A Flexible Processing-in-Memory Accelerator for Dynamic Channel-Adaptive Deep Neural Networks

Li Yang
Arizona State University
Tempe, AZ 85281
lyang166@asu.edu

Shaahin Angizi
University of Central Florida
Orlando, FL 32816
angizi@knights.ucf.edu

Deliang Fan
Arizona State University
Tempe, AZ 85281
dfan@asu.edu

Abstract— With the success of deep neural networks (DNN), many recent works have been focusing on developing hardware accelerator for power and resource-limited embedded system via model compression techniques, such as quantization, pruning, low-rank approximation, etc. However, almost all existing DNN structure is fixed after deployment, which lacks runtime adaptive DNN structure to adapt to its dynamic hardware resource, power budget, throughput requirement, as well as dynamic workload. Correspondingly, there is no runtime adaptive hardware platform to support dynamic DNN structure. To address this problem, we first propose a dynamic channel-adaptive deep neural network (CA-DNN) which can adjust the involved convolution channel (i.e. model size, computing load) at run-time (i.e. at inference stage without retraining) to dynamically trade off between power, speed, computing load and accuracy. Further, we utilize knowledge distillation method to optimize the model and quantize the model to 8-bits and 16-bits, respectively, for hardware friendly mapping. We test the proposed model on CIFAR-10 and ImageNet dataset by using ResNet. Comparing with the same model size of individual model, our CA-DNN achieves better accuracy. Moreover, as far as we know, we are the first to propose a Processing-in-Memory accelerator for such adaptive neural networks structure based on Spin Orbit Torque Magnetic Random Access Memory(SOT-MRAM) computational adaptive sub-arrays. Then, we comprehensively analyze the trade-off of the model with different channel-width between the accuracy and the hardware parameters, eg., energy, memory, and area overhead.

I. Introduction

Nowadays, Deep Neural Networks (DNNs), as the most popular deep learning algorithm, evolve to deeper layers, larger model size and denser connection. For instance, VGG-16[1] has 522MB of parameters and 30.8 GFLOP per image. ResNet[2] can obtain more than 100 layers. However, such DNNs are difficult to be deployed into a power and resource-limited system. To solve this problem, many recent works have been proposed to compress large DNNs, including network quantization[3], low-rank approximation[4], weight non-structured/structured pruning[5], [6], knowledge distillation[7], etc. However, the above methods mainly have two disadvantages: first, the model after compression is fixed at runtime, which is not flexible for a dynamic environment, like dynamic computing resource allocation, low/high power mode, throughput requirement, and dynamic workloads. Second, With different computing resources or application environment changing, the model has to be retrained using different compression ratios and, then reloaded to the target hardware, which involves very high cost.

From DNN hardawre accelerator design domain, the traditional isolated memory and computing units (GPU or CPU) 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[8]. To address these concerns, Processing-in-Memory (PIM) CNN accelerators, as a potentially viable way to address memory wall challenge, have been widely explored[8], [9], [10]. 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.

To address above discussed problems, in this work, we first propose a dynamic channel-adaptive DNN (CA-DNN) structure, which can adjust the involved convolution channel (i.e. model size, computing load) at runtime (i.e. at inference stage without retraining) to dynamically trade off between computing complexity (thus power, speed) and accuracy. Correspondingly, We further propose a PIM accelerator for the proposed adaptive CA-DNN structure based on SOT-MRAM computational adaptive sub-arrays. We summarize our main contributions as follows:

1) We propose a new training strategy for a new dynamic CA-DNN through knowledge distillation by treating the proposed model as an ensemble network consisting of multiple smaller sub-nets with different channel-width. Experiments on ImageNet and CIFAR-10 dataset show that our CA-DNN achieves either similar or better accuracy comparing with individual model and other recent works with the same model size. We also analyze the effectiveness of the knowledge distillation by ablation study on different teacher model sizes. Further, we quantize both the weights and activations to 8-bits for hardware friendly mapping.

