2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)

NAND-Net: Minimizing Computational Complexity of In-Memory Processing for Binary Neural Networks

Hyeonuk Kim, Jaehyeong Sim, Yeongjae Choi, Lee-Sup Kim
School of Electrical Engineering
Korea Advanced Institute of Science and Technology (KAIST)
{*hukim, jhsim, yjchoi*}*@mvlsi.kaist.ac.kr, lskim@ee.kaist.ac.kr*

***Abstract*—Popular deep learning technologies suffer from memory bottlenecks, which significantly degrade the energy-efficiency, especially in mobile environments. In-memory processing for binary neural networks (BNNs) has emerged as a promising solution to mitigate such bottlenecks, and various relevant works have been presented accordingly. However, their performances are severely limited by the overheads induced by the modification of the conventional memory architectures. To alleviate the performance degradation, we propose NAND-Net, an efficient architecture to minimize the computational complexity of in-memory processing for BNNs. Based on the observation that BNNs contain many redundancies, we decomposed each convolution into sub-convolutions and eliminated the unnecessary operations. In the remaining convolution, each binary multiplication (bitwise XNOR) is replaced by a bitwise NAND operation, which can be implemented without any bit cell modifications. This NAND operation further brings an opportunity to simplify the subsequent binary accumulations (popcounts). We reduced the operation cost of those popcounts by exploiting the data patterns of the NAND outputs. Compared to the prior state-of-the-art designs, NAND-Net achieves 1.04-2.4x speedup and 34-59% energy saving, thus making it a suitable solution to implement efficient in-memory processing for BNNs.**

I. Introduction

In recent years, deep learning technologies represented by Convolutional Neural Networks (CNNs) [1] have raised the level of the artificial intelligence (AI) further by achieving unprecedented performances in a variety of practical vision applications [2]–[5]. These CNN-based algorithms are replacing the conventional vision techniques utilizing hand-crafted features [6] in most of the complex tasks requiring high performances. Accordingly, various embedded processors [7]–[11] have been implemented to accelerate CNN, and some of them are equipped to the commercially available products such as smartphones [12]. The number of such mobile devices embedded with a CNN processor is growing exponentially as the coverage of deep learning algorithms expands and their state-of-the-art accuracies [13] are constantly updated.

However, the layer depth of CNN deepens as user applications become more complicated, resulting in an increased number of operations and enormous parameter size. Massive intermediate data are produced during CNN execution, and their movements between computing core and on-chip buffer suffer from a limited internal bandwidth and huge energy dissipations. Costly off-chip memory accesses also occur due to the limited on-chip buffer size. These memory bottlenecks make accelerating CNN resource-intensive and energy-hungry. To make matters worse, current computing trend towards on-device processing due to the latency and security issues of cloud computing [14], [15] makes the bottlenecks more critical since the hardware resources of mobile devices are extremely scarce compared to those of data centers.

Various prior works have been proposed to alleviate these hardware burdens. The two most promising solutions among them are the network binarization [16]–[18] and in-memory processing [19]–[22]. The network binarization constrains all the weights and activations to either $+1$ or -1, so it achieves a significant reduction of memory footprints and computational requirements. In the resulting binary neural network (BNN), each convolution is processed by simple bitwise operations which are bitwise XNORs and popcounts. Although some marginal accuracy degradations are inevitable, the amount of the decline is reduced to a negligible level by a series of relevant works [17], [18].

In-memory processing is one of the emerging techniques to address the memory wall problem of the conventional von Neumann architectures [23]. By merging the computation core and the memory component into a single unit, it eliminates all the data movements between them, which require two orders of magnitude more energy and latency compared to the intra-core movements [24]. Although implementing computation logic in the memory is not a trivial problem, the low logic complexity of BNN makes it feasible to design an in-memory accelerator for BNNs.

There are a couple of previous implementations targeting the in-memory processing for BNNs in SRAM and DRAM. Although they made considerable improvements in reducing the bottlenecks mentioned above, their approaches contain several inefficiencies when implementing the bitwise XNOR and popcount operations in the conventional memory architectures.

For SRAM, Liu et al. [25] expanded the conventional 6T cells into their custom 8T cells to support bitwise XNOR in a

2378-203X/19/$31.00 ©2019 IEEE
DOI 10.1109/HPCA.2019.00017

single cycle. Sense amplifiers are also replaced by ADCs to produce non-binary outputs. Although these modifications enable in-memory computations, the cost of them induces significant overheads which include the lowered cell density and the increased energy consumption.

