Sparse BD-Net: A Multiplication-less DNN with Sparse Binarized Depth-wise Separable Convolution

ZHEZHI HE and LI YANG, School of Electrical, Computer and Energy Engineering, Arizona State University
SHAAHIN ANGIZI, Department of ECE, University of Central Florida
ADNAN SIRAJ RAKIN and DELIANG FAN, School of Electrical, Computer and Energy Engineering, Arizona State University

In this work, we propose a multiplication-less binarized depthwise-separable convolution neural network, called BD-Net. BD-Net is designed to use binarized depthwise separable convolution block as the drop-in replacement of conventional spatial-convolution in deep convolution neural network (DNN). In BD-Net, the computation-expensive convolution operations (i.e., Multiplication and Accumulation) are converted into energy-efficient Addition/Subtraction operations. For further compressing the model size while maintaining the dominant computation in addition/subtraction, we propose a brand-new sparse binarization method with a hardware-oriented structured sparsity pattern. To successfully train such sparse BD-Net, we propose and leverage two techniques: (1) a modified group-lasso regularization whose group size is identical to the capacity of basic computing core in accelerator and (2) a weight penalty clipping technique to solve the disharmony issue between weight binarization and lasso regularization. The experiment results show that the proposed sparse BD-Net can achieve comparable or even better inference accuracy, in comparison to the full precision CNN baseline. Beyond that, a BD-Net customized process-in-memory accelerator is designed using SOT-MRAM, which owns characteristics of high channel expansion flexibility and computation parallelism. Through the detailed analysis from both software and hardware perspectives, we provide an intuitive design guidance for software/hardware co-design of DNN acceleration on mobile embedded systems. Note that this journal submission is the extended version of our previous published paper in ISVLSI 2018 [24].

CCS Concepts: • **Computer systems organization → Embedded systems**; **Neural networks**;

Additional Key Words and Phrases: Deep neural network, model compression, in-memory computing

ACM Reference format:
Zhezhi He, Li Yang, Shaahin Angizi, Adnan Siraj Rakin, and Deliang Fan. 2020. Sparse BD-Net: A Multiplication-less DNN with Sparse Binarized Depth-wise Separable Convolution. *J. Emerg. Technol. Comput. Syst.* 16, 2, Article 15 (January 2020), 24 pages.
https://doi.org/10.1145/3369391

This work is supported in part by the National Science Foundation under Grant Nos. 1740126, 1908495, 1931871, and Semiconductor Research Corporation nCORE.

Authors' addresses: Z. He, L. Yang, A. S. Rakin, and D. Fan, School of Electrical, Computer and Energy Engineering, Arizona State University, 650 E Tyler Mall, Tempe, Arizona, 85287-5706; emails: {zhezhihe, lyang166, asrakin, dfan}@asu.edu; S. Angizi, Department of ECE, University of Central Florida, 4328 Scorpius Street, Orlando, Florida, 32816-2362; email: angizi@knights.ucf.edu.

1550-4832/2020/01-ART15 $15.00
https://doi.org/10.1145/3369391

1 INTRODUCTION

Owing to the explosion of data, improvement of parallel computing ability resulting from GPU and continuous breakthroughs in algorithms, Artificial Neural Network (ANN) has achieved great success in recent years. Deep Neural Network (DNN), as one of the most popular state-of-the-art ANN, has shown its leading performance in various domains, such as speech recognition, computer vision, and data analysis [34]. Both theoretically and empirically, DNN has shown significant performance improvement over its shallow counterparts [37, 50].

Recently, the state-of-the-art DNN could achieve surpassing human accuracy in object classification task for large-scale datasets (e.g., ImageNet) [20, 32]. However, from the perspective of hardware deployment, DNNs still encounter the drawbacks in terms of computation and memory cost [23]. Many recent works have been proposed to address such high computational complexity and memory usage issues of existing DNN structures, including weight quantization [46] and pruning [51], Hoffman weight encoding [17], and compact layer with reduced number of parameters (e.g., depthwise-separable convolution [28]). For DNN deployment in hardware platforms with energy-efficiency and throughput requirement, weight quantization has become one of the must-have processing steps, where the quantization schemes can range from 8-bit resolution (high bit-width) [17, 53] to 1-bit resolution (extremely low bit-width) [25, 26, 29, 46]. In past years, massive research efforts have been invested in the direction of extremely low bit-width model quantization (i.e., binary $w \in \{-1, +1\}$), however, the quantized models still normally suffer from accuracy degradation, owing to the information loss caused by aggressive quantization scheme.

Using depthwise-separable convolution layer as the replacement of conventional 2-D spatial convolution layer is originally proposed and discussed in famous compact DNN – MobileNet [28], which further substitutes the conventional convolution in the state-of-the-art DNN architectures, including Xception [12], NAS-Net [54], and PNAS-Net [39]. It is known that the sensitivity to compression (i.e., quantization /pruning) varies for different layers. Thus, a straightforward countermeasure adopted in Reference [41] is to directly increase the width of layers (i.e., number of input/output channels). However, the drawback of width change in the aforementioned method is that the widths of connected layers are modified as well. In this work, the advantage of using depthwise separable convolution is not only its outstanding performance with compact structure, but also the representation capability can be changed through inner width expansion, which is decoupled with neighboring layers. With such characteristics—the combination of depthwise-separable convolution with our proposed sparse binarization technique—it is easy to achieve negligible accuracy drop with high model compression rate.

In this work, we discuss two binarization schemes to work along with the depthwise-separable convolution. The difference between the two schemes is twofold: function and location. For binarization function, scheme-1 adopts the identical function that is proposed in the original binary weight network in Reference [40]; while scheme-2 leverages the sparse binarization function together with the structure weight penalty and weight penalty clipping techniques that are proposed by us. For binarization location, scheme-1 performs the binarization on both the weight and output feature maps of depthwise convolution layer. Different from that, scheme-2 performs structured sparse binarization on the weights of depthwise and pointwise convolution layers. In general, we investigate the binarization on depthwise-separable convolution [48] block as an alternative module to conventional spatial convolution layer for maximizing the model size reduction and computation simplification with negligible accuracy degradation. The advantage of using binarized depthwise-separable convolution block is mainly threefold:

- Depthwise-separable convolution performs the same function of feature extraction and recombination as conventional convolution, but with less amount of parameters and computation workload.

- Binarization on weights/activation not only further reduces the model size (represented weight in binary-1 bit from original FP-32 bit), but also converts the dominant Multiplication and Accumulation (MAC) into pure Accumulation (i.e., Add/Sub).
- Through increasing the *channel multiplier* (a.k.a. inner channel expansion), the information loss caused by the aggressive binarization can be compensated, while the inner channel expansion does not affect the structures (i.e., number of input/output channels) of other externally connected layers.

With the advantages described above, there exist two binarization schemes that will be discussed in the following sections. One binarization scheme is mixed weight and activation binarization, while another scheme is weight binarization only. Both binarization schemes realize the same computation complexity reduction (i.e., using Add/Sub) with varying model size reduction ratio.

This article is the extended version of our previous published paper in ISVLSI 2018 [24] (which was also the best paper winner). We have extensively extended from the conference version with main contributions summarized as follows:

(1) We propose to use binarized depthwise-separable convolution with inner channel expansion as the drop-in replacement to conventional convolution layer, where we call neural networks constructed by such modules *BD-Net*. Two different binarization schemes are proposed and investigated. Theoretic analysis is provided as well to examine that our proposed binarized depthwise-separable convolution can effectively approximate the conventional spatial convolution.
(2) We further explore to introduce structured sparsity into binarized weight, utilizing a *Lasso*-based regularization method, which can further compress the model size and provide hardware-friendly sparse weight footprints for efficient computing. For solving the disharmony issue between weight-sparse binarization and group-lasso regularization, an effective weight penalty clipping technique is proposed. Intensive experiments are conducted to show the sparse BD-Net can achieve significant model compression rate with negligible accuracy degradation.
(3) From the hardware implementation perspective, we bring up a customized in-memory computing DNN accelerator to fully leverage the performance boost resulting from our proposed algorithm—BD-Net and its sparse variant. Our proposed accelerator is designed based on Spin-Orbit Torque Magnetic-RAM (SOT-MRAM), which integrates memory and logic functions upon a set of novel micro-architectural and circuit-level solutions. Accordingly, we present detailed analysis in terms of hyper-parameter configuration of sparse BD-Net, inference accuracy, and hardware resource utilization (e.g., throughputs, energy). It provides intuitive design guidance for hardware engineers for Software/Hardware co-design of DNN on mobile embedded systems.

For clarification, compared to our published conference paper [24], the contributions (2) and (3) enumerated above are new.

2 BINARIZED DEPTHWISE SEPARABLE CONVOLUTION

In this section, we first introduce the depthwise separable convolution module as an alternative to the conventional spatial convolution layer. Then, we propose and describe our binarization schemes. Based on such binarization schemes, we construct a DNN, called BD-Net, as a benchmark to examine the performance. In addition to normal BD-Net, we propose a sparse BD-Net variant, which leverages a *Lasso*-based regularization technique for introducing hardware-friendly structured sparsity to binarized weight footprint. At last, a theoretic analysis is presented as well to

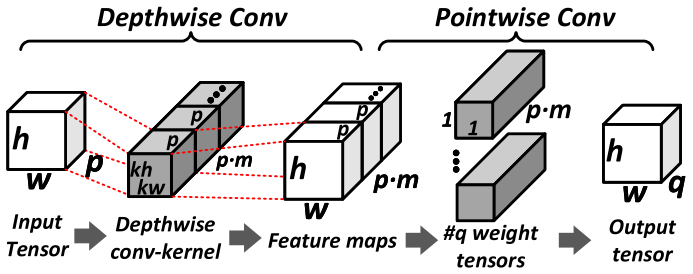


Fig. 1. Data-flow for depthwise separable convolution layer. Normally, the channel expansion m is 1. Feature maps (inter-layer tensors) are in white, while weight tensors are in grey.

demonstrate the proposed sparse binarized depthwise-separable convolution module can approximate the conventional spatial convolution layer.