2) We develop a SOT-MRAM based PIM accelerator for the proposed CA-DNN. To maximize computation in parallel, each kernel $f_{in}(kh \times kw \times C)$ is initially allocated to one single computational sub-array, which can dynamically select channel-width according to current configuration. Then, we comprehensively analyze the trade-off of the model with different channel-widths between the accuracy and the hardware parameters, eg., energy, memory, and area overhead.

978-1-7281-4123-7/20/$31.00

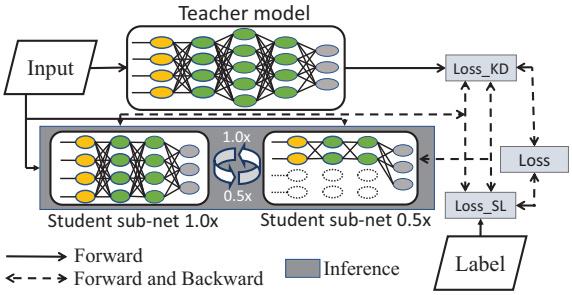


Figure 1: Proposed CA-DNN architecture.

II. Dynamic Channel-Adaptive DNN

A. Knowledge Distillation

Knowledge Distillation (KD) is one model compression method, which is generalized by [7]. The idea behind it is to train a small model (student) by mimicking a pre-trained, larger model or an ensemble model (teacher). In details, the student model is not only trained by minimizing the standard loss (SL) in which the target is the ground truth label, but also by minimizing the loss in which the target is the output of the teacher model (soft target). The standard loss function is:

$$\mathcal{L}_{\text{SL}} = \mathcal{H}(\delta(z_s), y_r) \tag{1}$$

Where $\mathcal{H}$ is the cross-entropy loss, z_s is the output of the teacher model after softmax function, δ represents the softmax function and y_r is the ground truth label. Differently, the loss of knowledge distillation is a KL-divergence loss function, which can be formulated as:

$$\mathcal{L}_{\text{KD}} = \tau^2 * \mathcal{KL}(\delta(z_s/\tau), \delta(z_t/\tau)) \tag{2}$$

Where the τ is a hyper-parameter that used to adjust the distance of each class result, and z_t is the output of the teacher model.

The final loss function is the combination of these two losses, which can be described as:

$$\mathcal{L}_{\text{Student}} = (1-\beta)\mathcal{L}_{SL} + \beta\mathcal{L}_{KD} \tag{3}$$

β is used to control the ratio between these two losses.

B. Channel-Adaptive Deep Neural Network(CA-DNN)

1) Proposed Framework: Unlike standard knowledge distillation framework, the student model in our proposed CA-DNN is an ensemble model, which includes multiple sub-nets by adjusting the channel-width in each convolution layer. We utilize channel-width as a factor for each layer in the student model to select sub-net. The Fig.1 shows an example of CA-DNN with two student sub-nets. Sub-net $1.0\times$ is the full size of the student model and the sub-net $0.5\times$ just keeps half number of feature map channels in each convolution layer, which shares partial weights with the full size student model. Note that, after training, these two models could be dynamically switched without retraining since sub-net $0.5\times$ is a sub-set of Sub-net $1.0\times$, but with different computing load and accuracy.

In addition, inspired by [11], each sub-net in the ensemble student model deploys independent batch-norm layer[12] which solves the feature aggregation inconsistency between different sub-nets. Meanwhile, since these batch-norm parameters is channel-wise, comparing with the convolutional and fully connected layer, the extra parameters and corresponding computation cost are negligible for different sub-nets.

2) Training Method: In terms of training, we aim to optimize the loss of the ensemble student to achieve the adaptive trade-off between accuracy and model size. The loss of ensemble student is defined as the accumulation of the loss of all sub-nets:

$$\mathcal{L}_{\text{ens}} = \sum_{i=1}^{S} \mathcal{L}^i_{subnet} \tag{4}$$

Where S is the number of the sub-nets, and i is the index of sub-nets list, eg, 4 sub-nets list $[1.0\times, 0.75\times, 0.5\times, 0.25\times]$. Algorithm 1 lists the proposed training method. First, we define the teacher model, sub set number, model size of sub-nets list in ensemble student model. In each training iteration, the forward path of teacher model and all sub-nets are executed one time, then loss of each sub-net is computed as formatted in Eq.3. Then, during the backward pass, we accumulate the gradient of all sub-nets which is mathematically same with the Eq.4, and then update weights.

