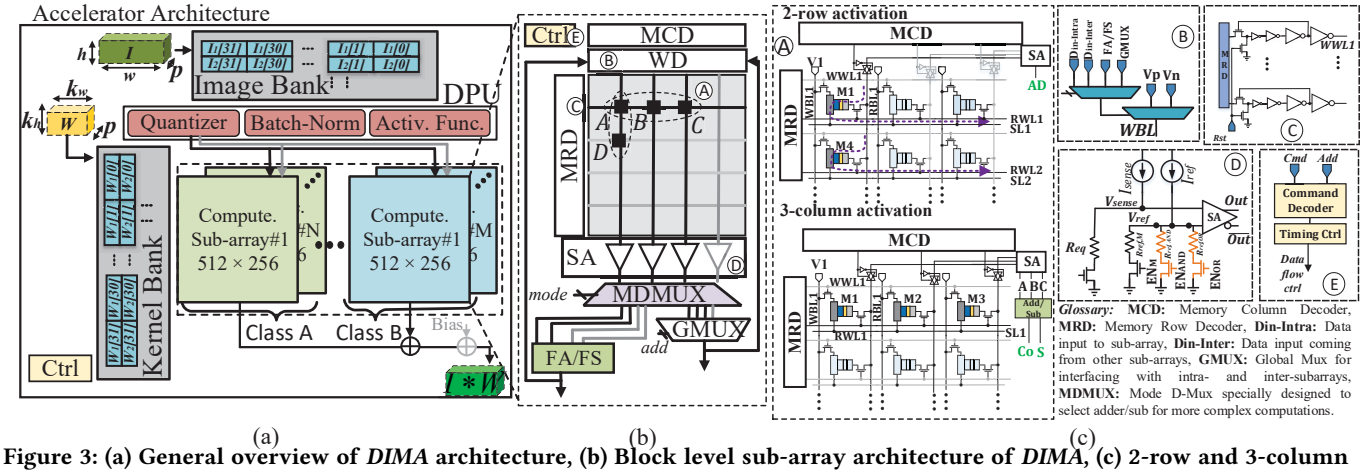

Figure 3: (a) General overview of *DIMA* architecture, (b) Block level sub-array architecture of *DIMA*, (c) 2-row and 3-column activation methods of computational sub-array and functional blocks.

operand write-back in conventional in-memory adder designs as well as accelerating in-memory processing. For this computation method, MCD is modified (similar to that of MRD) such that it can activate more than one RBL at the same time. As a result, more than one column can be sensed and routed from SAs to FA/FS unit. Assume *A*, *B* and *C* operands (in Fig. 3b) correspond to M1, M2 and M3 memory cells in Fig. 3c (A), respectively, the 3-column activation yields $Sum(/Difference)$ and $Carry(/Borrow)$ bits.

3.2 In-Memory Binary-Weight Convolver

DIMA offers in-memory Binary-Weight Convolver and in-memory Bit-Wise Convolver to handle main operations of the Add-Net based on the proposed in-memory computing schemes. From hardware implementation perspective, there are two types of convolution operations in Add-Net that need to be taken into account. The first one is binary-weight convolution located in basic block with binarized kernels and quantized inputs. The second one is bit-wise convolution located in inception layer and MLP in which convolution between different bit-width inputs and kernels requires bulk bit-wise operations. While both types can be implemented with either of *DIMA*'s convolution schemes, as will be described accordingly, we still propose two distinct convolution methods to boost the accelerator's performance with smaller area overhead. As shown in Fig. 3a, two classes of sub-arrays (*A* and *B*) are respectively allocated to in-memory binary-weight and bit-wise convolvers. Note that, all the computational sub-arrays support both memory and computing modes and only differs from required add-on hardware which will be discussed in the following. The ratio between *A* and *B* is specially determined considering the network specifications and performance constraints.