Regarding DRAM, XNOR-POP [21] placed XNOR gates near global sense amplifiers and connected them to the popcount engines on the logic die by TSVs. Ambit [26] and DRISA [22] implemented primitive logic operations such as bitwise AND and NOR in the cell array by exploiting the charge sharing effect at the cost of some architectural modifications. Bitwise XNOR and popcount operations are performed by combining the implemented primitive logic operations. However, processing each of them in this way takes more than a single cycle, so the overall throughput becomes severely degraded. All the three works quite modified the conventional DRAM architecture which is highly optimized for high density and low leakage during the fabrication process. Therefore, significant performance degradations are inevitable when processing large-scale BNN models.

To sum up, the prior works related to in-memory processing for BNNs require considerable modifications of the conventional memory architecture, resulting in substantial overheads to the memory performance including cell density, latency, and energy-efficiency. We recognize that the necessity of such modifications lies on naïve mapping of the BNN operations which are not suitable for the conventional memory architecture. These operations have to be optimized to reduce the performance overheads imposed on the memory.

In this paper, we propose NAND-Net, an efficient architecture which minimizes the computational complexity of in-memory processing for BNNs, to effectively address the issues stated above. We first observed the patterns of all the binary convolutions in each layer of the network. Based on the observation that they contain many redundancies which can be skipped by pre-computing, we decomposed each binary convolution into four sub-convolutions, which is inspired by the kernel decomposition scheme proposed by [27]. Three of the sub-convolutions can be neglected by exploiting the redundancy, and only the other is the main part which requires full runtime processing. In this main sub-convolution, the original XNOR-based multiplication is converted to a simple bitwise NAND operation which can be efficiently implemented in the memory without modifying bit cells. This NAND-based multiplication also can be converted to other primitive logic operations such as AND and NOR to reduce the design overheads in various memories. Furthermore, statistical properties on the data patterns of the NAND outputs bring an opportunity to optimize the subsequent popcounts. By exploiting them, the operation cost of the popcounts in terms of energy and latency can be reduced by more than half.

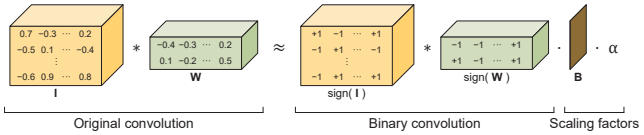


Figure 1. A binarized convolution in BNN [17]

The key features of this paper are summarized as follows.

- We propose NAND-Net, an efficient architecture to minimize the computational complexity of in-memory processing for BNNs. XNOR-based multiplications are converted to NAND-based ones which do not require any bit cell modifications.
- Based on the data patterns of the NAND outputs, popcount compression techniques are proposed to reduce the operation cost of in-memory popcounts.
- We implemented several in-memory processing designs for NAND-Net to verify the proposed architecture. We evaluated our designs with the prior state-of-the-art works.

The rest of this paper is organized as follows. Section 2 introduces the backgrounds of the fundamental concepts of this paper. Section 3 describes our proposed architecture to address the problems stated in Section 1. Section 4 shows the hardware implementations of the proposed NAND-Net. Section 5 shows the experimental results demonstrating the efficiency of the proposed techniques. Section 6 discusses the related works, and Section 7 concludes this paper.

II. Background

A. Binary Neural Network

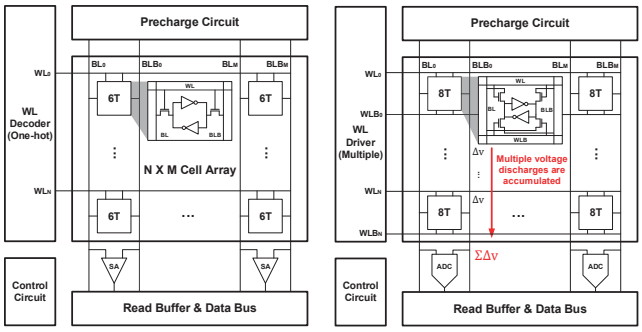
The binarization method for BNN in this paper is based on XNOR-Net [17]. For a convolutional layer whose input channel is c_{in}, the original convolution of feature map $\mathbf{I}$ and kernel $\mathbf{W}$ in Fig. 1 is binarized as follows:

$$\mathbf{I} * \mathbf{W} \approx (sign(\mathbf{I}) * sign(\mathbf{W})) \cdot \mathbf{B}\alpha \tag{1}$$

where $\mathbf{I} \in \mathbb{R}^{w_{in} \times h_{in} \times c_{in}}$ is the input feature map whose width and height are w_{in} and h_{in}, $\mathbf{W} \in \mathbb{R}^{k \times k \times c_{in}}$ is the original kernel whose size is k, $\alpha \in \mathbb{R}$ is the scaling factor for kernels, and $\mathbf{B} \in \mathbb{R}^{w_{out} \times h_{out}}$ is the scaling factor for feature maps in which w_{out} and h_{out} are the width and height of the output feature map.