Algorithm 1 CA-DNN training method

Require: Define the numbers and channel width of sub-nets in student model SNet. Given a pretrained teacher model TNet.

```
SNet and TNet initialization
for i ← 1, n_iters do
    Get a batch of data x and label y
    Execute forward of TNet: y_t = TNet(x)
    for sub-net i in SNet do
        Execute sub-net: y_s = SNet(x)
        Compute loss: L_sub-net = (1 − β)L_SL + βL_KD
        Compute and accumulate gradients
    end for
    update weights
end for
```

C. Experiments Results

We test the proposed CA-DNN on CIFAR-10[13] and ImageNet[14]. Further, we analyze the effectiveness of the teacher model and the flexibility of the student model.

1) CIFAR-10: ResNet20[2] is evaluated on CIFAR-10 dataset. In order to show the effectiveness of our proposed method, we compare with the most recent work S-NN[11]. Both networks use the same training configuration and model size. It is also worth to note that the teacher network here is the same ResNet20 model which has 91.2% accuracy. If we choose larger teacher model, better performance could be achieved which is discussed in II-C3. From the Table I, it can be seen that in terms of full precision (FP), our model achieves better accuracy on all the sub-nets. Moreover, to efficiently implement on the Processing-In-MRAM platform we proposed, we further quantize both activation and weights of convolutional layers to 8-bits and 16-bits. The results shown that both quantization have negligible accuracy loss.

Table I: CIFAR-10 results

Network	Width	S-NN	Ours		
		FP	FP	16-bit	8-bit
ResNet20	1.0x	89.8	91.1	91.0	91.1
	0.75x	88.4	90.2	89.8	89.8
	0.5x	85.6	87.5	87.0	86.9
	0.25x	79.5	81.0	80.7	80.6

2) ImageNet: We further examine the proposed CA-DNN on larger ImageNet dataset by using ResNet50 as student model and ResNet101 as teacher model. We also compare our results with S-NN and individual model(I-NN), which is the single network trained independently with the same model size and hyper-parameter configuration. As shown in Table II, we achieve best accuracy on each model size.

Table II: ImageNet results on ResNet50

Network	Width	I-NN	S-NN	Ours	Params(MB)
ResNet50	1.0x	76.1	76.0	76.6	25.5
	0.75x	74.7	74.9	75.4	14.7
	0.5x	72.0	72.1	72.4	6.9
	0.25x	63.8	65.0	65.2	2.0

3) Ablation Study and Analysis: •**Teacher Model Selection:** Given a fixed student ensemble model, we employ different teacher models to explore the effectiveness of knowledge distillation. As shown in Table III, we choose ResNet20 as the student and ResNet20/32/44 as the teacher, respectively, testing on CIFAR-10 dataset. The experiment shows that student ensemble model has better performance with larger teacher model.

Table III: Teacher model selection on CIFAR-10

Student	Width	Teacher		
		ResNet20	ResNet32	ResNet44
ResNet20	1.0x	91.1	91.4	91.9
	0.75x	90.2	90.7	91.2
	0.5x	87.5	88.6	88.7
	0.25x	81.0	82.9	82.5

•**Multiple Channel-Width Selection:** To show the flexibility of the proposed CA-DNN, we choose more sub-nets in a single student ensemble model. Table IV lists the ResNet20 which is assembled by 8 sub-nets on CIFAR-10 dataset. The accuracy of each sub-net decreases with smaller model size. In addition, in comparison to the 4 sub-nets case, the corresponding same sub-nets(0.25x, 0.75x, 1.0x) on 8 sub-nets case have almost the same accuracy, which demonstrate that the number of sub-nets in the ensemble model has very small effect on the accuracy of each sub-net.

