Processing-in-Memory Accelerator for Dynamic Neural Network with Run-Time Tuning of Accuracy, Power and Latency

Li Yang, Zhezhi He, Shaahin Angizi and Deliang Fan
School of Electrical, Computer and Energy Engineering, Arizona State University, Tempe, Arizona, USA
{lyang166, zhezhihe, sangizi, dfan}@asu.edu

2020 IEEE 33rd International System-on-Chip Conference (SOCC) | 978-1-7281-8746-4/20/$31.00 ©2020 IEEE | DOI: 10.1109/SOCC49529.2020.9524770

***Abstract*—With the widely deployment of powerful deep neural network (DNN) into smart, but resource limited IoT devices, many prior works have been proposed to compress DNN in a hardware-aware manner to reduce the computing complexity, while maintaining accuracy, such as weight quantization, pruning, convolution decomposition, etc. However, in typical DNN compression methods, a smaller, but fixed, network structure is generated from a relative large background model for resource limited hardware accelerator deployment. However, such optimization lacks the ability to tune its structure on-the-fly to best fit for a dynamic computing hardware resource allocation and workloads. In this paper, we mainly review two of our prior works [1], [2] to address this issue, discussing how to construct a dynamic DNN structure through either uniform or non-uniform channel selection based sub-network sampling. The constructed dynamic DNN could tune its computing path to involve different number of channels, thus providing the ability to trade-off between speed, power and accuracy on-the-fly after model deployment. Correspondingly, an emerging Spin-Orbit Torque Magnetic Random-Access-Memory (SOT-MRAM) based Processing-In-Memory (PIM) accelerator will also be discussed for such dynamic neural network structure.**

***Index Terms*—Processing-in-Memory, Dynamic neural network**

I. Introduction

Recently, Artificial Intelligence, especially Deep Neural Network (DNN), becomes the main stream intelligent data processing technology due to its remarkable performance in various domains, like image/video pattern recognition, nature language processing, etc. [3]. However, to improve DNN performance, it evolves into deeper, denser, wider network architecture with rapidly growing network size and computing complexity, which brings great challenge to deploy such powerful technique into power-, size-, resource-limited computing hardware, like IoT, embedded system, mobile phone, etc. To address such issue, many prior works have been proposed to compress DNN in a hardware-aware manner to reduce the computing complexity, while maintaining accuracy, such as weight quantization [4], [5], pruning [6], [7], convolution decomposition [8], [9], etc.

In general, such model compression techniques need to optimize for each individual target computing platform due to their unique computing resources. Moreover, for one specific computing hardware platform, after DNN model optimization and compression, it generated a smaller, but fixed, network structure from a relative large background model. However, the real-word is dynamic, meaning that the computing hardware platform is dynamic (e.g. high performance mode or low battery mode, dynamic computing resource allocation, etc.), as well as the working environment is dynamic (e.g. varying workloads from streaming input data due to dynamic sensing efforts or communication channels). Thus, it urges the need to create a new dynamic DNN structure that could equip with the ability to tune its structure on-the-fly to best fit for a dynamic computing hardware resource allocation and workloads.

To address above challenge, in this paper, we mainly review our two previous research works published in ASPDAC'20 and DAC'20 [1], [2] to summarize the construction of dynamic neural network. [1] samples the multiple *uniform* sub-nets from a large background super-net by adjusting the channel-width ratio, which is fixed among all layers, as well as leveraging knowledge distillation to improve accuracy of each sub-net. Furthermore, [2] presents a *non-uniform* sub-net selection method, inspired by an important observation that different DNN layers may have different sensitivities to capacity reduction. This observation has bee reported in many prior works in model pruning [10]–[12] or Network Architecture Search (NAS) [13]–[16]. In general, the overflow of dynamic neural network can be divided into two successive stages: *Sub-nets generation* and *Fused sub-nets training* [1], [2], [17]. In the first step, multiple sub-nets are sampled from a background model (aka. super-net). Two different sub-nets sampling methods are discussed corresponding to uniform and non-uniform sub-nets respectively. Then, the sampled sub-nets are fused into a cross entropy loss function for multi-objective optimizations to construct a dynamic neural network.