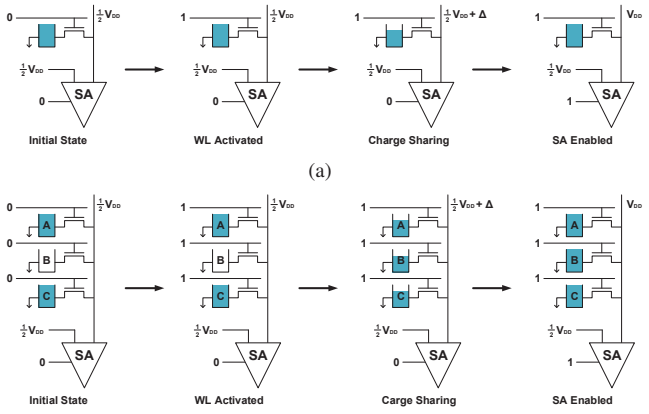
In general, α is merged into the subsequent normalization layer, and $\mathbf{B}$ is discarded since its effect on the network accuracy is negligible (e.g., less than 1% top-1 accuracy) [17]. Hence, they are omitted from the current convolutional layer.

The remaining part, convolving $sign(\mathbf{I})$ with $sign(\mathbf{W})$, is the main part of the binarized convolution. For efficient hardware mapping, $+1$ and -1 are encoded by 1 and 0, respectively. Then, all the multiply-and-accumulate operations are converted to XNOR-and-popcount operations, resulting in a considerable reduction of the computational



(a) (b)

Figure 2. SRAM architectures: (a) The conventional 6T SRAM, (b) In-memory processing with custom 8T cells in [25]



(a)

(b)

Figure 3. DRAM operations: (a) Read operation in the conventional 1T1C DRAM, (b) In-DRAM bitwise OR operation in [26]

requirements. The output difference due to encoding of -1 by 0 is corrected by (2) [28]:

$$O_{+1/-1} = 2 \cdot O_{1/0} - (k \times k \times c_{in}) \tag{2}$$

where $O_{+1/-1}$ is the output element from the original binary convolution, $O_{1/0}$ is the output element from the encoded binary convolution, and $k \times k \times c_{in}$ is the number of bitwise XNOR operations required for each output element.

The accuracy loss issue of XNOR-Net is considerably improved by ABC-Net [18], which exploits the combination of multiple binary weights and activations. This scheme achieves comparable prediction accuracies to those of the full-precision networks on ImageNet [29], so it enhances the feasibility of BNNs in practical applications further.

B. In-memory Processing in SRAM

SRAMs are commonly used as on-chip buffers in most of the digital systems including ASICs. In CNN accelerators, they usually store kernel and feature map data to exploit abundant data reuses.

The architecture of the conventional 6T SRAM is shown in Fig. 2(a). Once the data are stored in the cell array, each data word is accessed sequentially in a row-by-row fashion. For each access, a target row is activated by a one-hot-encoded word line (WL) signal. In each bit cell of the activated row, either bit line (BL) or bit line bar (BLB) is discharged depending on the cell data. The target read data is produced by sensing the difference between the voltages of BL and BLB, and then it is transmitted to the computing core through the data bus.

The memory bottlenecks of CNNs mentioned in Section I are mostly caused by this conventional flow. This single-row activation limits the memory bandwidth, so a significant portion of the data parallelisms of neural network [30] is restricted. In addition, massive data movements between a buffer and a core through the data bus consume incredibly more energy compared to logic operations (e.g., 50x more than a 32-bit floating point addition [31]).

The emerging in-memory processing in SRAM utilizes bit cell modification and multiple-row activation to address these bottlenecks. In Fig. 2(b), the conventional 6T cell is expanded by adding additional transistors to support binary multiplications such as bitwise XNORs. Each result of the multiplications is produced by sensing the difference between the voltages of BL and BLB in the modified bit cell. Activating multiple rows makes multiple voltage discharges along the common BL or BLB, which are summed together. Then, the difference between the total discharged voltages from BL and BLB becomes the result of an accumulate operation which is a popcount. Some of the peripheral circuits have to be modified to enable the multiple-row activation. WL decoder is expanded to produce the signal for activating multiple rows in addition to the one-hot-encoded signal for a single row. Sense amplifiers are also replaced by ADCs to produce multi-bit outputs.