Table IV: Multiple channel-width selection on CIFAR-10

Network	Channel Width								
	0.25x	0.35x	0.45x	0.5x	0.55x	0.65x	0.75x	0.85	1.0x
ResNet20	81.0	82.3	85.9	-	87.1	88.7	90.1	90.1	91.0
	81.0			87.5			90.2		91.1

III. Bit-wise Processing-in-MRAM Accelerator

Our proposed PIM accelerator architecture is depicted in Fig.2a with computational sub-arrays, kernel and image banks, and a Digital Processing Unit (DPU) with three sub-components as represented by Quantizer, Activation Function, and Batch Normalization. The platform is mainly controlled by Ctrl (located in each sub-array) to run whole DNNs layers. The architecture is inspired by preliminary IMCE [10]. In the first step, with Kernels (W) and Input feature maps (I) that are respectively saved in Kernel and Image Banks, W has to be instantly quantized for mapping into sub-arrays. Moreover, quantized shared kernels will be used for different inputs. This operation is implemented through DPU's Quantizer-Qunt.(shown in Fig.2a) and then outputs are sent to the sub-arrays, developed to handle the computational load employing PIM methods. In the second and third steps, as will be thoroughly explained, the parallel computational sub-arrays along with add-on counter and shifter units perform feature extraction. Eventually, the accelerator's DPU activates the resultant feature map to complete the fourth step by producing output feature map.

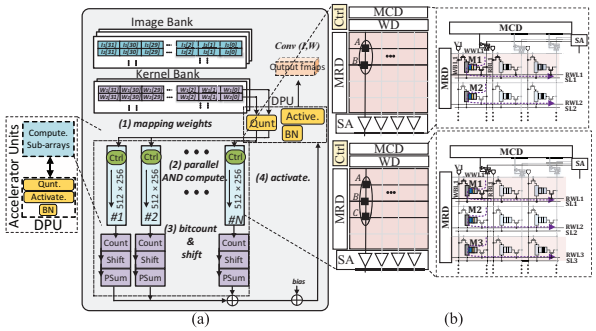


Figure 2: (a) MRAM accelerator platform, (b) Computational sub-array design.

•**Sub-array architecture:** Fig.2b illustrates the in-memory computing sub-array architecture implemented by SOT-MRAM. This sub-array is basically composed of a Memory Row Decoder(MRD), a Memory Column Decoder(MCD), a Write Driver(WD) and n Sense Amplifier(SA) ($n \in$ # of columns) that are adjustable by Ctrl to implement a dual mode computation i.e. memory write/read and in-memory logic operations. SOT-MRAM device is a composite device of a spin Hall metal(SHM) and a Magnetic Tunnel Junction(MTJ). Considering MTJ as the main storage element, the parallel magnetization resistance in both magnetic layers is represented by '0' and is lower than that of anti-parallel magnetization resistance ('1'). Every SOT-MRAM bit-cell in computational sub-arrays is connected to five controlling signals, namely Write Word Line(WWL), Write Bit Line(WBL), Read Word Line(RWL), Read Bit Line(RBL), and Source Line(SL). The computational sub-array is mainly developed to realize the computation between in-memory operands using two different mechanisms as called two-row activation and three-column activation. These mechanisms will be used to respectively implement bulk bit-wise in-memory AND and addition operations.

•**Bit-line computation mode with RSA:** The SOT-MRAM sub-array is designed to realize the bulk bit-wise in-memory logic operations between every two or three operands positioned in the same bit-line. Generally, in the 2-/ 3-input in-memory logic method, every two/three bits stored in the identical column are selected with the MRD [15] and simultaneously sensed by SA connected to the same bit-line. The Reconfigurable Sense Amplifier(RSA), as illustrated in Fig.3, has 3 sub-SAs and totally 5 reference-resistance branches that could be enabled by control bits (C_M, C_{OR3}, C_{MAJ}, C_{AND3}, C_{AND2}) by the Ctrl to implement the memory and