Moreover, in terms of DNN hardware accelerator design, the traditional Von-Neumann computing architecture faces large challenges in such data-/compute-intensive applications, 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 [1], [18]–[20]. To address these challenges, accelerating DNN in a Processing-in-Memory (PIM) platform is widely explored nowadays [1],

This work is supported in part by the National Science Foundation under Grant No.2005209, No.2003749, No.1931871 and Semiconductor Research Corporation nCORE

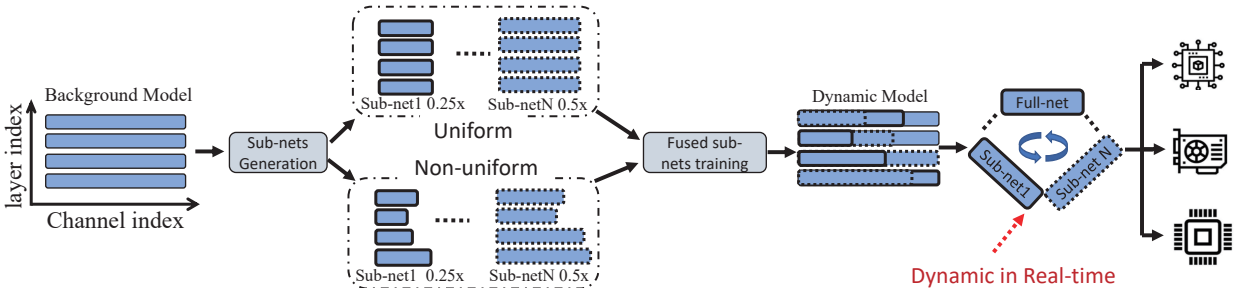


Figure 1: Overview of two-stage dynamic neural network framework for both uniform and non-uniform sub-nets sampling. In the first stage, #N sub-nets are sampled from the background model with difference sizes by a sub-nets generator. Note that, the weights of these sub-nets are **partially shared** according to the same weight channels utilization. Then in the followed second step, fused sub-nets training helps to construct the full-size *dynamic model*. In the end, these #N sub-nets can do inference individually to achieve the complete trade-off among speed, power consumption and accuracy [1], [2].

[18], [21], [22]. The key idea of utilizing PIM to accelerate DNN is to leverage the logic-in-memory design to directly implement the required computation of DNN within memory, without relocating the large DNN model parameters out of memory chip, leading to greatly reduced computing energy and latency. From device technology perspective, there are many recent works carried out by the emerging Non-Volatile Memories (NVM) that are promising to design such PIM concept, for example, Resistive Random-Access Memory (RRAM), Phase Change Memory (PCM), Spin-Transfer Torque Magnetic Random-Access Memory (STT-MRAM). Comparing with STT-MRAM, which is the leading non-volatile memory candidate, Spin-Orbit Torque Magnetic Random-Access-Memory (SOT-MRAM) not only improves the write/read stability but also requires a much smaller writing current due to strong spin-orbit coupling [23]. In this review paper, the SOT-MRAM based PIM accelerator will be discussed for the uniform dynamic neural network [1].

II. Dynamic Neural Network

In this section, we first review the dynamic neural network framework leveraging both uniform and non-uniform sampling methods and then summarize the main experiment results. As illustrated in 1, the general overflow of constructing dynamic neural network consists of two stages:**1) Sub-nets generation:** various sub-nets are sampled from the background model with different sizes. Two sub-nets generation methods will be discussed for uniform and non-uniform sub-nets respectively; **2) Fused sub-nets training:** the sub-nets sampled form stage 1 will be constructed to a dynamic inference model, where the overlapped weights are partially shared. To optimize the dynamic model, these sub-nets are fused into a cross entropy loss with multi-objectives optimization [17]. By doing so, the dynamic model could switch between different sub-nets for run-time trade-off between speed, power consumption and accuracy.