As the main operations of Add-Net, depthwise and pointwise convolutions are the most critical units of the accelerator, as they are responsible for the most iterative block which takes up the vast majority of the run-time in Add-Net. These units must keep high throughput and resource efficiency while handling different input widths at run-time. As discussed earlier, the main operation in this block is *add*/*sub*. So, here we propose an in-memory binary-weight convolver based on 3-column activation method to handle multi-bit addition/subtraction operations. While there are few designs for in-memory adder/subtractor in literature [22], to the best of our knowledge, this work is the first proposing a fast (2-cycle) and parallelable in-memory *add*/*sub* method. As seen in Section 3.1, 3-column activation method of the *DIMA* can be utilized to perform one-bit in-memory *add*/*sub* operation quite efficiently in one cycle (memory read). After the computation, the results need to be stored in the sub-array to be prepared for next computing round. This can be fulfilled using the modified WD and MRD in one cycle (memory write). We take the data organization shown in Fig. 4a as an instance to show 4-bit addition process. As can be seen each computational sub-array is able to implement *n*-bit *add*/*sub* operations (4-bit, herein) in $2 \times n$ cycles. In this case, *C3S3S2S1S0* is produced as the summation of *A3A2A1A0* and *B3B2B1B0* operands.

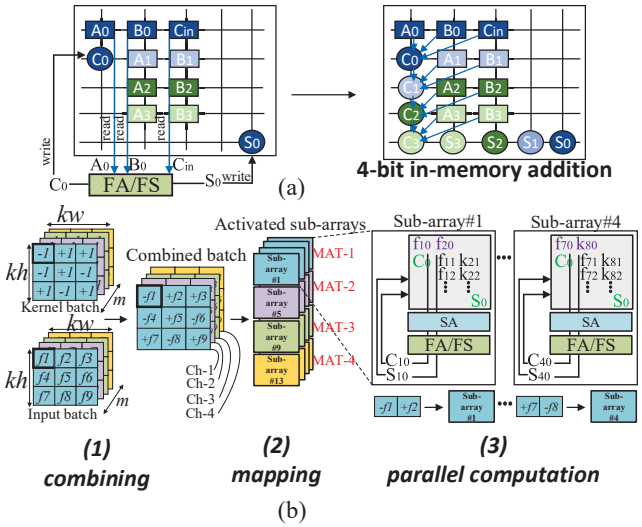

Figure 4: (a) Multi-bit in-memory addition in *DIMA*, (b) In-memory binary-weight convolver in *DIMA*.

Fig. 4b shows the requisite data organization and computation of depthwise and pointwise convolutional layers. Initially, *m* channels (here, 4) in the size of $kh \times kw$ (here, 3×3) are selected from input batch and accordingly produce a combined batch w.r.t. the corresponding {-1,+1} kernel batch. This combination is readily accomplished by changing the sign-bit of input data corresponding

to its kernel data. The combined batch is then mapped to the designated computational sub-arrays (Class A). Considering 16-activated sub-arrays (within 4 memory matrix (MAT) structures as depicted Fig. 4b), each combined batch's channel (Ch) can be processed using four parallel sub-arrays. Here, Ch-1 to Ch-4 are respectively mapped to MAT-1 to MAT-4. After mapping, the parallel activated sub-arrays of *DIMA* operate to produce the output feature maps leveraging the same addition/subtraction method shown in Fig. 4a.

3.3 In-Memory Bit-Wise Convolver

Besides depthwise and pointwise convolutional layers in the basic block, there are some other layers in the proposed CNN, such as inception layer (directly taking image as inputs, not replaced by basic block), pooling layer and MLP block. Note that, MLP layers can be equivalently implemented by convolution operations using 1×1 kernels [4]. Thus, the rest layers could be implemented all by convolution computation by exploiting *logic AND*, *bitcount*, and *bitshift* as rapid and parallelizable operations [4, 9]. Assume I is a sequence of M-bit input integers (3-bit as an example in Fig. 5) 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.