2.1 Depthwise Separable Convolution

Recently, the depthwise-separable convolution [48] has been widely used in many state-of-the-art deep neural networks, such as MobileNet [28], Xception [12], and NasNet [55] (basic block in Google AutoML), which replaces the conventional spatial convolution layers to reduce DNN computational cost and memory usage. As a factorized form of conventional spatial-convolution, the depthwise separable convolution consists of two parts: depthwise convolution and pointwise convolution (i.e., 1×1 convolution). The conventional spatial-convolution mainly performs channel-wise feature extraction, then combines those features to generate new representations. It is proven that such two-step tasks could be handled by depthwise convolution and pointwise convolution separately [48].

The operation of depthwise separable convolution is described in the form of data flow in Figure 1. Consider the input tensor is in the dimension of $h \times w \times p$, which denotes height, width, and channel, respectively. Note that, in the depthwise convolution layer, each input channel performs convolution with m kernels in the size of $kh \times kw$ (i.e., kernel-height and width) correspondingly, which produces $p \cdot m$ feature maps. m is defined as *channel expansion* in this work. We found that larger m could effectively compensate for accuracy degradation in weight binarization at the cost of model size, which will be explicitly investigated in the next section. Those generated feature maps are concatenated along its depth dimension as a $h \times w \times (p \cdot m)$[1] tensor, then taken as the input of the following pointwise convolution layer. The operation of such depthwise convolution can be mathematically described as:

$$\boldsymbol{F}_{j \cdot \mathrm{p}+i} = \boldsymbol{X}_i * \boldsymbol{W}_{j \cdot \mathrm{p}+i}, \quad i \in [1, \mathrm{p}]; j \in [0, m-1], \tag{1}$$

where i is the channel index input feature map $\boldsymbol{X} \in \mathbb{R}^{h \times w \times p}$. $(*)$ denotes the convolution (i.e., dot-product) between input and kernel $\boldsymbol{W}$. Contrary to the distinctive depthwise convolution, the pointwise layer is just a normal spatial-convolution layer with 1×1 convolution kernel size. Thus, it only linearly combines the $p \cdot m$ input feature maps to generate new representations in $q \times h \times w$.

From the perspective of computation, the cost of normal spatial-convolution layer is $h \times w \times kh \times kw \times p \times q$. As its approximating alternative, the cost of depthwise convolution is $h \times w \times kh \times kw \times p \times m$, while the cost of pointwise convolution is $h \times w \times p \times m \times q$. Thus, the ratio of computational cost between depthwise separable convolution and conventional convolution can

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

be calculated as:

$$\frac{h \cdot w \cdot kh \cdot kw \cdot p \cdot m + h \cdot w \cdot p \cdot m \cdot q}{h \cdot w \cdot kh \cdot kw \cdot p \cdot q} = \frac{m(kh \cdot kw + q)}{kh \cdot kw \cdot q} \overset{q>>kh \cdot kw}{\approx} \frac{m}{kh \cdot kw}, \tag{2}$$

where such ratio is approximated to $m/(kh \cdot kw)$ when the number of output channels $q >> kh \cdot kw$. Normally, when the channel expansion m is set to 1 (e.g., MobileNet [28]) and the kernel size $kh \times kw$ is 3×3, the computational cost is only $1/9$ of the normal spatial-convolution. Increasing the kernel size will lead to higher computational cost reduction. Note that the accuracy degradation can be minimized through choosing larger m, where its effect will be analyzed in Section 3.2.1.

2.2 Binarization Schemes

In this subsection, we discuss two binarization schemes where scheme-1 is a mixed binarization of weights and output feature maps of depthwise convolution, while scheme-2 is to fully binarize the weight of both depthwise and pointwise convolution layers with additional sparsity introduced.

2.2.1 Scheme-1: Mixed Binarization of Weight and Activation. For scheme-1, we directly apply sign function $sgn(\cdot)$ as the deterministic binarization activation function, similar to Reference [29], whose output is either +1 or −1. The non-differential problem of the $sgn(\cdot)$ is solved by well-known Straight-Through Estimator (STE) [9] with gradient approximation, which is widely adopted by many DNN quantization works. In the backward path, the input gradient of binarization activation function clones the gradient at output if it is in the range from −1 to +1. Otherwise, the gradient is canceled to preserve training performance. For quantization function without scaling, gradient clipping is quite important to avoid the approximated gradient error, which is well explained in literature [25]. In summary, such binarization activation function in forward and backward can be described as:

$$\text{Forward}: \quad \hat{r}_i = sgn(r_i) = \begin{cases} +1 & \text{if } r_i \geq 0, \\ -1 & \text{otherwise;} \end{cases} \tag{3}$$

$$\text{Backward}: \quad \frac{\partial \hat{r}_i}{\partial r_i} = \begin{cases} 1 & \text{if } |r_i| \leq 1 \\ 0 & \text{otherwise} \end{cases}; \qquad \forall r_i \in \boldsymbol{R}, \hat{r}_i \in \hat{\boldsymbol{R}}, \tag{4}$$

where $\boldsymbol{R}$ and $\hat{\boldsymbol{R}}$ are tensors before and after binarization, respectively. In scheme-1, we only apply Equation (4) and Equation (3) on the weight kernels and output feature maps of depthwise convolution in Figure 1. Such binarization scheme ensures the dominant `MAC` operations are converted in `Add/Sub` (i.e., pure accumulation). Note that the model size reduction results from scheme-1 is quite small, since the weights of the depthwise convolution layer only take a small portion of the entire model size. Note that the weight binarization process is conducted same as other famous quantization works [17, 29], which keeps two weight systems: one weight copy in FP32-bit for weight update using back-propagation, while another weight copy that binarized from FP32-bit counterpart is used for inference and calculating the inference loss.

2.2.2 Scheme-2: Sparse Binarization of Weight. In contrast to scheme-1, we propose to fully binarize the weights of both depthwise and pointwise convolution layers without activation binarization. Such binarization methodology follows the design guideline of memory usage and access (i.e., weight storage and loading from off-chip DRAM) that are currently the computational bottlenecks for DNN inference acceleration [17]. However, such combination may lead to $(kh \cdot kw \cdot 32)/m \times$ compression ratio, where such aggressive binarization method normally leads to significant accuracy degradation. Besides directly tuning channel multiplier m for model capacity compensation, we propose to introduce weight sparsity as an additional countermeasure.

Since introducing sparsity into binarized weight is mathematically equivalent to encode weights in ternary representation ($w \in \{-1, 0, +1\}$), we follow the weight quantization function from Reference [26], thus the quantization on lth layers' weight $\boldsymbol{W}_l$, which can be expressed as:

$$\text{Forward:} \quad \hat{w}_{i,l} = \alpha_l \cdot sbin(w_{i,l}) = \begin{cases} +\alpha_l & w_{i,l} > \Delta_l \\ 0 & |w_{i,l}| \le \Delta_l \\ -\alpha_l & w_{i,l} < -\Delta_l \end{cases}; \quad w_{i,l} \in \boldsymbol{W}_l \tag{5}$$

$$\alpha_l = \mathbb{E}(|\{w_{i,l}\}|), \quad \forall \{i \Big| |w_{i,l}| > \Delta_l\} \tag{6}$$

$$\text{Backward:} \quad \frac{\partial \hat{w}_{i,l}}{\partial w_{i,l}} = 1, \tag{7}$$

where α_l in Equation (6) and $\Delta_l = 0.05 \times max(|\boldsymbol{W}_l|)$ are layer-wise scaling coefficient and threshold, respectively, that calculated in the run-time during training. Contrary to the gradient approximation described in Equation (4), we emit the gradient clipping operation in Equation (7) for higher flexibility of trained weight scale, thanks to the incorporation of α_l in Equation (5).[2] The advantages of introducing additional sparsity are as follows: (1) It adds extra quantization level, which mitigates the information loss during model compression; (2) the dominant computation is still completed through `Add/Sub`, which is identical as dense binarization counterpart (scheme-1). However, directly applying scheme-2 still counters the drawback that the trained sparse weight patterns are highly irregular, which is not hardware-friendly in terms of sparsity encoding and computation efficiency. To generate a structured sparse pattern, we propose a regularization technique to achieve this goal as specified in Section 2.3.

2.3 Structured Sparse Pattern Through Regularization

Previous studies of weight sparsity can be mainly divided into two categories: non-structured sparsity [17, 18], and structured sparsity [33, 38, 51], where the main difference between these two counterparts is the regularity of sparse weight pattern. The non-structured sparsity methods lead to highly irregular sparse weight pattern, which makes it extremely difficult to encode the sparse weight efficiently due to the sparse indexing. Even though the non-structured sparsity method normally shows less accuracy degradation owing to the relatively higher flexibility in weight pattern, its performance improvement on hardware deployment is not tempting. In contrast to that, the structured weight pattern in a regular fashion (e.g., kernel-wise/channel-wise [51]) uses less memory for sparsity indexing and easily skips the computation w.r.t weights in zero value.

2.3.1 Definition of Structured Sparse Weight Pattern. Different from other structured sparsity methods [51], for maximizing the computation efficiency of DNN with sparse weight pattern on designated accelerator, we define the structured sparse pattern in the same shape of capacity as basic computing core in that accelerator. Taking a conventional convolution layer with 4-dimensional weight tensor $\boldsymbol{W} \in \mathbb{R}^{q \times p \times kh \times kw}$ as an example, the hardware accelerator normally reshapes the 4-D weight tensor into 2-D matrix in the shape of $(q, p \times kh \times kw)$ to perform matrix multiplication. Owing to the architecture difference between the depthwise convolution layer and pointwise convolution layer in Figure 1, we design structured sparse weight pattern for them separately.

For the depthwise convolution, since its weights are in dimension of $(p \times m, kh \times kw)$, we set the sparse weight pattern as $kh \times kw$, which means our structured sparsity method is supposed to force 2D kernels in the shape of $kh \times kw$ to be zero during training. As we flatten the weight tensor of pointwise convolution layer into a vector in dimension of $q \times p \cdot m \times 1 \times 1$, the sparse pattern

[2]Please refer to the gradient correction theory described in Reference [25] for explanation.

is designed as a vector whose length is a fixed value g. Note that g is configured as a universal sparse pattern across different pointwise convolution layers throughout the entire DNN. In this work, since g is not only the length of sparse pattern for pointwise convolution layer, but also the capacity of basic computing core of designed accelerator, it is expected that the chosen g can balance the DNN inference accuracy and computation parallelism of the designated accelerator. A rough range of g is $g = 2^n$ and $kh \cdot kw < g < p \cdot m$.