A. Problem definition

Suppose that we have a neural network $f(\boldsymbol{x}; \mathbf{W})$ with L layers, where $\boldsymbol{x}$ is the input data and $\mathbf{W}_l$ is the weight parameters with the size of $\mathbb{R}^{p\times q\times kh\times kw}$. To perform the sub-nets sampling, we also denote the sub-net-ith weight binary mask as $\mathbf{M}_{l,i} \in \{0,1\}^{p\times q\times 1\times 1}$. Thus, the overflow as shown in Fig.1 can be formatted as:

$$\overbrace{\min_{\{\mathbf{W}_l\}_{l=1}^{L}} \sum_{i=1}^{N} \mathcal{L}_i\Big(f(\boldsymbol{x}; \{\mathbf{W}_l \cdot \mathbf{M}_{l,i}\}_{l=1}^{L}), \boldsymbol{y}\Big)}^{\text{Stage 1}}$$
$$\text{s.t.} \underbrace{R(\mathbf{M}_{0,i}) = R(\mathbf{M}_{1,i}) = ... = R(\mathbf{M}_{L,i})}_{\text{Stage 2: uniform}} \tag{1}$$
$$\underbrace{\{\mathbf{M}_{l,i}\}_{l=1}^{L} = \arg\min \mathcal{L}^*(f(\boldsymbol{x}; \{\mathbf{W}_l \cdot \mathbf{M}_{l,i}\}), \boldsymbol{y}, \lambda_i, \beta_i)}_{\text{Stage 2: non-uniform}}$$

where N is the total number of sub-nets, and $i \in \{1, ..., N\}$ is the index of sub-nets. Note that, the weights are partially shared for all sub-nets. For the uniform sub-nets generation, R represents the channel-width ratio which is a fixed constant among all layers. For the non-uniform alternative, we have two hyper-parameters λ_i and β_i, which are utilized to adjust model capacity during sub-nets sampling. $\mathcal{L}_i$ is the general loss function(i.e. cross-entropy) for sub-net-i, and $\mathcal{L}^*$ is the loss with a weight penalty term to perform sampling. We will discuss each stage in the two subsections below.

B. Sub-nets generation

1) Uniform sub-nets generation method: The channel-width ratio is utilized as a factor for each layer to select sub-net. All layers share the identical channel-width ratio (e.g., $0.25\times$, $0.5\times$) which is manually configured. For example, the sub-net $0.5\times$ sequentially selects half number of weight output channels for all layers except the last one [1], [17].

2) Non-uniform sub-nets generation method: Different from the uniform sub-nets generation, which simply selects sub-nets by hand, the non-uniform dynamic neural network utilizes optimized group Lasso-based regularization method, named clipped Lasso [2], [7], to learn the sub-nets structures as depicted in Eq.1.

Group Lasso-based regularization is a general technique for structured pruning [12], which is a weight penalty term that

consistently acts on all the weights. Different from that, the clipped group Lasso only works on the "important" weights (larger magnitude) by utilizing an adaptive Weight Penalty Clipping (WPC) as the countermeasure [2], which is:

$$\mathcal{L}* = \mathcal{L}(f(\boldsymbol{x}; \{\mathbf{W}_l\}_{l=1}^{L}), \boldsymbol{y}) + \lambda \sum_{l=1}^{L} \sum_{i=1}^{G_l} \underbrace{\min\left(||\mathbf{W}_{l,i}||_2; \tau_l\right)}_{\text{WPC}}$$
$$\text{s.t.} \quad \tau_l = \beta \cdot \frac{1}{G_l} \sum_{i=1}^{G_l} ||\mathbf{W}_{l,i}||_2 \tag{2}$$

Where λ is a scaling factor to adjust the weight penalty strength. Through choosing different λ_i, multiple non-uniform sub-nets with different model sizes can be generated [7]. Moreover, to avoid the weight penalty unnecessarily acts on all weights, the adaptive threshold τ_l is used to clip the penalty on weights with larger L_2-norm $||\boldsymbol{W}_{l,i}||_2$. The clipping threshold τ_l is calculated by mean of $||\boldsymbol{W}_{l,i}||_2$ with a scaling hyper-parameter β. Thus, during the training processing, the weight penalty will be only added on the "important" weights, which has larger L_2-norm $||\boldsymbol{W}_{l,i}||_2$ comparing with the threshold τ_l. Otherwise, the clipped group Lasso term will be treated as a constant, where has no effect on backward propagation of the corresponding weights.