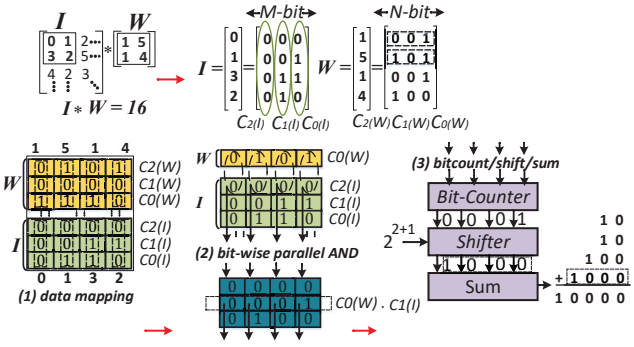

Figure 5: In-memory bit-wise convolver in *DIMA*.

We index the bits of each I_i element from LSB to MSB with $m = [0, M-1]$. 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:

$$I * W = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} 2^{m+n} bitcount(and(C_n(W), C_m(I))) \quad (5)$$

As shown in data mapping step in Fig. 5, $C_2(W)$-$C_0(W)$ 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. 5. 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. 5, "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. While the computationally-expensive bulk AND operation can be readily performed leveraging *DIMA*'s 2-row activation method in entire memory sub-arrays like the one in [9], we still need to incorporate CMOS bit-counter and shifter units in some of computational sub-arrays (Class B). 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.

4 PERFORMANCE EVALUATION

4.1 Accuracy

4.1.1 DNN Software setup. Add-Net architecture is constructed under the Pytorch framework, which recently optimized its depthwise convolution backend CUDA library to accelerate the training process. We employ the Adam as optimizer to minimize the cross-entropy loss. Owing to the large variation of intermediate output caused by frequently-adjusted binary weights and binarization activation function, small learning rate are preferable. We set the initial learning rate as 0.001, which is reduced to 0.0001 through scheduling.

4.1.2 Results. The experiments for Add-Net are conducted using two NVIDIA 1080Ti GPUs for image recognition application with four common image data-sets i.e. MNIST, SVHN, CIFAR-10 and ImageNet. The test accuracy of Add-Net on these data-sets are reported in Table 2. The model configurations for four data-sets are as follow: (1) MNIST: 16 input channels, 5 basic blocks, 128 hidden neuron, 64 batch size, 3×3 kernel size, 4 channel multiplier. (2) SVHN: 128 input channels, 5 basic blocks, 512 hidden neuron, 128 batch size, 3×3 kernel size, 4 channel multiplier. (3) CIFAR10: 512 input channels, 10 basic blocks, 512 hidden neuron, 32 batch size, 3×3 kernel size, 4 channel multiplier (4) ImageNet: ResNet-18 topology with conventional 3x3 spatial convolution layers replaced by the basic-blocks described in Fig. 2, 256 batch size, 4 channel multiplier. The baseline CNN reported in Table 2 uses normal spatial convolution without any quantization, while the Add-Net uses the aforementioned quantization and binarization techniques in addition to the depthwise separable convolution layer. The hyper-parameters configurations are identical between baseline and this work. The results show that Add-Net can achieve the state-of-the-art test accuracy, which is close to its normal spatial-convolution counterparts. In large scale ImagNet simulation, we adopt 8-bit activation quantization and binary depthwise separable convolution, the additional 2× model size reduction is at the cost of ~ 2% accuracy degradation compared to BWN [5]. Note that, we didn't report accuracy of BinaryConnect and BNN in ImageNet since they do not report accuracy of binary Resnet.

Table 2: Top-1 inference accuracy (%) of MNIST, SVHN, CIFAR10 and ImageNet.

	Baseline CNN	Add-Net (this work)	BinaryConnect [23]	BNN [18]	BWN [5]
MNIST	99.46	99.45	98.99	98.60	–
SVHN	94.29	94.73	97.85	97.49	–
CIFAR-10	91.25	89.54	91.73	89.85	–
ImageNet	65.39	58.80	–	–	60.8
Comp. Rate	1×	64×	–	–	32×

4.1.3 Effect of hyper-parameters. We examine the effect of hyper-parameters on Add-Net performance. Since the neural networks are trained from scratch instead of fine-tuning from the pre-trained model, we chose SVHN as the representative experiment data-set to report the results. It can be seen that by increasing the channel multiplier from 1 to 16, the accuracy gradually increases.