2.3.2 Group-Lasso Regularization with Hardware-friendly Sparse Pattern. To obtain structured binarized weight pattern in the predefined pattern as described in Section 2.3.1, we propose an accelerator-specific group-lasso regularization method originated from Reference [51]. Given the vectorized input tensor $\boldsymbol{x}$ and target label $\boldsymbol{t}$, the loss function used for training the targeted DNN is reformatted as an ensemble loss function:

$$\mathcal{L}_{\text{ens}} = \mathcal{L}\Big(f\Big(\boldsymbol{x}; \{\hat{\boldsymbol{W}}_l\}_{l=1}^{L}\Big), \boldsymbol{t}\Big) + \lambda \sum_{l=1}^{L} \sum_{i=1}^{G_l} \mathcal{P}(\boldsymbol{W}_{l,i}), \tag{8}$$

where $f(\boldsymbol{x}; \{\hat{\boldsymbol{W}}_l\}_{l=1}^{L})$ computes the outputs of DNN parameterized by the sparse binarized weight tensors $\{\hat{\boldsymbol{W}}_l\}_{l=1}^{L}$ w.r.t. the input $\boldsymbol{x}$. $\mathcal{L}(\cdot, \cdot)$ is the objective function of DNN (i.e., cross-entropy loss in this work). $\mathcal{P}(\boldsymbol{W}_{l,i}) = ||\boldsymbol{W}_{l,i}||_2$ is the L^2-norm of the indexed weight group $\boldsymbol{W}_{l,i}$. The second term in the R.H.S of Equation (8) is the L^1-norm of $\mathcal{P}(\boldsymbol{W}_{l,i})$ (a.k.a. group lasso [51]), which acts as the group-wise weight penalty for improving the group-wise sparsity during the optimization. G_l is the number of groups in the lth layer, and λ is the hyper-parameter to be tuned based on the dataset.

2.3.3 Weight Penalty Clipping with Self-adjustable Threshold. Since the structured regularization term proposed in Equation (8) is designed to integrate with sparse weight binarization functions (Equation (5) to Equation (7)) in scheme-2, it will encourage the intensity of spatially neighboring weights to drop below thresholds Δ_l with defined sparse pattern. However, the preliminary experiments conducted by us indicate the disharmony when naively combining two methods. As the countermeasure, we make the further adjustment on the in-group L^2-norm weight penalty, which can be described as:

$$\begin{aligned} \delta_l &= a \cdot \frac{1}{G_l} \sum_{i=1}^{G_l} ||\boldsymbol{W}_{l,i}||_2, \\ \text{s.t.} \quad \mathcal{P}(\boldsymbol{W}_{l,i}) &= \min(||\boldsymbol{W}_{l,i}||_2, \delta_l), \end{aligned} \tag{9}$$

where δ_l is the layer-wise threshold to clip the in-group L^2-norm weight penalty, and a is coefficient to scale δ_l. Note that once the in-group L^2-norm penalty of $\boldsymbol{W}_{l,i}$ is clipped, the cross-group L^1-norm penalty is clipped as well. We name such method as the weight penalty clipping. Consider two cases:

- When $||\boldsymbol{W}_{l,i}||_2 \geq \delta_l$, it indicates that weights in $\boldsymbol{W}_{l,i}$ are relatively large (i.e., important), which are not supposed to be pruned by the group-lasso term in Equation (9). Then, the weight penalty clipping is performed that replaces the weight penalty of $\boldsymbol{W}_{l,i}$ in $\mathcal{L}_{\text{ens}}$ with δ_l. Here, we have to highlight that δ_l is treated as a constant, where its calculation is removed from the backward computation graph.
- When $||\boldsymbol{W}_{l,i}||_2 < \delta_l$, we keep the weight penalty of $\boldsymbol{W}_{l,i}$ in its original format, thus the group-lasso term in Equation (8) can continuously affect on $\boldsymbol{W}_{l,i}$ and prune the weights in group-wise fashion.

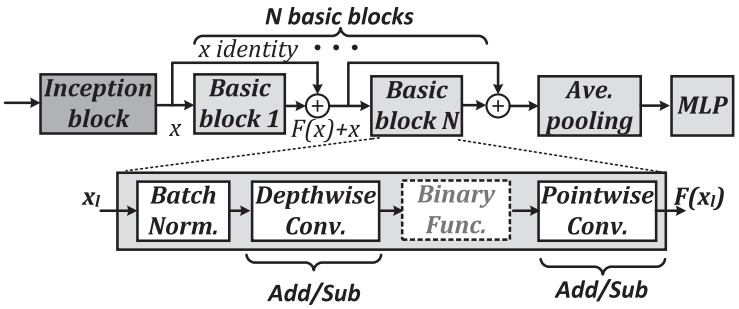


Fig. 2. Block diagram of the multiplication-less BD-Net. For binarization scheme-1, the binarized activation function is inserted between depthwise (with binarized weight) and pointwise convolution layers. For the sparse BD-Net using binarization scheme-2, there is no binarized activation function.

Our weight penalty clipping method is an optimized version of the method adopted by Lebedev et al. in Reference [33]. The original work manually set the δ_l, which is fixed throughout the entire training process. Instead of using fixed δ_l, we propose the self-adjustable δ_l as described in Equation (9), which gives higher flexibility for the group-wise weight pruning. Based on our simulation results, the weight penalty clipping with self-adjustable threshold is helpful to mitigate the disharmony between group-wise weight pruning (i.e., group lasso) and weight-sparse binarization.

2.4 BD-NET: DNN with Binarized Depthwise Separable Convolution

Taking depthwise-separable convolution module as a basic component with two proposed binarization schemes introduced in Section 2.2 and additional structured sparsity regularization technique, we construct a DNN referring to the topology of ResNet [19], called BD-NET. Note that ResNet architecture is only used as an example; other state-of-the-art network architecture could be used as well. The block diagram of DNN analyzed in this work is presented in Figure 2. It sequentially composes an inception block[3] (3×3 spatial convolution, Batch-normalization and ReLU), N depthwise separable convolution basic blocks, average pooling layer, and Multi-Layer Perceptron (MLP, a.k.a. fully connected layer). As the key component in our proposed neural network, basic block includes batch normalization, depthwise convolution with binarized weight, binarized activation function, and pointwise convolution. It is worth noting that for the sparse binarization method, there is no binarized activation function. Similar to BWN [46] and LBCNN [31], we place the batch normalization in front of the convolution layer with binarized weight, since scheme-1 does not include an internal scaling coefficient, thus utilizing batch-norm layer to perform the scaling function.

In summary, the output response of basic block for the two binarization schemes can be correspondingly expressed as:

$$\text{Scheme-1:} \quad x_{l+1}^t = \sum_{s=1}^{p\cdot m} Sgn(Sgn(W_l^s) * BN(x_l^s)) \cdot \overline{W}_{l,s}^t, \tag{10}$$

$$\text{Scheme-2:} \quad x_{l+1}^t = \sum_{s=1}^{p\cdot m} ((\alpha_l \cdot sbin(W_l^s) * BN(x_l^s))) \cdot (\overline{\alpha}_l \cdot sbin(\overline{W}_{l,s}^t)), \tag{11}$$

where $s \in [1 : p \cdot m]$ and $t \in [1 : q]$ indexes the input channel and output channel, respectively. l is the module index of basic block, while p is the number of input channels of lth basic block. W_l

[3]Similar to previous works of binarized/quantized neural network [29, 46, 53], we do not introduce binarization to the first inception block.

Table 1. Hardware Cost Analysis [28] of Standard Convolution in CNN and Binarized Depthwise Separable Convolution in This Work

	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 Scheme-1	0	$h \cdot w \cdot kh \cdot kw \cdot p \cdot m + h \cdot w \cdot p \cdot m \cdot q$	$kh \cdot kw \cdot p \cdot m \cdot 1 + p \cdot m \cdot q \cdot 32$
This work Scheme-2	$h \cdot w \cdot p \cdot m + h \cdot w \cdot q$	$h \cdot w \cdot kh \cdot kw \cdot p \cdot m + h \cdot w \cdot p \cdot m \cdot q$	$kh \cdot kw \cdot p \cdot m \cdot 2 + p \cdot m \cdot q \cdot 2$
$\frac{\text{scheme-1}}{\text{CNN}}$	0	$\frac{m}{q} + \frac{m}{kh \cdot kw}$	$\frac{1}{q \cdot 32} + \frac{m}{kh \cdot kw}$
$\frac{\text{scheme-2}}{\text{CNN}}$	$\frac{m}{q \cdot kh \cdot kw} + \frac{m}{p \cdot kh \cdot kw}$	$\frac{m}{q} + \frac{m}{kh \cdot kw}$	$\frac{2}{q \cdot 32} + \frac{m \cdot 2}{kh \cdot kw \cdot 32}$

Multiplications of Batch-norm are excluded.

is the learned depthwise convolution kernel in FP-32, while $\overline{W}_l$ denotes the weights of pointwise convolution layer. α_l and $\bar{\alpha}_l$ are the scaling coefficients in binarization scheme-2, corresponding to depthwise convolution and pointwise convolution, respectively. $BN()$ is the Affine function of batch normalization layer [30].

The objective of BD-Net is to build an efficient hardware-friendly compact DNN while preserving accuracy. To demonstrate the efficiency of BD-Net with/without sparsity, we analyze the hardware cost of standard spatial convolution and binarized depthwise separable convolution in terms of computation cost and parameter size. For the cost analysis, we adopt the similar method adopted in Reference [28] to give a general idea about the computation and memory cost with different configurations. As tabulated in Table 1, for computing complexity, it can be seen that scheme-1 not only fully replaces the `MAC` to the hardware efficient `Add/Sub`, but also reduces the number of operations by a factor of $\sim m/(kh \cdot kw)$. Scheme-2 keeps the same number of `Add/Sub` operations with scheme-1, but significantly reduces the memory cost by an extra 16×. Note that, for estimating the memory cost of scheme-2, we consider the worst case that the introduced structured sparse weight pattern leads to no further memory reduction.