C. Fused sub-nets training

After sampling multiple sub-nets from stage one, the fused sub-net training is leveraged to fuse those sampled sub-nets into a cross entropy loss function for multiple objective optimizations (Eq.1). Fig. 1 shows the training processing. Similar with multiple individual networks optimization, all sub-nets go through the forward and backward to compute their gradients. Then differently, these gradients are gathered to update the weights once. In spite of the fact that the computation complexity is increased in comparison with a single network training scheme, the dynamic model framework reduces the developer-costs of training these sub-nets individually. Furthermore, it achieves the complete trade-off among speed, power and accuracy in run-time for a trained inference model. So, utilizing dynamic neural network, users can adjust the model to the suitable sub-net according the current hardware and environment requirements in real-time. In addition, to avoid the effect on shared batch-norm layers, we follow [17] to use independent batch-norm layers for each sub-net.

The fused training scheme are slightly different between uniform and non-uniform counterparts as shown in Fig. 2. For uniform dynamic neural network training, knowledge distillation [1], [24] is utilized to improve accuracy while sacrificing the memory and computation cost. The main idea behind it is to train sub-nets by mimicking a pretrained larger model (teacher net as shown in Fig. 2(a)). In practice, the sub-nets are not only learned by the general cross entropy loss with data labels, but also by distilling knowledge from the teacher network whose output is treated as soft label. On the other hand, non-uniform fused training just computes a cross entropy loss for all sampled sub-nets as shown in Fig. 2(b) [2].

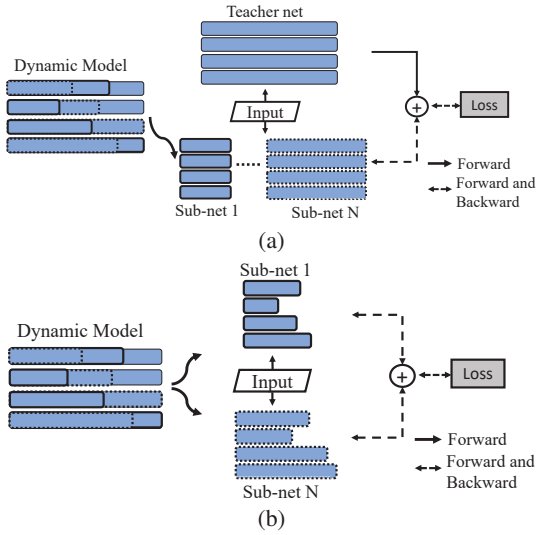


Figure 2: Fused sub-nets training pipeline for (a) uniform [1] and (b) non-uniform sub-nets [2].

D. Dynamic neural network evaluation

We perform the experiments for both uniform [1] and non-uniform dynamic neural network [2] on CIFAR-10 [25] and ImageNet [26] dataset using ResNet [27] architecture. Extensive experiments analysis and ablation study have been conducted in our prior works [1], [2]. Here we only show important experimental results.

1) Uniform dynamic neural network: ResNet20 [27] is tested on CIFAR-10 dataset. It compares with Slimmable Neural Network (S-NN) [17], which also presents channel-width based uniform dynamic neural network but without utilizing knowledge distillation. Both methods have the same configurations and are compared under the same model capacity. Table I shows the accuracy results. To deploy the dynamic model to our PIM platform later, instead of using full precision (FP) model, both weight and activation are further quantized into 8-bit and 16-bit with little accuracy degradation.

Network	Width	S-NN [17]	Ours [1]		
		FP	FP	16-bit	8-bit
ResNet20	1.0×	89.8	91.1	91.0	91.1
	0.75×	88.4	90.2	89.8	89.8
	0.5×	85.6	87.5	87.0	86.9
	0.25×	79.5	81.0	80.7	80.6

Table I: Inference accuracy (%) of ResNet20 on Cifar10 for uniform dynamic neural network [1].

We further study the performance with ResNet50 on ImageNet dataset. To prove the non-uniform sampling method, [2] also does comparison with the single network which are trained independently under the same sizes. As depicted in Table II, the best accuracy is achieved on each model size.