Table 3: The SVHN test accuracy w.r.t channel multiplier (m)

	Spatial Conv.		Add-Net				
weight-precision	32-bit	1-bit	1-bit	1-bit	1-bit	1-bit	8-bit
input-precision	32-bit	8-bit	8-bit	8-bit	8-bit	8-bit	8-bit
m	-	-	1	2	4	8	16
Top-1 Test Accuracy	94.29	94.84	93.82	94.61	94.73	94.84	95.02

4.2 Memory Storage

The comparison of model efficiency in terms of memory usage required for processing one convolutional layer between Add-Net and CNN baseline (32-bit and 8-bit) is shown in Fig. 6a. We observe that the binarized convolution blocks in Add-Net are much more memory-friendly than the CNN counterpart, when using small channel multiplier m. The memory storage reduction of a convolutional layer in Add-Net to 32-bit CNN baseline is specifically reported in Fig. 6b. For instance, 97.2% and 88.7% reduction are achieved when m equals 1 and 4, respectively. This reduction mainly comes from the reduced number of fixed kernel as discussed in section 2.2. However, for eliminating the accuracy gap between Add-Net and baseline CNN, larger channel multiplier is desired. For instance 55.1% reduction is obtained setting m to 16. Considering a trade-off between accuracy, memory storage, energy consumption and performance, the m parameter can be precisely tuned to meet a specific hardware's requirement. For instance, sacrificing the inference accuracy by ~0.3% (i.e. changing m from 16 to 4) in SVHN data-set leads to ~4× memory saving with a considerably lower energy and higher performance as will be discussed in next subsections.

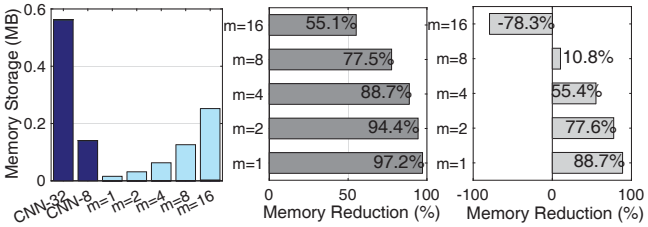

Figure 6: (a) Memory storage of CNN vs. Add-Net in different configuration for a single conv. layer, (b) Memory storage reduction of Add-Net to 32-bit and (c) 8-bit CNN baseline.

We have also plotted the memory storage reduction of Add-Net to a sample 8-bit CNN baseline as a quantized network in Fig. 6c. Note that such CNN shows a comparable accuracy to Add-Net or 32-bit CNN based on Table 3. This plot intuitively shows that in order to get a higher accuracy, Add-Net will require larger memory storage compared to 8-bit CNN in a specific case (m=16) which reverses the memory reduction trend.

4.3 Area

Fig. 7 shows the breakdown of the area overhead resulted from *DIMA*'s add-on hardware to memory chip. Our experiments show that, in total, *DIMA* imposes 2.8% area overhead to the original memory die, where Pinatubo [10], DRISA [8] incur 0.9% and 5% area overhead, respectively. We observe that the modified controller and drivers contribute more than 50% of this area overhead in a memory group. It is obvious that enlarging the chip area brings in a higher performance for *DIMA* and other designs due to the increased number of sub-arrays, though the die size directly impacts the chip cost. Therefore, in order to have a fair comparison with different accelerators, the area-normalized results (performance/energy per area) will be reported henceforth.

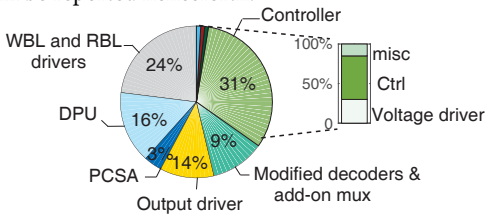

Figure 7: Area overhead of *DIMA* in a memory group.