2.5 Theoretic Analysis

Theoretically, if the binarized depthwise separable convolution (both scheme-1 and scheme-2) can effectively approximate the conventional spatial-convolution with its weights in full-precision, such two types of binarized convolution layers are supposed to be capable of producing similar output tensor w.r.t the same input tensor [31, 45]. The detailed analysis is performed below.

2.5.1 Scheme-1. First, for fair comparison, let us assume the input tensor and output tensor of the conventional convolution and binarized depthwise separable convolution are in the identical dimension. We define the real value weight of standard convolution as $\boldsymbol{W'} \in \mathbb{R}^{p \times q \times kh \times kw}$, and the weight of depthwise convolution as $\boldsymbol{W} \in \mathbb{B}^{(p \cdot m) \times kh \times kw}$, where $\mathbb{B} = \{-1, +1\}$. Note that we temporarily ignore the dimension of mini-batch for simplicity. We define y as one element in the output tensor of our proposed binarized depthwise separable convolution. y' is the corresponding element (i.e., with identical dimension index) in the output tensor of conventional spatial-convolution. Both of those two types of convolution are fed with same input tensor $\boldsymbol{X} \in \mathbb{R}^{p \times h \times w}$. For calculating y', a subset of weights in $\boldsymbol{W'}$ is vectorized as $\boldsymbol{w'} \in \mathbb{R}^{(p \cdot kh \cdot kw) \times 1}$. Meanwhile, in the input tensor $\boldsymbol{X}$, a vectorized patch $\boldsymbol{x} \in \mathbb{R}^{(p \cdot kh \cdot kw) \times 1}$ is selected by $\boldsymbol{w'}$ to perform dot-product computation. Thus, the microscopic output of convolution for two types of convolution can be mathematically described

as:

$$y' = \boldsymbol{w'}^T \boldsymbol{x} \in \mathbb{R}, \tag{12}$$

$$y = (Sgn(\boldsymbol{W} \circledast \boldsymbol{x}))^T \boldsymbol{\beta} = \boldsymbol{b}^T \boldsymbol{\beta}, \tag{13}$$

where $\boldsymbol{\beta} \in \mathbb{R}^{(p \cdot m) \times 1}$ is a vectorized subset of pointwise convolution weight $\overline{\boldsymbol{W}} \in \mathbb{R}^{(p \cdot m) \times q}$ along its first dimension. Here, we simply use $\circledast$ to denote the depthwise convolution operation described in Equation (1) without specific notations in mathematic expression. Note that the result of $\boldsymbol{W} \circledast \boldsymbol{x}$ is in the shape of $\mathbb{R}^{(p \cdot m) \times 1}$; thus, we can obtain that $\boldsymbol{b} = Sgn(\boldsymbol{W} \circledast \boldsymbol{x}) \in \mathbb{B}^{(p \cdot m) \times 1}$. Therefore, we can rewrite Equation (13) as $y = \sum_{i=1}^{p \cdot m} \pm\beta_i = \pm\beta_1 \pm \beta_2 \pm \cdots \pm \beta_{p \cdot m}$. Moreover, since $\boldsymbol{\beta}$ is a vector with $p \cdot m$ trainable real value parameters (i.e., pointwise weight), we could convert the problem of approximating y with y' to the problem of optimizing $\boldsymbol{\beta}$:

$$\underset{\boldsymbol{W}, \boldsymbol{\beta}}{\text{minimize}}\ \mathcal{L}(y, y') = ||y - \boldsymbol{b}^T \boldsymbol{\beta}||. \tag{14}$$

An obvious solution to this optimization problem is to choose $\beta_i \cdot b_i$ to y while setting all the other elements in $\boldsymbol{\beta}$ to be zero. Thus, there always exists a vector $\boldsymbol{\beta}$ to make $y' = y = \boldsymbol{b}^T \boldsymbol{\beta}$.

2.5.2 Scheme-2. In this sparse binarization scheme, the weight of depthwise convolution now becomes $\boldsymbol{W} \in \mathbb{T}^{(p \cdot m) \times kh \times kw}$, where $\mathbb{T} = \{-1, 0, +1\}$. With the same notations and logical flow described in Section 2.5.1, removing the binarized activation function reformats the Equation (13) as:

$$y = (\boldsymbol{W} \circledast \boldsymbol{x})^T \boldsymbol{\beta} = \boldsymbol{b}^T \boldsymbol{\beta}, \tag{15}$$

where $\boldsymbol{\beta} \in \mathbb{T}^{(p \cdot m) \times 1}$, since the weights of the pointwise convolution layer are trained with sparse binarization. To solve the same optimization problem in Equation (14), it is relatively more difficult to find a solution $\boldsymbol{\beta}$ with minimized loss between y and y', except $\boldsymbol{W}$ is well trained.

3 EXPERIMENTS ANALYSIS

Our experiments mainly include two parts. First, we provide comprehensive comparisons with other related works and analyze the effect of hyper-parameter (i.e., channel expansion and number of channels) by using binarization scheme-1. Second, we examine the effectiveness of sparse binarization scheme-2 with structured weight sparsity.

3.1 Software Experiment Setup

The software experiments of this work are performed under the framework of pytorch, which recently optimized its depthwise convolution backend CUDA library to accelerate the training process. The depthwise convolution is a special case of grouped convolution, where the number of groups should be set to the number of input channels. We employ the stochastic gradient descent with momentum = 0.9 as the 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 is preferable. We set the initial learning rate as 0.001, which is reduced to 0.0001 through scheduling. Learning rate larger than 0.01 will cause severe fluctuation with no accuracy enhancement during the model training.

The experiments are performed on three common image datasets: MNIST, SVHN, and CIFAR-10. MNIST is a 28×28 grayscale handwritten digit (0–9) image dataset with 60K samples in training set and 10K samples in test set. The Street View House Number (SVHN) contains 32×32 RGB real-world images with 73,257 samples in training set and 26,032 samples in test set. CIFAR-10 is a colorful 32×32 image dataset that contains 10 classes of real-world objects/animals with 50K training samples and 10K test samples. The test accuracy of MNIST, SVHN, and CIFAR-10 are

Table 2. Inference Accuracy (%) of MNIST, SVHN, and CIFAR-10

	Baseline CNN	BD-Net (this work)	BinaryConnect [13]	BNN [40]	Dorefa-Net [53]	BWN [46]
MNIST	99.46	99.41	98.99	98.60	-	98.69
SVHN	94.29	93.66	97.85	97.49	97.52	97.46
CIFAR-10	91.25	92.41	91.73	89.85	-	89.49

reported in Table 2 using the deep neural network structure (BD-Net) described in Section 2.4. The model configurations for scheme-1 are:

- MNIST: 16 input channels, 5 basic blocks, 128 hidden neurons, 64 batch size, 3×3 kernel size, 4 channel expansion.
- SVHN: 128 input channels, 5 basic blocks, 512 hidden neurons, 64 batch size, 3×3 kernel size, 4 channel expansion.
- CIFAR-10: 512 input channels, 10 basic blocks, 512 hidden neurons, 20 batch size, 3×3 kernel size, 4 channel expansion.

In addition, for the sparse binarization in scheme-2, we challenge it with more compact architectures that are enumerated as:

- MNIST: 16 input channels, 3 basic blocks, 128 hidden neurons, 256 batch size, 3×3 kernel size, 2 channel expansion.
- SVHN: 64 input channels, 3 basic blocks, 256 hidden neurons, 256 batch size, 3×3 kernel size, 2 channel expansion.
- CIFAR-10: 256 input channels, 5 basic blocks, 512 hidden neurons, 256 batch size, 3×3 kernel size, 2 channel expansion.

Note that, to ensure fair comparison between sparse BD-Net and its baseline CNN counterparts, identical hyper-parameters are used for both CNN baseline and sparse BD-Net. As the results show in Table 2, sparse BD-Net using binarization scheme-1 can obtain close or even better accuracy with respect to its baseline counterpart. Furthermore, for the largest CIFAR-10 dataset, our work can achieve the best accuracy in comparison with other works applied with weight binarization techniques as tabulated in Table 2.

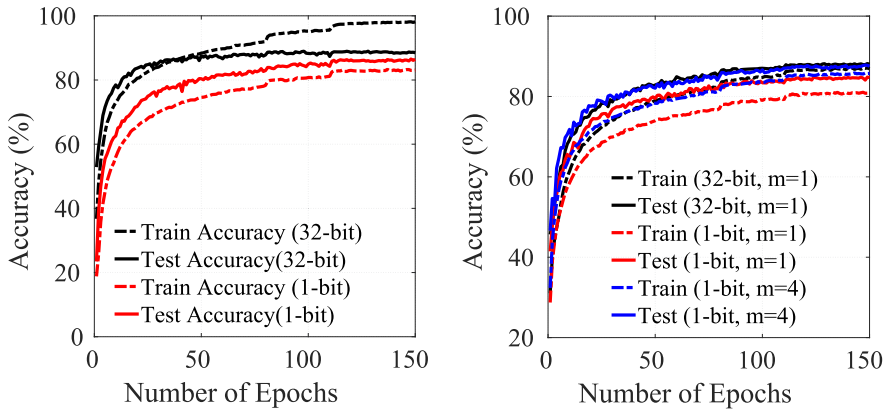
3.2 Effect of Hyper-parameters

In this subsection, we examine the effect of hyper-parameters on neural network performance. Since the neural networks are trained from scratch instead of fine-tuning from the pretrained model, we chose the CIFAR-10 as the representative experiment dataset to report the results.

3.2.1 Channel Expansion. As discussed in Section 2.1, increasing the channel expansion m enriches the intermediate feature sets, thus leading to more variant combinations as the output of convolution layer. We run trails with small input/output channel ($p = q = 64$, much smaller than that shown in Table 2) to investigate the effect of channel expansion on neural network accuracy. As the results show in Table 3, the precision denotes the number of bits used for weights of convolution kernel. Note that, for depthwise separable convolution, precision only refers to the depthwise part; the data type of pointwise weights are still real numbers. Moreover, for depthwise separable convolution with 1-bit convolution weight, the intermediate binarized activation function is included. This experiment shows that directly using depthwise separable convolution ($m = 1$) to replace spatial-convolution layer only results in slight accuracy degradation (<2%). The binarization applied on depthwise separable convolution will lead to further accuracy drop.