The feasibility of in-memory processing in SRAM is proved by a couple of fabricated chips [19], [20], [25]. Although they have made considerable improvements, the overhead imposed on the memory itself is not negligible. The modified bit cells impair the layout of the cell array optimized for density, thereby reducing the memory capacity by around 55% [25]. Besides, activating multiple rows generates huge current along each BL and BLB, resulting in large power consumption which is proportional to the number of the activated rows. The ADCs also drain 2-5 times more energy than sense amplifiers, reducing the energy-efficiency of the memory. Considering all these problems, mapping BNN to SRAM naïvely is not an appropriate approach.

C. In-memory Processing in DRAM

The DRAM is also a commonly used type of memories, usually as an off-chip main memory. At the top level, the DRAM consists of multiple ranks, which are the sets of multiple banks. Each bank is composed of multiple sub-

arrays and command-related peripheral logic blocks. Each subarray is divided into bit cell arrays, sense amplifiers, and row decoders.

The read access of the conventional 1T1C DRAM is described in Fig. 3(a). In the initial state, each BL is precharged to $\frac{1}{2}V_{DD}$, and WL is in the off state. Issuing ACTIVATION command activates a target row by turning on the WL of that row, and each bit cell in the activated row is connected to its BL. Then, charge sharing occurs between the cell capacitor and BL, inducing voltage discharge. Depending on the cell data, the amount of the voltage discharge along BL is determined. The sense amplifier amplifies the difference between the BL voltage and the reference voltage ($\frac{1}{2}V_{DD}$), and then the cell data is copied to the row buffer.

Unlike SRAM, implementing in-memory logic operations in DRAM is extremely challenging due to the different fabrication process. The DRAM process supports only three metal layers and is ultimately optimized for the maximum density, whereas the logic process has more than ten metal layers and is optimized for the speed [32]. Modifying the bit cells and the peripheral circuits of DRAM induces incomparably larger overheads than the case of SRAM. As an alternative, a prior work related to in-memory processing in DRAM implemented primitive bitwise operations such as AND and OR [26]. Three rows are simultaneously activated to utilize the charge sharing effect as shown in Fig. 3(b). Two of them are two inputs for the target bitwise operation, and the other row is the selection row which determines the type of the operation. Since the output of the bitwise operation is binary, replacing sense amplifiers by ADCs is not required.

The limitation of this approach lies on the throughput issues when building complex logic functions. Only the implemented bitwise operations can be processed in a single cycle. The other logic functions are processed by serially cascading them. Even bitwise XNORs require more than one cycle, or the same data have to be copied to the other cells, resulting in the reduced cell capacity. To build complex multi-stage logic functions including popcounts, either latency or the memory capacity has to be sacrificed additionally. Therefore, processing neural network algorithms in DRAM suffers from severe performance degradations.

III. Proposed Architecture

As mentioned in Section II, each convolution in XNOR-Net is performed by bitwise XNOR and popcount, but implementing them in memories causes significant performance overhead. In the proposed NAND-Net, all the bitwise XNOR and popcounts are converted to simpler operations, which are flexible and can be implemented in memories efficiently. There are two key features in NAND-Net, which are conv decomposition and popcount compression.

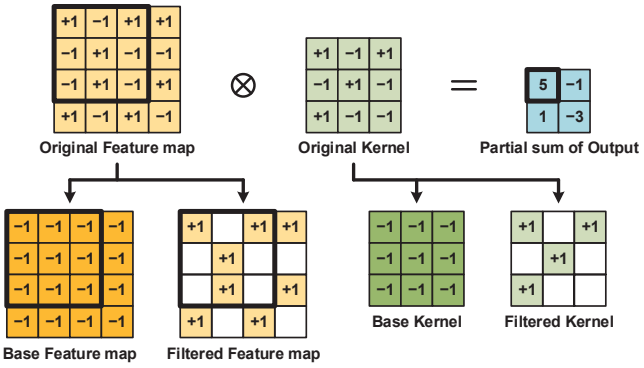


Figure 4. Decomposition method for each feature map and kernel

A. Conv Decomposition

1) Decomposition Method: Conv decomposition optimizes the binary multiplication which is bitwise XNOR. To convert XNOR into a simpler operation, we focused on the fact that an arbitrary sequence of binary numbers $(+1, -1)$ can be represented by collecting the positional information of only either $+1$ or -1. For example, a binary sequence $S = (+1, -1, -1, +1, -1)$ can be reconstructed from $P = (0, 3)$ which is the positional information of $+1$s in S. With this observation, we extended the kernel decomposition scheme in [27] which targets only binary-weight CNNs into the proposed conv decomposition to consider both binary feature map and binary kernel. In this architecture, each binary convolution is decomposed into four sub-convolutions which will be described in the following subsections.