Fig. 8a illustrates and compares the computational area overhead required by conventional CNN and Add-Net for performing their main operations. As can be seen, eliminating the *mul* in depthwise separable convolution operations in Add-Net brings considerable area-efficiency (up to 95.2%) compared to CNN that can be exploited to enable higher computation parallelism. However, to fit a deep neural network into a low-end ASIC, the number of logic cells may come to shortage. The normal countermeasure is to split the computation and multiplex the computation kernel, which restraints the throughout. The better solution is to use the proposed binarized depthwise separable convolution with the fine-tuning.

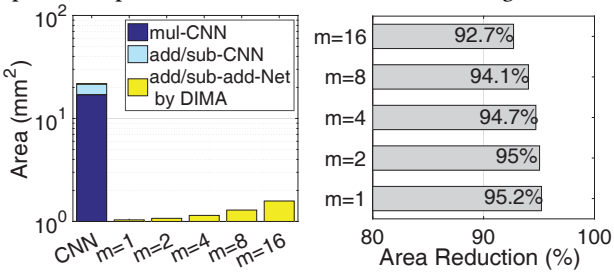

Figure 8: (a) Normalized log-scaled area overheard imposed by CNN and Add-Net for the computation, (b) Area reduction of the proposed network to CNN baseline.

4.4 Hardware Setup

In the following, we compare *DIMA* running Add-Net with state-of-the-art inference acceleration solutions (based on DRAM, ReRAM, ASIC and GPU) running a baseline CNN.

4.4.1 Modeling Setup. **Bit-width configuration:** For *DIMA*, 8-bit quantized Add-Net with distinct configurations (m=1, 2, 4, 8, and 16) is considered for evaluation. In order to have a fair comparison, we employ a particular bit-width for weight and activation <W:A> (<1:8>) for DRAM-, ReRAM, ASIC- and GPU-based acceleration methods rather than full-precision. **Data-set:** The SVHN data-set

[24] is selected for evaluation. The images are re-sized to 40×40 and fed to the model. **CNN Model:** A CNN with 6 (bit-wise) convolutional layers, 2 (average) pooling layers and 2 MLP layers is adopted. MLP layers are equivalently implemented by convolutions.