Table 3. The CIFAR-10 Test Accuracy with Various Configurations (i.e., Precision and Channel Expansion)

	Spatial Convolution		Depthwise Separable Convolution					
precision	32-bit	1-bit	32-bit	1-bit	1-bit	1-bit	1-bit	1-bit
m	-	-	1	1	2	4	8	16
Top-1 Test Accuracy	89.28%	86.9%	87.93 %	85.62%	86.39%	87.44%	87.8%	89.0%



(a) Baseline CNN (b) BD-Net

Fig. 3. The accuracy evolution curve of (a) Baseline CNN using normal spatial-convolution with 32-bit and 1-bit weight. (b) BD-Net using binarized depthwise separable convolution with varying channel expansion $\boldsymbol{m}$.

Table 4. Test Accuracy with Varying Number of Intermediate Channels (i.e., Input and Output Channels)

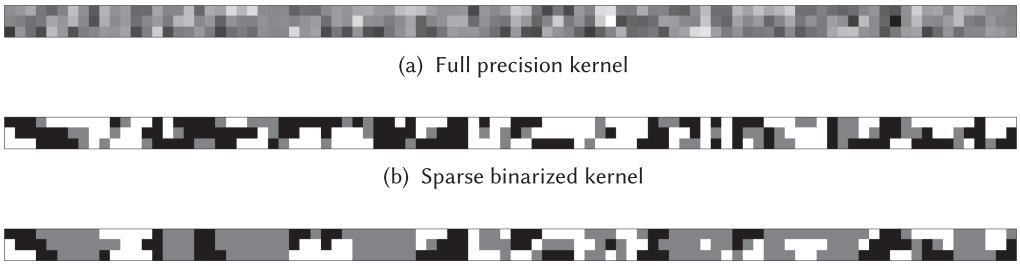
number of channels (p,q)	64	128	256	384	512
Baseline CNN	89.28%	90.47%	91.44%	91.56%	91.25%
BD-Net	87.44%	90.65%	91.76%	92.26%	92.41%

m = 4 in this experiment.

However, increasing m can effectively narrow the the accuracy gap between BD-Net and its normal convolution counterpart.

Beyond that, we include the accuracy evolution curves in Figure 3 to show that our binarized depthwise separable convolution is helpful to prevent network from over-fitting. As shown in Figure 3(a), the training curve of normal spatial convolution (black curve) shows overfitting due to over-parameterization in the convolution layer. Directly applying the weight binarization techniques on convolution weights as introduced in BinaryConnect [13] does weaken over-fitting, but they lower the test accuracy as well. On the contrary, our proposed BD-Net can achieve almost the same accuracy as baseline CNN, which avoids the over-fitting problem.

3.2.2 Number of Channels. Table 4 shows the CIFAR-10 test accuracy with various numbers of intermediate channels, which denotes the input and output channels of basic blocks in BD-Net (assuming $p = q$). It can be seen that increasing the number of intermediate channels improves accuracy. As we discussed above, the baseline CNN with normal spatial-convolution easily suffers



(a) Full precision kernel

(b) Sparse binarized kernel

(c) binarized kernel with structured sparsity pattern

Fig. 4. Kernel visualization for the first depthwise layer on MNIST dataset. For (b) and (c), patterns in {white, grey, black} denote {−1, 0, +1}, respectively.

Table 5. Weight-sparse Binarization and Structured Pruning Analysis (%) of MNIST, SVHN, and CIFAR-10

	Full precision CNN	Full precision CNN + Lasso [51]	Structured sparsity	Full precision BD-Net	Sparse BD-Net (32-bit)	Sparse BD-Net (16-bit)	Sparse BD-Net (8-bit)	Sparse BD-Net + Lasso (full precision)	Structured sparsity
MNIST	97.93	97.91	23%	98.68	98.63	98.68	98.69	98.75	18%
SVHN	90.99	90.63	35%	92.94	92.87	92.59	92.53	93.04	21%
CIFAR-10	89.35	89.45	56%	89.59	90.00	89.06	88.59	90.42	47%

D-Net denotes the depthwise-separable convolution without weight binarization. Note that we apply fixed-point activation quantization from Reference [8], with varying bit-width on sparse BD-Net. Such activation quantization is for the accelerator compatibility purpose in the following sections.

from over-fitting if the network topology is not fine-tuned. When the number of intermediate channels is larger than 128, the test accuracies of BD-Net are above the baseline CNN. In general, our BD-Net is expected to achieve even higher accuracy with further fine-tuning on the model.

We notice that reducing the number of channels lowers the computational cost and model size exponentially (referring to Table 1). Other methods, like further quantizing the intermediate tensor to a lower number of bits, only reduce the hardware cost linearly, due to the convolution being performed using `Add/Sub` instead of `MAC`. Moreover, the accuracy improvement using a large number of channels is limited. Thus, we are inclined to scale the number of intermediate channels using channel expansion first, once the hardware deployment of BD-Net meets power or memory bottleneck.

3.3 Effect of Weight Binarization with Structured Sparse Pattern

We first try to visualize the weights of first depthwise convolution in different training cases. As shown in Figure 4(b), without assistance of the lasso regularization term, the sparse binarization function in Equation (5) leads to not only small sparsity ration, but also highly irregular sparse pattern. In contrast to that, leveraging lasso-based regularization with predefined sparse pattern, Figure 4(c) shows the trained weights are in highly regular shape. It is also intriguing to notice that the nonzero weights learned in Figure 4(b) and 4(c) are very similar.

Table 5 lists the accuracy of proposed sparse BD-Net by using the binarization scheme-2 with group-lasso–based regularization and proposed weight penalty clipping techniques. Note that the weight penalty clipping technique is considered to be included by default if we simultaneously use sparse binarization and group-lasso regularization for structured sparsity pattern. It is worthwhile to note that we compare our structured pruning sparse BD-Net with full precision CNN using the pruning method in Reference [51]. The lower structured sparsity of our method is because the base CNN owns redundant model comparing with the proposed BD-Net. In addition

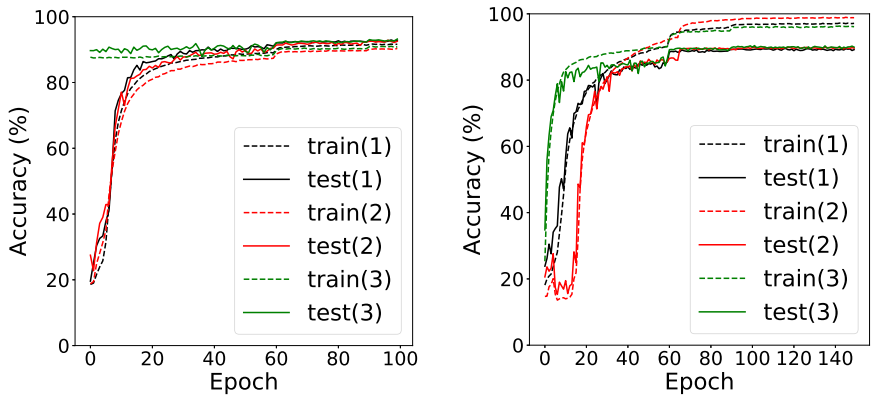


Fig. 5. The accuracy evolution curve of BD-Net on (a) SVHN and (b) CIFAR-10 datasets during training. Three different training setups are considered, which are (1) full precision (FP32) baseline, (2) sparse binarization without group-lasso regularization, and (3) sparse binarization with group-lasso regularization. The group-lasso regularization includes the proposed weight penalty clipping technique by default. For configuration (3), models are fine-tuned from FP32 baseline model on both SVHN and CIFAR10.

to the results reported in Table 5, the evolution curves of train-subset accuracy and test-set accuracy are depicted in Figure 5 with different training configurations. We do observe the training difficulties with the model compression enabled, which encourages us to fine-tune the full-precision pretrained model for both training efficiency and final converged accuracy. As listed in Table 5, after sparse binarization, the accuracy is maintained at the same level with full precision counterparts on MNIST and SVHN datasets, which is even better than the full precision baseline on CIFAR10 dataset. In addition, we further quantize the input activation from 8-bit to 32-bit by using the method proposed in Reference [8], where this step is done for the purpose of accelerator compatibility. Furthermore, introducing structured sparse weight pattern (i.e., sparse BD-Net + Lasso) achieves better accuracy in comparison to the full precision baselines with 0.18, 0.21, 0.47 structured sparsity on these three datasets, respectively. The structured sparsity is defined as the ratio of the number of weight groups with all zero values. Figure 4 intuitively shows the effectiveness of the sparse binarization with/without lasso techniques.

4 PROCESSING-IN-MEMORY ACCELERATION

In addition to the sparse binarization together with the depthwise-separable convolution, we propose a customized Processing-In-Memory (PIM)-based accelerator to further boost the inference performance of proposed sparse BD-Net. Such a platform is expected to achieve four significant objectives exploiting the well-explored Spin-Orbit Torque Magnetic Random Access Memory (SOT-MRAM) [3, 6, 7, 22, 27]: (1) Reducing the energy consumption of convolutional layers through utilizing efficient *add*/*sub*-based computing after binarization/sparse-binarization (i.e., scheme-1 and scheme-2); (2) Reducing the memory (i.e., DNN 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 presented CNN In-Memory Accelerator is shown in Figure 6(a) 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 the Control (Ctrl) unit to process entire BD-Net implemented by both scheme-1 and scheme-2. Assume the

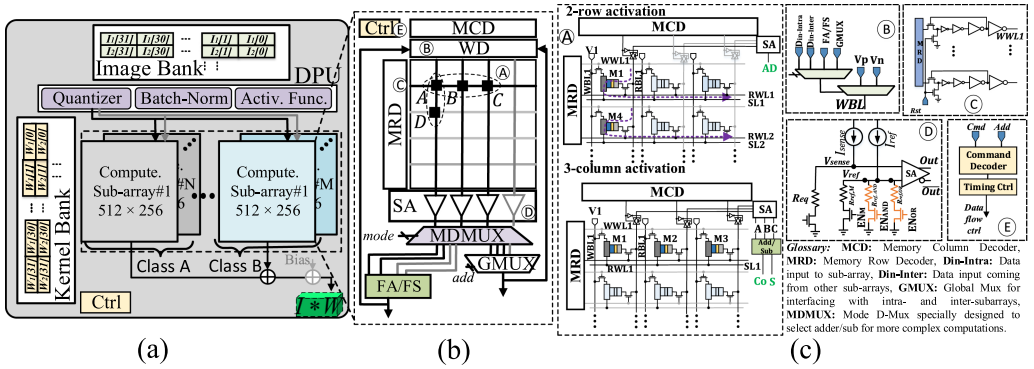


Fig. 6. (a) Overview of the presented PIM architecture, (b) Block-level sub-array architecture, (c) 2-row and 3-column activation mechanisms and functional blocks.