Network	Width	I-NN	S-NN [17]	Ours [1]	Params(MB)
ResNet50	1.0×	76.1	76.0	76.6	25.5
	0.75×	74.7	74.9	75.4	14.7
	0.5×	72.0	72.1	72.4	6.9
	0.25×	63.8	65.0	65.2	2.0

Table II: Inference accuracy (%) of ResNet50 on ImageNet for uniform dynamic neural network [2].

2) Non-uniform dynamic neural network: Similar to the experiments of uniform dynamic neural network, four non-uniform sub-nets are sampled. The experiment results are shown in Table. III. Non-uniform method achieves almost the same or better accuracy in both individual and dynamic networks, with even smaller number of parameters (10^6) and FLOPS (10^8) of each sub-net, .

Sub-nets		subnet1	subnet2	subnet3	subnet4
S-NN [17]	Parameters	0.83	3.05	6.68	11.68
	FLOPS	1.35	4.83	10.4	18.14
	Individual	49.9	61.1	66.7	69.7
	Dynamic	48.7	60.9	66.6	69.4
Ours [2]	Parameters	0.66	2.73	5.14	12.2
	FLOPS	0.89	3.97	7.17	19.8
	Individual	50.1	62.6	66.9	71.4
	Dynamic	48.4	61.8	66.8	69.8

Table III: Inference accuracy (%) of ResNet18 on ImageNet for non-uniform dynamic neural network [2].

III. Bit-wise PIM Accelerator

In this section, we present a Processing-in-Memory (PIM) accelerator architecture for uniform dynamic neural network [1]. The presented platform, as depicted in Fig 3a, consists of image and kernel banks, computational memory sub-arrays, and a Digital Processing Unit (DPU). DPU is essentially composed of three components indicated by Active. (Activation) Function, Quantizer (Qunt.), and Batch Normalization (BN). A controller unit in every sub-array governs and synchronize the platform to execute various DNNs layers. The presented design is inspired by our preliminary works in PIM domain mainly IMCE [22], CMP-PIM [19], and GraphS [28]. Prior to the computation, Input feature maps (denoted by I) and Kernels (W) are required to be stored in Image and Kernel Banks as shown in Fig 3a. During the first computation stage (1), W, kernels can be instantly quantized by DPU's Quantizer and then mapped into computational sub-arrays. The PIM-enhanced sub-arrays are especially designed to take the computational load and process it in parallel. During the (2) and (3) stages, the sub-arrays with add-on shifter/counter units execute feature extraction based on method explained accordingly. Eventually, the DPU activates the feature maps and generates the output (stage (4) in Fig 3a).

•**Sub-array architecture.** Fig. 3b illustrates the processing in memory sub-array architecture implemented by SOT-MRAM. To realize a dual mode computation supporting both memory

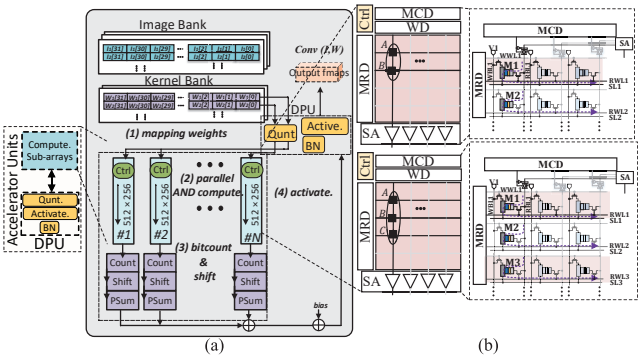


Figure 3: (a) MRAM accelerator platform, (b) Computational sub-array design [1].

read/write and PIM operations, the sub-array is composed of a Memory Row Decoder (MRD), a Memory Column Decoder (MCD), a Write Driver (WD) and y configurable Sense Amplifier (SA) ($y \in$ # of columns) by controller. SOT-MRAM device is a composite device of a Magnetic Tunnel Junction (MTJ) and a spin Hall metal (SHM). With MTJ as the storage element, the parallel magnetization resistance in both magnetic layers is shown by '0' and is smaller than that of anti-parallel magnetization resistance ('1'). As shown in Fig. 3b, a SOT-MRAM bit-cell is controlled by 5 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 implements bulk bit-wise in-memory AND and addition operations between in-memory operands with 2-row activation and 3-row activation to respectively.