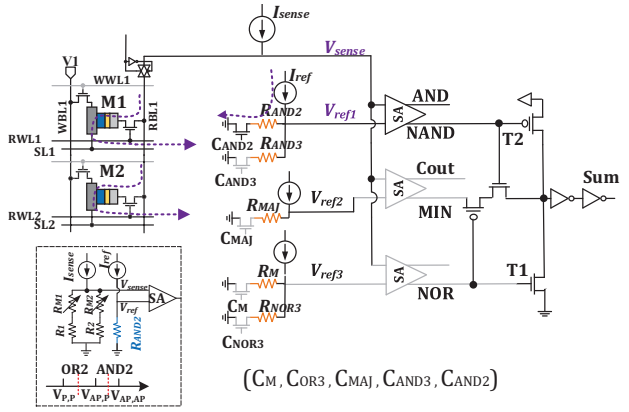


Figure 3: The reconfigurable SA for implementing single cycle AND2 and addition operations.

computation modes. RSA realizes memory read and single-threshold logic operations through activating one control bit in either branches at a time. Additionally, by activating more than two control bits at a time, more than one logic functions could be simultaneously realized with SAs. This could be employed to generate complex multi-threshold logic functions such as XOR3/XNOR3. The RSA works based on the following mechanism: the voltage generated after injecting a small amount of current (reference current) over the selected reference by Ctrl will be compared with the sensed voltage of equivalent resistance of such parallel connected bit cells and their cascaded access transistors after injecting a small amount of current (sense current) over the resistors. Through selecting different reference resistances e.g. R_M and R_{AND2}, the RSA can perform basic memory and 2-input in-memory Boolean AND function, respectively. For example, to realize AND operation, R_{ref} is set at the midpoint of $R_{AP}//R_P$ ('1','0') and $R_{AP}//R_{AP}$ ('1','1'). With the data organization shown in Fig.2b, where A and B operands correspond to M1 and M2 memory cells, respectively, 2-input in-memory outputs AB in only one memory cycle. The idea of voltage comparison between V_{sense} and V_{ref} for implementing AND2 is shown on Fig.3. The computational sub-array can also perform add/sub operation based on RSA in a single cycle. Extensive analysis of effects of process variation has been conducted in preliminary work[10], which will not be shown here due to space limit.

IV. Adaptive Co-Optimized Mapping

A. Intra Sub-array Parallelism

To efficiently exploit the computation resources and maximize in-memory parallelism of the accelerator while running the proposed dynamic CA-DNN, each kernel $f_{in}(kh\times kw\times C)$ is initially allocated to one single computational sub-array as shown in Fig.4a. Therefore, kernel $f_{out}(N)$ will be first mapped to N sub-arrays. For very large kernels, where the kernel size exceeds the sub-array column size, the data organization of inputs and weights in the accelerator can be simply tailored. In this way, number of required sub-array (N_s) can be formulated as $\left\lceil \frac{(N\times R_k)\times n_p}{S_r} \right\rceil$, $R_k = \frac{kh\times kw\times C}{S_c}$. Where R_k is the number of required sub-array's rows per kernel and S_r and S_c are sub-array size (row and column). $n_p \in$ {2,4,8,..} represents the scaling coefficient for making copies of kernels, where its minimum value is limited by 2 enabling computation in one sub-array and its maximum is limited by the maximum number of allocated sub-arrays. After mapping, the sub-arrays can work individually but in parallel to process bit-wise convolution through two consecutive steps delineated in Fig.5, namely *parallel AND computation* and *bitcount & shift*.