4.4.2 Accelerators' Setup. ***DIMA:*** We configure the *DIMA*'s memory sub-array organization with 256 rows and 512 columns per mat organized in a H-tree routing manner, 2×2 mats (with 2/2 and 2/2 as Row and Column Activations) per bank, 8×8 banks (with 1/8 and 8/8 as Row and Column Activations) per group; in total 16 groups and 512Mb total capacity. The ratio of computational sub-array (class A: class B) is obtained 7:2. To assess the performance of *DIMA* as a new PIM platform, a comprehensive device-to-architecture evaluation framework along with two in-house simulators are developed. First, at the device level, we jointly use the Non-Equilibrium Green's Function (NEGF) and Landau-Lifshitz-Gilbert (LLG) with spin Hall effect equations to model SOT-MRAM bitcell [25, 26]. For the circuit level simulation, a Verilog-A model of 2T1R SOT-MRAM device is developed to co-simulate with the interface CMOS circuits in Cadence Spectre and SPICE. 45nm North Carolina State University (NCSU) Product Development Kit (PDK) library [27] is used in SPICE to verify the proposed design and acquire the performance of designs. Second, an architectural-level simulator is built based on NVSim [28]. Based on the device/circuit level results, our simulator can alter the configuration files (.cfg) corresponding to different array organization and report performance metrics for PIM operations. The controllers and add-on circuits are synthesized by Design Compiler [29] with an industry library. Third, a behavioral-level simulator is developed in Matlab calculating the latency and energy that *DIMA* spends considering a particular application. In addition, it has a mapping optimization framework for the CNN. ***DRAM:*** We developed a DRISA-like [8] accelerator for low bit-width CNNs. Two different computing methods of DRISA named 3*T*1*C* and 1*T*1*C-adder* were selected for comparison. The 3T1C uses DRAM cells themselves for computing and performs NOR logic on BLs. However, 1T1C-adder exploits a large *n*-bit adder circuit for *n*-bit BLs after SAs. We modified CACTI [30] for evaluation of DRAM's solutions. Similar to [8], the controllers and adders were synthesized in Design Compiler [29]. ***ReRAM:*** A Prime-like [7] accelerator with two full functional (FF) sub-arrays and one buffer sub-array per bank (totally 64 sub-arrays) were considered for evaluation. In FF subarrays, for each mat, there are 256×256 ReRAM cells and eight 8-bit reconfigurable SAs. For evaluation, NVSim simulator [28] was extensively modified to emulate Prime functionality. Note that the default NVSim's ReRAM cell file (.cell) was adopted for the assessment. ***ASIC:*** We developed a YodaNN-like [2] ASIC accelerator. To have a fair comparison, we select two versions with either 8×8 tiles (33MB eDRAM) or 16×16 tiles (129MB eDRAM). Accordingly, we synthesized the designs with Design Compiler [29] under 45 nm process node. The eDRAM and SRAM performance were estimated using CACTI [30]. ***GPU:*** We used the NVIDIA GTX 1080Ti Pascal GPU. It has 3584 CUDA cores running at 1.5GHz (11TFLOPs peak performance). The energy consumption was measured with NVIDIA's system management interface. Similar to [8], we scaled the achieved results by 50% to exclude the energy consumed by cooling, etc. Accordingly, based on 8-bit configuration of <A>, we aggressively scaled GPU results by ×4 to get the peak performance for the quantized network. Note that, GPU doesn't support fixed point CNN and real scale ratio should be less than four [8].

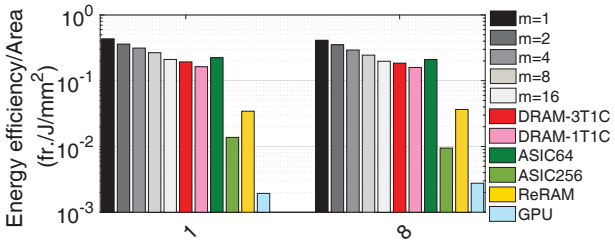

Figure 9: Energy-efficiency of different accelerators normalized to area (Y-axis=Log scale).

4.5 Energy

Fig. 9 shows the *DIMA*'s energy-efficiency results (frames per joule) on Add-Net compared to different accelerators for performing a similar task with a batch size of 1 and 8. As can be seen, the larger the m is, the lower energy-efficiency is obtained, we nevertheless take $m = 4$ as a mean to compare with the other platforms. As shown, *DIMA* solution offers the highest energy-efficiency normalized to area compared to others owning to its fast, energy-efficient and parallel operations. We observe that *DIMA*'s solution is ~ 1.9× and 1.6× more energy-efficient than that of DRAM-3T1C and 1T1C-adder, respectively. In addition to large refresh power of DRAM-based PIM accelerators [8], they are dealing with a destructive *data-overwritten* issue due to the charge sharing characteristic of capacitors. It means that the result of computation will ultimately overwrites the operands. To solve this issue in the context of DRAM, *multi-cycle operations* are set forth which has further degraded PIM performance. Note that, despite the n-bit adder located after SAs in DRAM-1T1C-adder solution will bring higher performance compared to 1T1C, it has limited its energy-efficiency. We observe that *DIMA* solution is 1.4× more energy-efficient than the best ASIC solution. This energy reduction mainly comes from two sources: 1) standard convolution is replaced with energy-efficient depthwise separable convolution and 2) *mul* in convolution is converted to *add*/*sub* due to binarization. Fig. 9 also shows that *DIMA* obtains ~8.4× saving in energy compared to ReRAM solution.