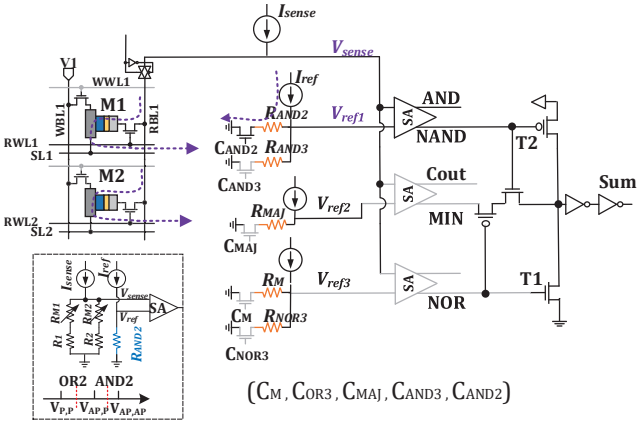


Figure 4: The configurable SA to realize single cycle AND2 and addition operations, adapted from [1].

•**Bit-line computing:** In the 2-/ 3-row activation PIM mechanism, MRD [29] activates two/three bits saved in the similar column. Then the SA can simultaneously sense the connected SOT-MRAM cells resistance within every bit-line. As shown

in Fig 4, the configurable Sense Amplifier (CSA), consists of three sub-SAs and 5 reference resistors. The integrated references are enabled by the controller through enable bits (C_{M}, C_{OR3}, C_{MAJ}, C_{AND3}, C_{AND2}). The CSA is designed to implement read operation and one-threshold logic operations such as `(N)AND2/(N)OR2/(N)AND3/(N)OR3/` by enabling one control bit at a time. Besides, the CSA could implement up to three logic operations at a same time realized with its SAs. Such scheme is used to produce 2-threshold logic operations such as `XOR3/XNOR3`. As shown in Fig 4, the CSA works as follow: by injecting a very small current I_{sense} over the selected cells in the memory, a sense voltage V_{sense} is generated on the RBL that goes to the first input of 3 sub-SAs as voltage comparator. At the same time, a reference current (I_{ref}) is injected to reference resistance branches to generate a V_{ref} at the second input of sub-SAs. Depending on the enable bit configuration up to three references could be selected. The CSA's voltage comparison idea between $\mathrm{V_{sense}}$ and $\mathrm{V_{ref}}$ to realize `(N)AND2/(N)OR2` is shown on Fig 4. For example, through selecting R_{M} and R_{AND2}, the CSA respectively executes memory and 2-input AND function. We'll refer the readership to Ref. [1] to get more information about reference tuning. Besides, the sub-array implements single cycle `add/sub` operation leveraging CSA. Comprehensive experiments on process variation of 2-/3- row activation mechanisms have been performed in our preliminary works [22], [28].

IV. Hardware Evaluation Results

In this section, we elaborate on the evaluation configuration, then study the performance trade-off between the hardware specifications (e.g., overhead, memory, energy) and accuracy.

A. Platform Setup

The PIM accelerator is configured with 256×512 memory sub-array and 512Mb total capacity adhered to an H-tree routing fashion. We built a device to architecture evaluation framework starting from the device modeling using Non-Equilibrium Green Function (NEGF) and Landau-Lifshitz-Gilbert (LLG) equation with spin Hall effect term [22], [30]. The we developed a Verilog-A model for 2T1R SOT-MTAM cell and used it in Cadence Spectre to simulate it with other CMOS circuitry. To estimate the circuit performance, we used 45nm NCSU PDK library [31]. Based on the results, we got from circuit simulations, we extensively modified NVSim [32] by presenting a particular PIM library at architecture level. The simulator updates the configuration files (.cfg) w.r.t. various memory array organization and the model size. We then coded a behavioral simulator that calculates the latency and energy parameters that the platform consumes to run the uniform dynamic neural network. Additionally, we considered a mapping optimization algorithm w.r.t. the resources to boost the throughput. Here, we take the 8-bit quantized ResNet20 to run on the CIFAR-10 data-set.