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 that are designed to handle the computational load 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 Figure 6(a)) and then results are mapped to the parallel sub-arrays.

4.1 PIM Sub-array

Figure 6(b) shows the presented PIM sub-array architecture that is accordingly implemented by SOT-MRAM in Figure 6(c) (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 Full-Adder/Subtractor (FA/FS) unit and can be adjusted by Ctrl unit (E) to work in dual mode that performs both memory `write`/`read` and in-memory logic operations. The SOT-MRAM cell adopted here is a composite structure of Spin Hall Metal (SHM) and Magnetic Tunnel Junction (MTJ), which is known for its better energy efficiency and speed for memory `write` in comparison to STT-MRAM. 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 the following operations:

(1) Memory Write/Read: To *write* a data bit in any of the SOT-MRAM cells (e.g., M1 in Figure 6(c) (A)), write current should be injected through the SHM (Tungsten, $\beta - W$ [44]) of SOT-MRAM. Therefore, WWL1 should be activated by the MRD where SL1 is grounded. Now, 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 computational sub-array performs the computation between in-memory operands using two distinct mechanisms referred to as 2-row activation and 3-column activation. The main ideas behind developing 2-row and 3-column activation mechanisms are to perform bulk bit-wise in-memory AND operation and in-memory addition/subtraction, respectively.

In the 2-row activation mechanism, every two bits stored in the same column can be selected and sensed simultaneously employing modified MRD [36], as depicted in Figure 6(c) (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, consider the data organization shown in Figure 6(b) where A and D operands correspond to M1 and M4 memory cells in Figure 6(c) (A), respectively, 2-row activation mechanism generates AD after SA. To validate the variation tolerance of sense circuit, we have performed Monte-Carlo simulation with 100K trials. A $\sigma = 5\%$ variation is added on the Resistance-Area product (RA_P), and a $\sigma = 10\%$ process variation is added on the TMR. The simulation result of (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 mechanism, we have devised a Mode demultiplexer (MDMUX) right after SAs to switch between memory mode and this new computing mechanism. As can be seen in block-level sub-array architecture (Figure 6(b)), 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 operand write-back in conventional in-memory adder designs as well as accelerating in-memory processing. For this computation mechanism, 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 Figure 6(b)) correspond to M1, M2, and M3 memory cells in Figure 6(c) (A), respectively, the 3-column activation yields $Sum(/Difference)$ and $Carry(/Borrow)$ bits.

4.2 In-memory Binary-weight Convolver

The presented PIM accelerator offers in-memory binary-weight convolution with and without sparsity, and in-memory Bit-Wise Convolver to handle main operations of the sparse BD-Net based on the in-memory computing mechanisms. From a hardware-implementation perspective, there are two types of convolution operations in sparse BD-Net that need to be taken into account. The first one is binary and sparse binary convolution located in basic block with binarized, sparse 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 convolution schemes of the PIM accelerator, we still show two different convolution methods to boost the accelerator's performance with smaller area overhead. As depicted in Figure 6(a), two classes of sub-arrays (A and B) are, respectively, allocated to in-memory binary-weight with and without sparsity and bit-wise convolvers. Note that all the computational sub-arrays support both memory and computing modes and only differ from required add-on hardware that will be discussed in the following.

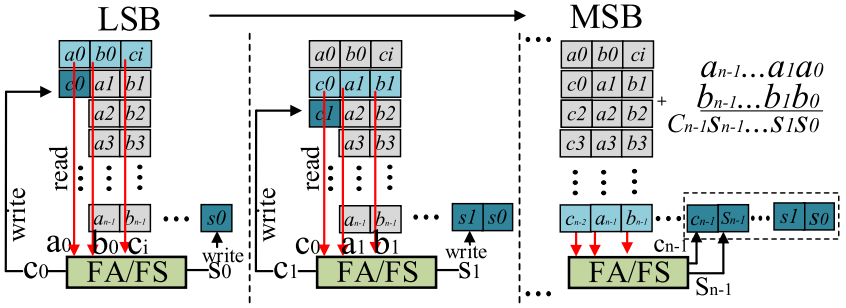


Fig. 7. Realization of n-bit in-memory addition in the presented PIM platform.

The ratio between A and B is specially determined considering the network specifications and performance constraints.

As the main operations of the sparse BD-Net, depthwise and pointwise convolutions are the most critical units of the accelerator, as they are responsible for the most iterative block that takes up the vast majority of the run-time in the sparse BD-Net. These units must keep high throughput and resource efficiency while handling different input widths at run-time. The main operation in this block is *add*/*sub*. So, here we present an in-memory binary-weight convolver with and without sparsity based on 3-column activation mechanism to handle multi-bit *add*/*sub* operations of BD-Net scheme-1/scheme-2. While there are few designs for in-memory adder/subtractor in the literature [4, 5], this work is the first showing a fast (2-cycle) and parallelable in-memory *add*/*sub* operation. The 3-column activation mechanism 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 the next computing round. This can be fulfilled using the modified WD and MRD in one cycle (memory write).

We use the data organization depicted in Figure 7 as the main mapping method to perform an n-bit *add*/*sub* operation within computational sub-arrays. As shown, operands ($a_{n-1}...a_1a_0$ and $b_{n-1}...b_1b_0$) are initially loaded and organized into different memory rows such that one computation can be performed per memory cycle. Here in Figure 7 L.H.S., LSBs of two operands along with carry-in (ci) are selected and processed to generate Carry-out ($c0$) and Sum ($s0$). After the computation, the results need to be stored in the sub-array to be prepared for the next computing round. This can be fulfilled using the modified WD and MRD in one cycle (memory write). The same process continues towards the MSBs computation (Figure 7 R.H.S.). Therefore, each computational sub-array implements an n-bit *add*/*sub* operation in $2 \times n$ cycles (n read + n write). Finally, $c_{n-1}s_{n-1}...s_1s_0$ is produced and stored in designated computational sub-array.

Leveraging the presented idea in multiple PIM sub-arrays can provide a parallel computation scheme. Figure 8(a) shows the requisite data organization and computation of depthwise and pointwise convolutional layers. Initially, c 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} or {−1,0,+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 in Figure 8(a)), each combined batch's channel (Ch) can be processed using four parallel sub-arrays. Here, Ch-1 to Ch-4 are mapped to mat-1 to mat-4, respectively. After mapping, the parallel activated sub-arrays of PIM platform operate to produce the output feature maps leveraging the same *add*/*sub* method shown in Figure 8(b).

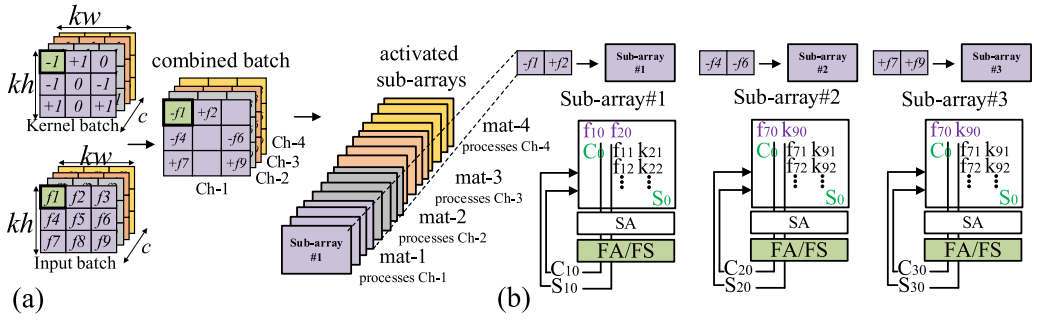


Fig. 8. (a) Combining step, where input batch is processed according to the binary or sparse binary kernel batch to produce a combined batch, (b) Parallel computing step in the presented PIM platform to perform in-memory addition in different mats.

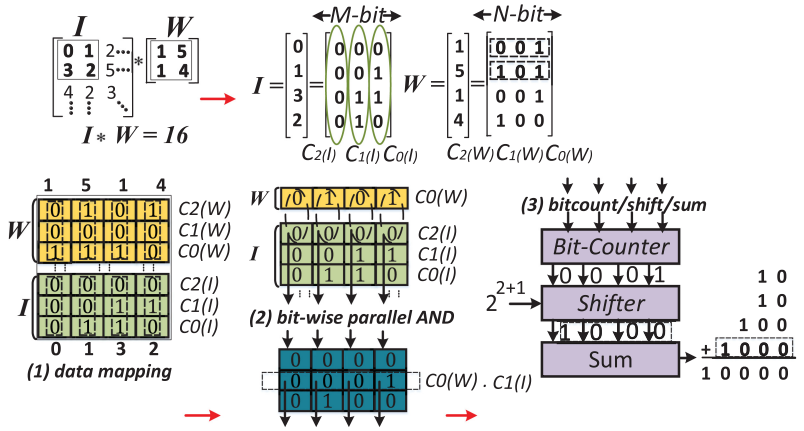


Fig. 9. In-memory bit-wise convolver. The presented computation mechanism can be easily extended to 32-bit fixed-point operands.