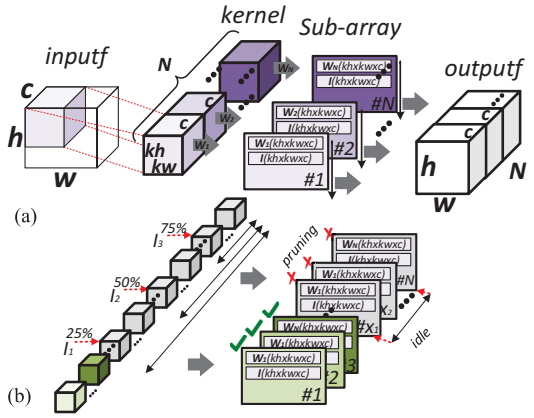


Figure 4: (a) Intra sub-array parallelism and hardware mapping methodology to generate output feature map, (b) Adaptive sub-array pruning for saving in-memory resources. I_i-index can be dynamically set by the controller to exclude a part of sub-arrays from computation.

The key idea behind performing bit-wise convolution is to exploit bulk bit-wise *AND* as a parallelizable operation, *bit-count*, and *bitshift* to accelerate MACs in convolutional layers. The AND-based convolution of k-bit fixed point integers has been presented in [16]. There are some other layers in CNNs, such as inception layer (directly taking image as inputs and not necessarily quantized) and Fully-Connected (FC) layer. These layer can be equivalently implemented by convolution operations using 1×1 kernels [16]. Thus, all layers could be implemented by convolution computation by exploiting these operations [10], [16]:

$$I * W = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} 2^{m+n} bitcount(AND2(C_n(W), C_m(I))) \quad (5)$$

Assuming 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. 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 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 $W = \sum_{n=0}^{N-1} 2^n c_n(W)$. As shown in data mapping step in Fig.5, $C_2(W)$-$C_0(W)$ are consequently mapped to the designated sub-arrays. Accordingly, $C_2(I) - C_0(I)$ are

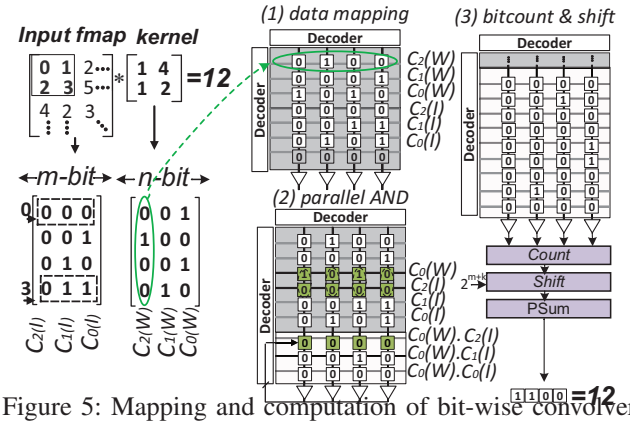


Figure 5: Mapping and computation of bit-wise convolver.

mapped in the following memory rows in the same way. Now, computational sub-array can perform bit-wise parallel AND2 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. Bitcount is then implemented using Count unit and passes the data to a Shifter implemented by consecutive memory read and write operations (FRC). As depicted in Fig.5, "0001", produced by in-memory adder is left-shifted by 3-bit ($\times 2^{2+1}$) to "1000". Eventually, in-memory bit-wise adder can produce the output fmaps. Note that the bit-wise convolver supports different configurations of weight and activation (<W:A>=< n:m >).

B. Adaptive and Recoverable Sub-array Pruning

In this subsection, we propose an adaptive and recoverable sub-array pruning mechanism based on our proposed channel-adaptive DNN structure to dynamically trade-off between power, speed, latency, model size and accuracy of the accelerator. After hardware mapping of convolution kernels to the computational sub-arrays based on mechanism explained in previous subsection, the controller needs to internally store only three indexes for the memory address corresponding to the first pruned channel in different model sizes (i.e. 0.25x, 0.50x, and 0.75x.). Leveraging this mechanism, the accelerator could be dynamically adjusted to exclude the pruned kernels (sub-arrays) from computation according to the predefined constraints. In this way, the accelerator can save energy and achieve speed-up as compared with inefficient direct mapping approaches typically used in different PIM platforms. Fig.4b shows how such adaptive feature is applied in the computational sub-arrays with an index-I_i. For example, the platform can readily prune 25% of the computational sub-arrays considering index-I_1.