B. Energy Consumption

In this subsection, we analyze and report the energy consumption of the presented PIM platform emphasizing on two intensive operations i.e. the bit-wise AND and write-back. In this way, Fig. 5a depicts the break-down of number of AND operations in various convolutional layers of ResNet20. Similarly, the latency is proportional to the computation load. Fig. 5b shows the estimated energy consumption of ResNet20 with various model sizes split into 4 regions namely: AND-compute, AND-WB, bitcount-shift, and add operations. It is worth pointing out that we plotted the energy consumption of Bit-Counter and Shifter together in Fig. 5b. Based on the figure, it can be observed that the number of PIM operations and so energy consumption diminish as a result of presented adaptive sub-array pruning technique. The presented technique essentially eliminates the energy overhead of the the indolent computational sub-arrays sacrificing the accuracy. Based on our analysis, the energy consumption of 0.25× model reduces by a factor of 2.7 as compared with the baseline PIM platform.

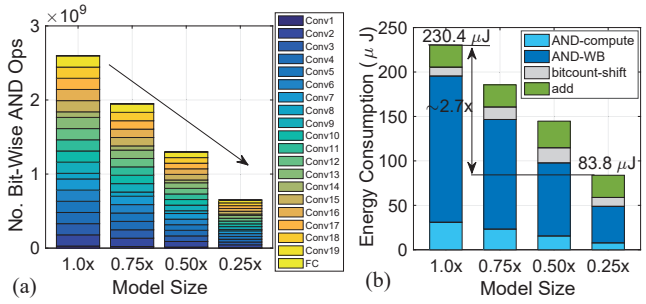


Figure 5: Break-down of (a) number of AND Ops to process ResNet20's convolutional layers w.r.t. model size, (b) Energy consumption of the PIM design under various model sizes [1].

C. Memory Storage Requirement

Fig. 6 reports the PIM platform's memory storage to execute a full-precision and quantized (8-bit) ResNet20 model for CIFAR-10 with various model sizes. We can see that by adaptively tuning the channel-width, the memory storage is different. In addition, Fig. 6 depicts the accuracy and the memory storage trade-offs for the PIM design. In the figure, for example 8:0.50x configuration represents 8-bit-quantized ResNet20 with 0.50x as model size.

D. Area Overhead

We consider 3 important hardware cost sources to estimately calculate the area overhead of the design. (1) The CSAs' add-on transistors connected to every BL. (2) The enhanced multi-row activated MRD overhead; we edited all WL driver by inserting two extra transistors in the memory decoder buffer. (3) The overhead of Ctrl's to handle CSA's enable bits. We consider area of the counter and shifter as a part of Ctrl area. In general, the PIM platform incurs 7.9% overhead to main memory die [1]). Fig. 7a reports such area overhead

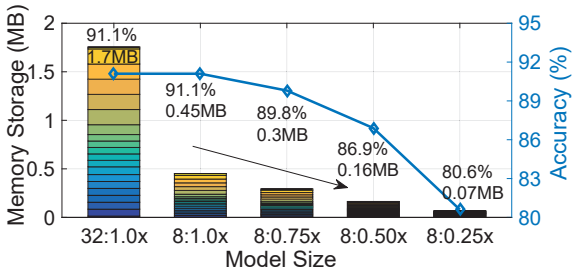
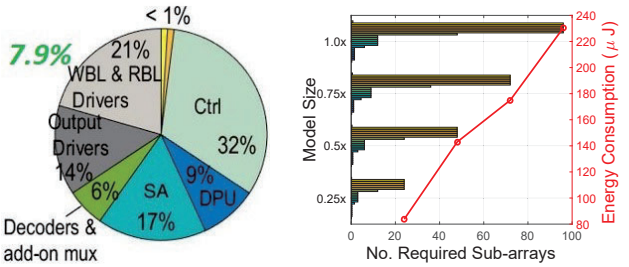


Figure 6: The accuracy/memory storage trade-off in our PIM platfrom for various model sizes [1]).

breakdown. In addition, in Fig. 7b, we profile the distribution of area of the PIM components to run ResNet-20 for various convolutional layers to process an image with various model sizes. Besides, having various number of computational sub-arrays, Fig. 7b shows the energy consumption by different model sizes. According to this, we can clearly see how effectively the presented adaptive sub-array pruning technique could be leveraged to reduce the PIM platform's power budget through dynamically cutting down the sub-arrays.