4.3 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 [53]. Thus, the rest of the layers could be implemented all by convolution computation by exploiting *logic AND*, *bitcount*, and *bitshift* as rapid and parallelizable operations [6, 53]. The presented computation mechanism can be easily extended to 32-bit fixed-point operands. Assume I is a sequence of M-bit input integers (3-bit as an example in Figure 9) 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 $m = [0, M-1]$. Accordingly, we represent a second sequence denoted as $C_m(I)$ including the combination of mth 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 mth 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 follows:

$$I * W = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} 2^{m+n} bitcount(and(C_n(W), C_m(I))). \quad (16)$$

As shown in data mapping step in Figure 9, $C_2(W)$-$C_0(W)$ 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 Figure 9. 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 Figure 9, "0001," as a 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 2-row activation method in entire memory sub-arrays like the one in Reference [6], we still need to incorporate CMOS bit-counter and shifter units in some of computational sub-arrays (Class B). The 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.

4.4 Hardware Setup

In the following, we explain the hardware and network setups for the presented PIM platform running BD-Net scheme-2 compared with state-of-the-art inference acceleration solutions, i.e., DRAM, ReRAM, ASIC, and GPU running a baseline CNN.

4.4.1 Accelerators' Setup. ***MRAM*** [4]***:*** We set up the presented PIM's sub-array organization with 256 rows and 512 columns in a 512 Mb total memory capacity. The ratio of computational sub-array (class A: class B) is obtained: 7:2. A comprehensive device-to-architecture evaluation framework along with two in-house simulators are developed to assess the performance of our PIM platform. We first use the Non-Equilibrium Green's Function (NEGF) jointly with Landau-Lifshitz-Gilbert (LLG) with spin Hall effect equations to model SOT-MRAM bitcell [16, 21] at device level. Then, we develop a Verilog-A model of 2T1R SOT-MRAM bit cell at circuit level to co-simulate with the interface CMOS circuits in Cadence Spectre and SPICE with 45 nm North Carolina State University (NCSU) Product Development Kit (PDK) library [1]. We then develop an architectural-level simulator on top of NVSim [14]. The controllers and add-on circuits are synthesized by Design Compiler [49] with an industry library. In addition, a behavioral-level simulator is developed in Matlab, calculating the latency and energy that the platform spends considering a particular network configuration. ***DRAM:*** We developed a DRISA-like [35]-1T1C accelerator for low bit-width CNNs. To do so, the DRAM cell configuration is extracted from Rambus simulator [15] and applied in our circuit-level simulation. Accordingly, we modifed the Cacti [10] to obtain system-level results. ***ReRAM:*** A Prime-like [11] accelerator with two full functional sub-arrays and one buffer sub-array per bank (totally 64 sub-arrays) are considered for assessment. For simulation, we jointly use NVSim [14] and MNSim [52] to achieve the memory-level and network-level results. ***ASIC:*** We developed a YodaNN-like [2] ASIC accelerator with 64 tiles. We synthesized the platform with Design Compiler [49] with 45 nm technology. The eDRAM and SRAM performances were estimated with CACTI [42]. ***GPU:*** We used the NVIDIA GTX 1080Ti Pascal GPU. It has 3,584

[4]MRAM and processing-in-MRAM platform's terms are used alternatively in the manuscript. This will also apply to other PIM platforms, e.g., DRAM stands for processing-in-DRAM platform.

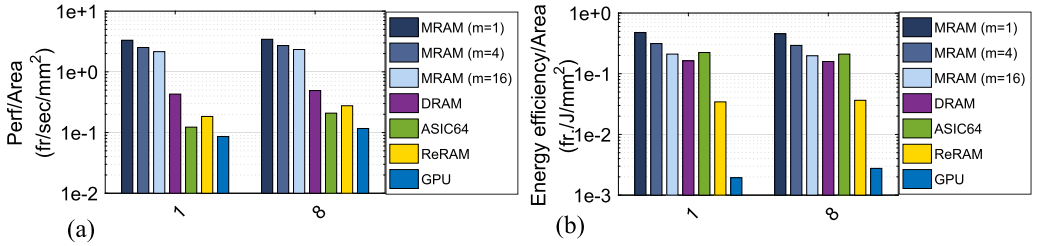


Fig. 10. (a) Performance and (b) Energy-efficiency of different platforms normalized by area (Y-axis=Log scale).

CUDA cores running at 1.5 GHz (11 TFLOPs peak performance). The energy consumption was measured with NVIDIA's system management interface. Similar to Reference [35], we scaled the achieved results by 50% to exclude the energy consumed by cooling, and so on.

4.4.2 Modeling Setup. We consider BD-Net with distinct channel expansion configurations (m= 1, 4, and 16) for evaluation. To have a fair comparison, we use a particular bit-width for weight and activation <W:A> (<1:32>) for DRAM-, ReRAM, ASIC-, and GPU-based acceleration methods. The SVHN dataset [43] is selected for evaluation. The images are re-sized to 32×32 and fed to the models. A CNN with 1 inception block, 3 basic blocks, 1 average layer, and 2 MLP layers is adopted to be accelerated using DRAM, ReRAM, ASIC, and GPU platforms. MLP layers are equivalently implemented by convolutions.

4.5 Experiment Results

In this subsection, we evaluate and report performance, energy-efficiency, and resource utilization ratio for different platforms. To make a fair comparison with existing platforms, we normalized the energy and performance results to area based on the DRISA [35] method.

4.5.1 Performance. Figure 10(a) compares the processing-in-MRAM throughput (frames per second) results with three configurations with different accelerators. Based on the results, MRAM with $m = 1$ shows the highest performance compared to other designs. As for $m = 4$, the MRAM accelerator is 25.7× and 15.8× faster on average than GPU and ASIC-64 solutions, respectively. This efficiency can be related to parallel and ultra-fast in-memory operations of MRAM compared to multi-cycle ASIC and GPU operations as well as the potential mismatch between data movement and computation in Von-Neumann computing methods. Additionally, the MRAM solution is 11.3× faster than the ReRAM method. It is worth pointing out that ReRAM accelerators suffer matrix splitting owing to intrinsically limited bit levels of ReRAM device, thus more sub-arrays need to be occupied. This can further limit parallelism methods. Additionally, a ReRAM crossbar imposes a large peripheral circuit overhead due to existing DAC/ADC and buffers occupying roughly 85% of area [6, 11].

4.5.2 Energy Efficiency. Figure 10(b) shows the MRAM's energy-efficiency results (frames per joule) on BD-Net scheme-2 compared with 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 $m = 4$ to compare with the other platforms. As shown, MRAM solution offers the highest energy-efficiency normalized to area compared to others owing to its energy-efficient and parallel operations. We observe that MRAM's solution is 129× more energy-efficient than the GPU solution. As compared with DRAM and ASIC platforms, MRAM achieves 1.8× and 1.3× energy saving, respectively. While the DRAM platform shows the least area (due to

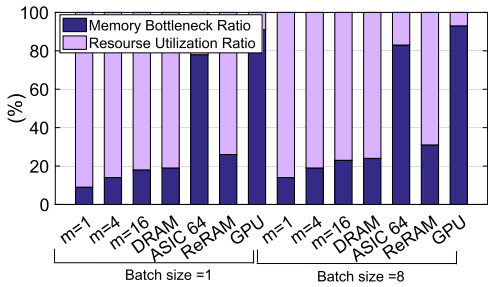


Fig. 11. Estimation of Resource Utilization Ratio for different platforms, experimentally extracted considering the number of memory access for each platform.

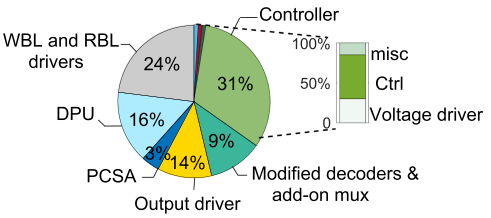


Fig. 12. Area overhead of the presented processing-in-memory platform in a memory group.

DRISA's 1T1C method [35] with unmodified memory SA), it suffers large refresh power of DRAM-based PIM accelerators [47]. Besides, it is dealing with a destructive *data-overwritten* issue due to the charge-sharing characteristic of capacitors. It means that the result of computation will ultimately overwrite the operands. To solve this issue in the context of DRAM, *multi-cycle operations* are set forth, which has further degraded PIM performance. Figure 10(b) also shows that MRAM obtains ~ 8.5× saving in energy compared with the ReRAM solution. Generally, 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 sparse binarization.

4.5.3 Resource Utilization. We estimated the time fraction at which the computation has to wait for data; and on-/off-chip data transfer limits the performance, referred to as memory bottleneck ratio for different platforms, as depicted in Figure 11 considering batch sizes 1 and 8. This evaluation is done through the peak performance and experimentally extracted results for each platform considering number of memory access. We observe that processing-in-memory solutions, i.e., MRAM (m = 1,4,16), DRAM, and ReRAM, spend less than 35% time for data transfer and memory access. But, ASIC and GPU, as Von-Neumann computing platforms, spend over 70% and 90% of time, respectively, waiting for the loading data from the memory. In this way, we estimate the resource utilization ratio for different platforms. We observe that SOT-MRAM platform with different BD-Net configuration achieves the highest ratio by efficiently utilizing one average 72% of its computation resources. It can be seen that the smaller the m is, the higher resource utilization to memory bottleneck ratio is obtained for the presented accelerator. It is worth pointing out that the GPU has utilized only ~5% of its resources to perform the similar task.

4.5.4 Area Overhead. To assess the area overhead of presented processing-MRAM platform on top of MRAM chip, several hardware cost sources must be taken into consideration, as broken down in Figure 12. This includes DPU, the add-on transistors to SAs; in our design, each SA requires two additional transistors connected to each *BL* (Figure 6(c)) to enable in-memory computing; the modified MRD overhead; we modify each *WL* driver by adding two more transistors in the typical buffer chain based on the method used in Reference [36]; the ctrl's overhead to control enable bits; ctrl generates the activation bits with MUX units with 6 transistors, where we observe that the modified controller and drivers contribute more than 50% of this area overhead in a memory group. To sum it up, our accelerator imposes 2.8% area overhead to the original memory die.