V. Evaluation Results

A. Platform Setup

We configure our in-memory accelerator with a 512Mb total capacity and 256×512 memory sub-array organized in a H-tree routing manner. For the simulations, we developed an extensive bottom-up evaluation framework. For the device simulations, NEGF and LLG with spin Hall effect equations were taken to model SOT-MRAM bit-cell [17], [10]. At circuit level, we develop a Verilog-A model for 2T1R bit-cell, which can be used along with interface CMOS circuits in Cadence Spectre. We used 45nm NCSU PDK library [18] to assess the presented circuit designs and obtain the performance metrics. At architectural level, based on the device-circuit results, we first extensively modified NVSim [19] by developing specific PIM library. The simulator can change the configuration files of NVSim (.cfg) according to the model size and various memory array organization. Then, we develop a behavioral simulator in Matlab that assess the energy and latency parameters that the accelerator consumes to run CA-DNNs. Besides, we integrated our mapping optimization algorithm to increase the throughput w.r.t. the available resources. Here, we use ResNet20 with 8-bit quantized convolutional layers to test on CIFAR-10 dataset.

B. Energy Consumption Estimation

The bulk bit-wise AND and its write-back operations are considered as crucial sources of energy consumption in our MRAM accelerator. Therefore, we first report the number of required AND Ops for processing input fmaps. Fig. 6a illustrates the break-down of AND Ops in the platform in different convolutional layers. Correspondingly, the latency is also proportional to the computing requirement as discussed here. Fig.6b reports the energy consumption of ResNet20's convolutional layers divided into four different regions AND-$compute$ AND-WB, $bitcount$-$shift$, and add operations under different model sizes. Note that the energy consumption of Shifter and Bit-Counter are plotted together in Fig.6b. Here, we can observe how adaptive sub-array pruning technique reduces the number of operations and accordingly energy consumption by cutting off the idle sub-arrays from computation as explained earlier. We observe that, 0.25x model achieves ∼2.7× reduction in energy consumption compared with 1.0x model sacrificing the accuracy as explained later.

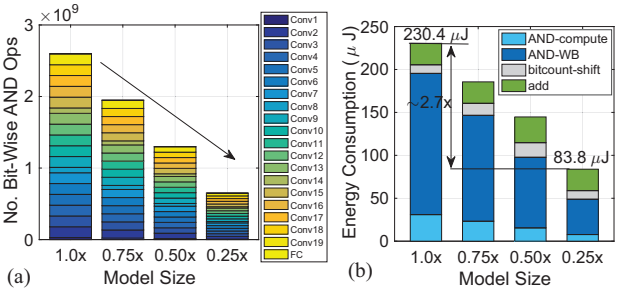


Figure 6: (a) Break-down of no. AND operations in convolutional layers of ResNet20 w.r.t. model size, (b) Energy distribution under different configurations.

C. Memory Storage Requirement

Fig.7 shows breakdown of the accelerator's memory storage to run full-precision and 8-bit quantized ResNet20 model for CIFAR-10 under different model sizes. It can be observed that by adjusting the channel-width, the required memory storage is different. Besides, Fig.7 shows the memory storage and accuracy trade-off of our PIM platform. Based on this, 8:0.50x network achieves 10.6× reduction in memory storage compared with the 32-bit network while sacrificing the accuracy by 4.2%.

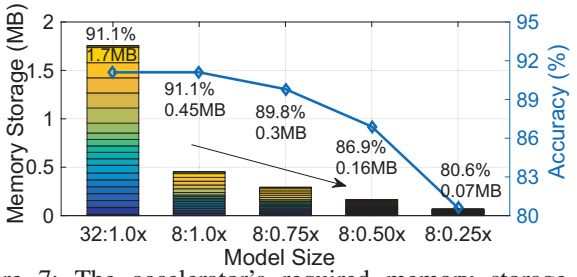
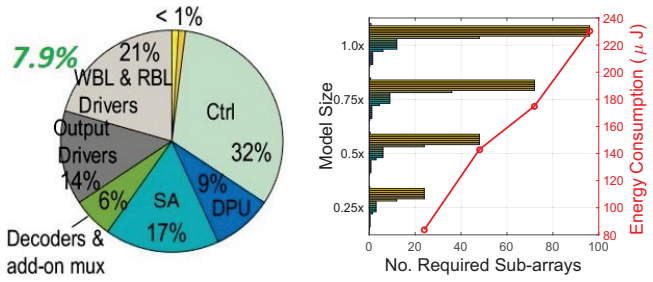