4.6 Performance

Fig. 10 shows the *DIMA* performance (frames per second) results on Add-Net in different configuration space of channel multiplier. We observe that the smaller the m is, the higher performance per area is obtained for *DIMA* with higher energy-efficiency (Fig. 9). Fig. 10 demonstrates that *DIMA* solution ($m = 4$) is 5.6× faster than the best DRAM solution (1T1C-adder) and 15.7× faster than ASIC-64 solution. This is mainly because of (1) ultra-fast and parallel in-memory operations of *DIMA* compared to multi-cycle DRAM operations and (2) the existing mismatch between computation and data movement in ASIC designs and even 1T1C-adder solution. As a result, ASIC-256 with more tiles does not show higher performance. We can also observe that the larger the batch is, the higher performance is obtained for *DIMA* solution compared DRAMs owning to the its more paralleled computations. Additionally, it can be seen that *DIMA* is 11.2× faster that ReRAM solution. Note that ReRAM design employs matrix splitting due to intrinsically limited bit levels of RRAM device so multiple sub-arrays are occupied. Besides. ReRAM crossbar has a large peripheral circuit's overhead such as

buffers and DAC/ADC which contribute more than 85% of area [7].

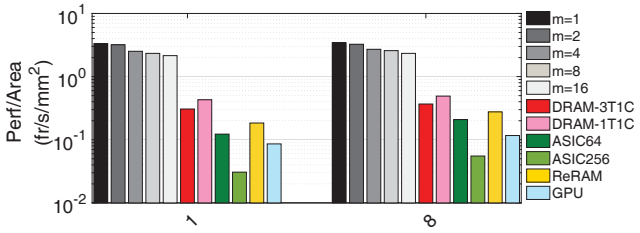

Figure 10: Performance evaluation of different accelerators normalized by area (Y-axis=Log scale).

4.7 Memory Wall

Fig. 11a depicts the memory bottleneck ratio i.e. the time fraction at which the computation has to wait for data and on-/off-chip data transfer obstructs its performance (memory wall happens). The evaluation is performed according to the peak performance and experimentally extracted results for each platform considering number of memory access. The results[3] show the *DIMA*'s favorable solution for solving memory wall issue. (1) We observe that *DIMA* (with different m size) and DRAM-3T1C solutions spend less than ~14% time for memory access and data transfer. However, ASIC- and DRAM-1T1C accelerators spend more than 90% time waiting for the loading data. (2) In the larger batch size, ReRAM solution shows lower memory bottleneck ratio even compared with *DIMA*. This comes from two sources: (1) increased number of computational cycles and (2) unbalanced computation and data movement of *DIMA*'s binary-weight convolver due to limitation in number of activated sub-arrays when number of operands increases. The less memory wall ratio can be interpreted as the higher resource utilization ratio for the accelerators which is plotted in Fig. 11b. We observe that *DIMA* can efficiently utilize up to 50% of its computation resources. Note that the smaller the m is, the higher resource utilization is achieved. Overall, *DIMA*, DRAM-3T1C and ReRAM solutions can demonstrate the highest ratio (up to 65% which reconfirms the results reported in Fig. 11a.

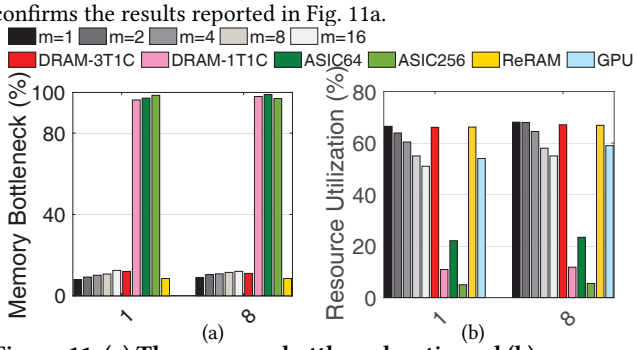

Figure 11: (a) The memory bottleneck ratio and (b) resource utilization ratio.