5 SUMMARY AND FUTURE WORK

In this work, we have shown a multiplication-less deep convolution neural network that replaces the normal spatial convolution layer with binarized depthwise separable convolution. Such a neural network compression technique significantly reduces the hardware utilization in terms of both computation and memory.

Compared to the accuracy degradation caused by the binarization of normal spatial-convolution using the techniques from BinaryConnect [13] and BWN [46], our method can compensate such accuracy loss through tuning the internal hyper-parameters (i.e., channel expansion) effectively. In this work, the convolution kernel (i.e., feature filter) and the generated feature maps are binarized. According to the experiments performed in this work, a hypothesis can be drawn as guidance for further experiment direction. For searching the correlation between local pixels and the captured feature maps, high resolution is not essential. However, the new representation generated at the output of convolution layer is expected to be higher in precision to avoid the information loss in the forward path during the inference. Our future work will try to apply other neural network compression techniques, such as quantization and pruning, on the pointwise convolution to further compress the convolution layers. Moreover, we will try to alternate the normal convolution layers of other famous network structures with our binarized depthwise separable convolution and examine the model performance. Besides, we presented a processing-in-MRAM accelerator to further accelerate BD-Net. Our simulation results showed that, with almost the same accuracy to the baseline, our PIM design obtains 129× better energy-efficiency and ~25.7× speedup compared to the GPU platform.

REFERENCES

[1] 2011. NCSU EDA FreePDK45. Retrieved from: http://www.eda.ncsu.edu/wiki/FreePDK45:Contents.

[2] Renzo Andri et al. 2016. YodaNN: An ultra-low power convolutional neural network accelerator based on binary weights. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'16)*. IEEE, 236–241.

[3] Shaahin Angizi, Zhezhi He, and Deliang Fan. 2018. DIMA: A depthwise CNN in-memory accelerator. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'18)*. IEEE, 1–8.

[4] Shaahin Angizi, Zhezhi He, and Deliang Fan. 2018. PIMA-logic: A novel processing-in-memory architecture for highly flexible and energy-efficient logic computation. In *Proceedings of the 55th Design Automation Conference*. ACM, 162.

[5] Shaahin Angizi, Zhezhi He, Farhana Parveen, and Deliang Fan. 2017. RIMPA: A new reconfigurable dual-mode in-memory processing architecture with spin Hall effect-driven domain wall motion device. In *Proceedings of the IEEE Computer Society Symposium on VLSI (ISVLSI'17)*. IEEE, 45–50.

[6] Shaahin Angizi, Zhezhi He, Farhana Parveen, and Deliang Fan. 2018. IMCE: Energy-efficient bit-wise in-memory convolution engine for deep neural network. In *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*. IEEE Press, 111–116.

[7] Shaahin Angizi, Jiao Sun, Wei Zhang, and Deliang Fan. 2019. AlignS: A processing-in-memory accelerator for DNA short read alignment leveraging SOT-MRAM. In *Proceedings of the 56th Design Automation Conference*. ACM, 144.

[8] Ron Banner, Itay Hubara, Elad Hoffer, and Daniel Soudry. 2018. Scalable methods for 8-bit training of neural networks. In *Proceedings of the Advances in Neural Information Processing Systems Conference*. 5145–5153.

[9] Yoshua Bengio et al. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv:1308.3432* (2013).

[10] Ke Chen et al. 2012. CACTI-3DD: Architecture-level modeling for 3D die-stacked DRAM main memory. In *Proceedings of the Design, Automation, and Test in Europe Conference (DATE'12)*. IEEE, 33–38.

[11] Ping Chi et al. 2016. PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory. In *Proceedings of the International Symposium on Computer Architecture (ISCA'16)*. IEEE Press.

[12] François Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1251–1258.

[13] Matthieu Courbariaux et al. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Proceedings of the Advances in Neural Information Processing Systems Conference*. 3123–3131.

[14] Xiangyu Dong et al. 2014. NVSim: A circuit-level performance, energy, and area model for emerging non-volatile memory. In *Emerging Memory Technologies*. Springer, 15–50.

[15] Thomas Vogelsang. 2010. Understanding the energy consumption of dynamic random access memories. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society.

[16] Xuanyao Fong, Sumeet K. Gupta et al. 2011. KNACK: A hybrid spin-charge mixed-mode simulator for evaluating different genres of spin-transfer torque MRAM bit-cells. In *Proceedings of the International Conference on Simulation of Semiconductor Processes and Devices (SISPAD'11)*. IEEE, 51–54.

[17] Song Han et al. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization, and Huffman coding. In *Proceedings of the International Conference on Learning Representations (ICLR'15)*.

[18] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Proceedings of the Advances in Neural Information Processing Systems Conference*. 1135–1143.

[19] Kaiming He et al. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Vision and Pattern Recognition (CVPR'16)*. 770–778.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*. 1026–1034.

[21] Zhezhi He et al. 2017. High performance and energy-efficient in-memory computing architecture based on SOT-MRAM. In *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH'17)*. IEEE, 97–102.

[22] Zhezhi He, Shaahin Angizi, and Deliang Fan. 2017. Exploring STT-MRAM based in-memory computing paradigm with application of image edge extraction. In *Proceedings of the IEEE International Conference on Computer Design (ICCD'17)*. IEEE, 439–446.

[23] Zhezhi He, Shaahin Angizi, and Deliang Fan. 2018. Accelerating low bit-width deep convolution neural network in MRAM. In *Proceedings of the IEEE Computer Society Symposium on VLSI (ISVLSI'18)*. IEEE, 533–538.

[24] Zhezhi He, Shaahin Angizi, Adnan Siraj Rakin, and Deliang Fan. 2018. BD-NET: A multiplication-less DNN with binarized depthwise separable convolution. In *Proceedings of the IEEE Computer Society Symposium on VLSI (ISVLSI'18)*. IEEE, 130–135.

[25] Zhezhi He and Deliang Fan. 2019. 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*.

[26] Zhezhi He, Boqing Gong, and Deliang Fan. 2019. Optimize deep convolutional neural network with ternarized weights and high accuracy. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV'19)*. IEEE, 913–921.

[27] Zhezhi He, Yang Zhang, Shaahin Angizi, Boqing Gong, and Deliang Fan. 2018. Exploring a SOT-MRAM based in-memory computing for data processing. *IEEE Trans. Multi-Scale Comput. Syst.* 4, 4 (2018), 676–685.

[28] Andrew G. Howard et al. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint:1704.04861* (2017).

[29] Itay Hubara et al. 2016. Binarized neural networks. In *Proceedings of the Advances in Neural Information Processing Systems Conference*. 4107–4115.

[30] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning*. 448–456.

[31] Felix Juefei-Xu et al. 2016. Local binary convolutional neural networks. *arXiv preprint arXiv:1608.06049* (2016).

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Proceedings of the Advances in Neural Information Processing Systems Conference*. 1097–1105.

[33] Vadim Lebedev and Victor Lempitsky. 2016. Fast ConvNets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2554–2564.

[34] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[35] Shuangchen Li et al. 2017. DRISA: A DRAM-based reconfigurable in-situ accelerator. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO'17)*. ACM, 288–301.

[36] Shuangchen Li, Cong Xu et al. 2016. Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories. In *Proceedings of the Design Automation Conference (DAC'16)*. IEEE.

[37] Shiyu Liang and R. Srikant. 2017. Why deep neural networks for function approximation? In *Proceedings of the International Conference on Learning Representations (ICLR'17)*.

[38] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. 2015. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 806–814.

[39] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV'18)*. 19–34.

[40] Courbariaux Matthieu et al.2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1. *arXiv:1602.02830* (2016).

[41] Asit Mishra, Eriko Nurvitadhi, Jeffrey J. Cook, and Debbie Marr. 2018. WRPN: Wide reduced-precision networks. In *Proceedings of the International Conference on Learning Representations (ICLR'18)*.
[42] Naveen Muralimanohar et al. 2009. CACTI 6.0: A tool to model large caches. *HP Laboratories* (2009), 22–31.
[43] Yuval Netzer et al. 2011. Reading digits in natural images with unsupervised feature learning. In *Proceedings of the NIPS Workshop*, Vol. 2011. 5.
[44] Chi-Feng Pai et al. 2012. Spin transfer torque devices utilizing the giant spin Hall effect of tungsten. *Appl. Phys. Lett.* 101, 12 (2012), 122404.
[45] Wei Pan, Xiaofan Lin, and Cong Zhao. 2017. Towards accurate binary convolutional neural network. In *Proceedings of the Advances in Neural Information Processing Systems Conference*. 344–352.
[46] Mohammad Rastegari et al. 2016. XNORr-Net: Imagenet classification using binary convolutional neural networks. In *Proceedings of the European Conference on Computer Vision*. Springer, 525–542.
[47] Vivek Seshadri et al. 2017. Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO'17)*. ACM, 273–287.
[48] Laurent Sifre and Stéphane Mallat. 2014. *Rigid-motion Scattering for Image Classification*. Ph.D. Dissertation. Citeseer.
[49] Synopsys Design Compiler. Product Version 14.9.2014. Synopsys, Inc. https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-nxt.html.
[50] Gregor Urban, Krzysztof J. Geras, Samira Ebrahimi Kahou, Ozlem Aslan, Shengjie Wang, Rich Caruana, Abdelrahman Mohamed, Matthai Philipose, and Matt Richardson. 2016. Do deep convolutional nets really need to be deep and convolutional?*arXiv preprint arXiv:1603.05691* (2016).
[51] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Proceedings of the Advances in Neural Information Processing Systems Conference*. 2074–2082.
[52] Lixue Xia, Boxun Li, Tianqi Tang, Peng Gu, Pai-Yu Chen, Shimeng Yu, Yu Cao, Yu Wang, Yuan Xie, and Huazhong Yang. 2017. MNSIM: Simulation platform for memristor-based neuromorphic computing system. *IEEE Trans. Comput.-Aided Des. Integ. Circ. Syst.* 37, 5 (2017), 1009–1022.
[53] Shuchang Zhou et al. 2016. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint:1606.06160* (2016).
[54] Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*.
[55] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8697–8710.

Received May 2019; revised September 2019; accepted October 2019