Figure 7: The accelerator's required memory storage and accuracy trade-off for different model sizes. (8:0.75x denotes 8-bit quantized ResNet20 with 0.75x model size.)

D. Area Overhead

To estimate the area overhead of MRAM platform, three main hardware cost sources must be taken into consideration. First, add-on transistors to SAs connected to each BL. Second, the modified MRD overhead; we modify each WL driver by adding two more transistors in the typical buffer chain. Third, the Ctrl's overhead to control enable bits; ctrl generates the activation bits with MUX units with 6 transistors. We considered the area of shifter and counter units as a part of Ctrl area. Overall, the presented processing-in-MRAM platform imposes 7.9% overhead to main memory die. Fig.8a reports such area overhead breakdown. Moreover, Fig.8b profiles the area distribution of different convolutional layers of the ResNet20 based on number of required computational sub-arrays for processing a single image with various model sizes. It also reports the energy consumed by each model size while taking different number of sub-arrays. Based on this, we can observe that how adaptive sub-array pruning mechanism can save energy by dynamically cutting down the number of sub-arrays.



(a) (b)

Figure 8: (a) Breakdown of area overhead of the accelerator, (b) Trade-off between area and energy consumption for different model sizes.

VI. Conclusion

In this work, we first propose a CA-DNN which can dynamicly adjust model size at runtime without retraining. Experiments on CIFAR-10 and ImageNet both validate the effectiveness of CA-DNN. Then we develop a PIM accelerator for the CA-DNN and validate it on CIFAR-10 dataset by using ResNet20 model. We further comprehensively analyze the trade-off between the accuracy and the hardware parameters, eg., energy, memory, and area overhead.

Acknowledgments

This work is supported in part by the National Science Foundation under Grant No.1740126, No.1908495, No.1931871 and Semiconductor Research Corporation nCORE

References

[1] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[2] K. He *et al.*, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016, pp. 770–778.
[3] I. Hubara *et al.*, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.
[4] E. L. Denton *et al.*, "Exploiting linear structure within convolutional networks for efficient evaluation," in *NIPS*, 2014, pp. 1269–1277.
[5] S. Han *et al.*, "Learning both weights and connections for efficient neural network," in *NIPS*, 2015, pp. 1135–1143.
[6] W. Wen *et al.*, "Learning structured sparsity in deep neural networks," in *NIPS*, 2016, pp. 2074–2082.
[7] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.
[8] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3. IEEE Press, 2016, pp. 27–39.
[9] S. Li *et al.*, "Drisa: A dram-based reconfigurable in-situ accelerator," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2017, pp. 288–301.
[10] S. Angizi *et al.*, "Imce: energy-efficient bit-wise in-memory convolution engine for deep neural network," in *23rd ASP-DAC*. IEEE Press, 2018, pp. 111–116.
[11] J. Yu *et al.*, "Slimmable neural networks," *arXiv preprint arXiv:1812.08928*, 2018.
[12] S. Ioffe *et al.*, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015, pp. 448–456.
[13] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.
[14] J. Deng *et al.*, "Imagenet: A large-scale hierarchical image database," in *CVPR*. IEEE, 2009, pp. 248–255.
[15] 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.
[16] S. Zhou *et al.*, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.
[17] X. Fong *et al.*, "Spin-transfer torque devices for logic and memory: Prospects and perspectives," *IEEE TCAD*, vol. 35, 2016.
[18] (2011) Ncsu eda freepdk45. [Online]. Available: http://www.eda.ncsu.edu/wiki/FreePDK45:Contents
[19] 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.