(a) (b)

Figure 7: (a) The area overhead pie-chart of the PIM design, (b) Trade-off between energy consumption and area for various model sizes [1]).

V. Conclusion

In this work, we explicitly review the method to construct a dynamic neural network, which mainly includes two steps: sub-nets generation and fused sub-nets training. Two different sub-nets generation methods are presented, which are used for the uniform and non-uniform dynamic neural networks respectively. Moreover, in terms of the hardware acceleration, we discuss a processing-in-MRAM platform for such neural network to accelerate dynamic inference. In the end, the performance trade-off between the hardware specification (e.g. memory, energy, overhead) and accuracy is elaborated.

References

[1] L. Yang *et al.*, "A flexible processing-in-memory accelerator for dynamic channel-adaptive deep neural networks," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020.

[2] L. Yang, Z. He, and D. Fan, "Non-uniform dnn structured subnets sampling for dynamic inference," *57th Design Automation Conference (DAC)*, 2020.

[3] Y. LeCun *et al.*, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[4] Z. He and D. Fan, "Simultaneously optimizing weight and quantizer of ternary neural network using truncated gaussian approximation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 438–11 446.

[5] 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.

[6] S. Han *et al.*, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *ICLR'16*.

[7] L. Yang *et al.*, "Harmonious coexistence of structured weight pruning and ternarization for deep neural networks." in *AAAI*, 2020, pp. 6623–6630.

[8] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint:1704.04861*, 2017.

[9] M. Sandler *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *CVPR*, 2018, pp. 4510–4520.

[10] H. Li *et al.*, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[11] D. Molchanov *et al.*, "Variational dropout sparsifies deep neural networks," in *ICML*. JMLR. org, 2017, pp. 2498–2507.

[12] W. Wen *et al.*, "Learning structured sparsity in deep neural networks," in *NIPS*, 2016, pp. 2074–2082.

[13] B. Zoph *et al.*, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[14] C. Liu *et al.*, "Progressive neural architecture search," in *ECCV*, 2018, pp. 19–34.

[15] H. Liu *et al.*, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[16] W. Jiang, L. Yang, S. Dasgupta, J. Hu, and Y. Shi, "Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start," *arXiv preprint arXiv:2007.09087*, 2020.

[17] J. Yu *et al.*, "Slimmable neural networks," *arXiv preprint arXiv:1812.08928*, 2018.

[18] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 27–39, 2016.

[19] S. Angizi, Z. He *et al.*, "Cmp-pim: an energy-efficient comparator-based processing-in-memory neural network accelerator," in *Proceedings of the 55th Annual Design Automation Conference*. ACM, 2018, p. 105.

[20] K. Ni *et al.*, "Ferroelectric ternary content-addressable memory for one-shot learning," *Nature Electronics*, vol. 2, no. 11, pp. 521–529, 2019.

[21] S. Li *et al.*, "Drisa: A dram-based reconfigurable in-situ accelerator," in *MICRO*, 2017.

[22] S. Angizi *et al.*, "Imce: energy-efficient bit-wise in-memory convolution engine for deep neural network," in *Proceedings of the 23rd ASP-DAC*.

[23] Z. He, S. Angizi, and D. Fan, "Accelerating low bit-width deep convolution neural network in mram," in *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2018, pp. 533–538.

[24] G. Hinton *et al.*, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[25] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[26] J. Deng *et al.*, "Imagenet: A large-scale hierarchical image database," in *CVPR*. Ieee, 2009, pp. 248–255.

[27] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE CVPR*, 2016, pp. 770–778.

[28] S. Angizi *et al.*, "Graphs: A graph processing accelerator leveraging sot-mram," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 378–383.

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

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

[31] (2011) Ncsu eda freepdk45. [Online]. Available: http://www.eda.ncsu.edu/wiki/FreePDK45:Contents

[32] 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.