5 CONCLUSION

In this paper, we first proposed a new deep CNN architecture, called Add-Net. In Add-Net, the computationally-expensive convolution operations are converted into hardware-friendly Addition/Subtraction operations. Then, we proposed a Depthwise CNN In-Memory Accelerator (*DIMA*) based on SOT-MRAM computational sub-arrays to efficiently process the Add-Net. Our device-to-architecture co-simulation results shows that with a comparable inference accuracy to the baseline CNN on the different data-set, *DIMA* can obtain ~1.4× better energy-efficiency and 15.7× speedup compared to ASICs, and, ~1.6× better energy-efficiency and 5.6× speedup over the best DRAM-based accelerators.

[3]GPU data could not be accurately reported for this evaluation.

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*, 2015.
[2] R. Andri *et al.*, "Yodann: An ultra-low power convolutional neural network accelerator based on binary weights," in *ISVLSI*. IEEE, 2016, pp. 236–241.
[3] S. Han *et al.*, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *ICLR'16*, 2015.
[4] S. Zhou *et al.*, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint:1606.06160*, 2016.
[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] C. Tai *et al.*, "Convolutional neural networks with low-rank regularization," *arXiv preprint arXiv:1511.06067*, 2015.
[7] 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.
[8] S. Li *et al.*, "Drisa: A dram-based reconfigurable in-situ accelerator," in *Micro*. ACM, 2017, pp. 288–301.
[9] S. Angizi *et al.*, "Imce: energy-efficient bit-wise in-memory convolution engine for deep neural network," in *Proceedings of the 23rd ASP-DAC*. IEEE Press, 2018, pp. 111–116.
[10] S. Li, C. Xu *et al.*, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *DAC*. IEEE, 2016.
[11] S. Aga *et al.*, "Compute caches," in *High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on*. IEEE, 2017, pp. 481–492.
[12] B. C. Lee *et al.*, "Architecting phase change memory as a scalable dram alternative," in *ACM SIGARCH Computer Architecture News*, vol. 37. ACM, 2009.
[13] X. Fong *et al.*, "Spin-transfer torque devices for logic and memory: Prospects and perspectives," *IEEE TCAD*, vol. 35, 2016.
[14] S.-W. Chung *et al.*, "4gbit density stt-mram using perpendicular mtj realized with compact cell structure," in *IEDM*. IEEE, 2016.
[15] L. Sifre and S. Mallat, "Rigid-motion scattering for image classification," Ph.D. dissertation, Citeseer, 2014.
[16] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint:1704.04861*, 2017.
[17] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *arXiv preprint:1610.02357*, 2016.
[18] C. Matthieu et al., "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or-1," *arXiv:1602.02830*, 2016.
[19] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE CVPR*, 2016, pp. 770–778.
[20] Y. Bengio *et al.*, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv:1308.3432*, 2013.
[21] C.-F. Pai *et al.*, "Spin transfer torque devices utilizing the giant spin hall effect of tungsten," *Applied Physics Letters*, 2012.
[22] S. Angizi *et al.*, "Rimpa: A new reconfigurable dual-mode in-memory processing architecture with spin hall effect-driven domain wall motion device," in *ISVLSI*. IEEE, 2017, pp. 45–50.
[23] 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.
[24] Y. Netzer *et al.*, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop*, vol. 2011, 2011, p. 5.
[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. Fong, S. K. Gupta *et al.*, "Knack: A hybrid spin-charge mixed-mode simulator for evaluating different genres of spin-transfer torque mram bit-cells," in *SISPAD*. IEEE, 2011, pp. 51–54.
[27] (2011) Ncsu eda freepdk45. [Online]. Available: http://www.eda.ncsu.edu/wiki/FreePDK45:Contents
[28] 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.
[29] S. D. C. P. V. . Synopsys, Inc.
[30] K. Chen *et al.*, "Cacti-3dd: Architecture-level modeling for 3d die-stacked dram main memory," in *DATE, 2012*. IEEE, 2012, pp. 33–38.