For every convolutional layer in BNN, each feature map and each kernel are decomposed into two sub-feature maps and two sub-kernels, respectively, as depicted in Fig. 4. The decomposed sub-feature maps are a base feature map and a filtered feature map. The base feature map is made by replacing all $+1$ elements in the original feature map with -1s. The filtered feature map is made by eliminating all -1 elements in the original feature map, so it takes the form of a perforated feature map in which -1s are filtered. The same method is applied to the decomposed two sub-kernels which are a base kernel and a filtered kernel. The original feature map and kernel are reconstructed by (3) and (4):

$$\mathbf{I}_o = \mathbf{I}_b + 2 \cdot \mathbf{I}_f \tag{3}$$

$$\mathbf{W}_o = \mathbf{W}_b + 2 \cdot \mathbf{W}_f \tag{4}$$

where the subscripts 'o', 'b', and 'f' denote 'original', 'base', and 'filtered', respectively.

With the decomposed sub-kernels and sub-feature maps, the original convolution is decomposed into four sub-convolutions by the following equations. They are represented in Fig. 5.

$$\mathbf{I}_o * \mathbf{W}_o = (\mathbf{I}_b + 2 \cdot \mathbf{I}_f) * (\mathbf{W}_b + 2 \cdot \mathbf{W}_f) \tag{5}$$

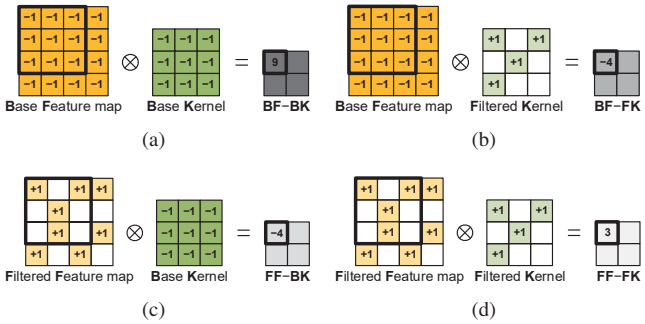


Figure 5. Decomposed sub-convolutions: (a) BF-BK convolution, (b) BF-FK convolution, (c) FF-BK convolution, (d) FF-FK convolution

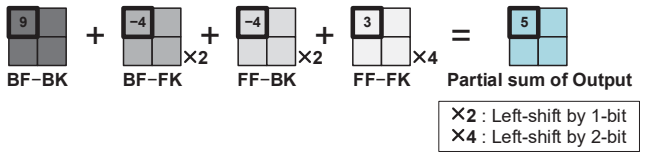


Figure 6. Reconstruction of the original convolution

$$= \mathbf{I}_b * \mathbf{W}_b + 2 \cdot (\mathbf{I}_b * \mathbf{W}_f) + 2 \cdot (\mathbf{I}_f * \mathbf{W}_b) + 4 \cdot (\mathbf{I}_f * \mathbf{W}_f) \quad (6)$$

Fig. 6 illustrates the reconstruction of the original convolution. Once the four sub-convolutions are done, the linear combination of the four outputs in (6) is computed by only left-shift and add operations without any multiplications.

In the subsequent subsections, each of the decomposed four sub-convolutions will be discussed in detail. The numbers of the input and output feature maps are M and N, respectively.

2) Base Feature map & Base Kernel: In the sub-convolution between **B**ase **F**eature map and **B**ase **K**ernel (**BF-BK**), all the elements in feature maps and kernels are -1 as shown in Fig. 5(a). Since each result of the multiplications between each feature map element and each kernel element is 1, the output value is a constant computed by counting the number of accumulations assigned to it. This constant equals to $k \times k \times M$, which can be pre-computed on offline without any runtime information of real-time feature maps. If this constant is merged into the bias element, adding the result of the BF-BK convolution to the remaining sub-convolutions in (6) can be hidden. Therefore, BF-BK convolution causes no runtime overheads during the execution.

3) Base Feature map & Filtered Kernel: Convolving **B**ase **F**eature map with **F**iltered **K**ernel (**BF-FK**) does not produce constant outputs due to the perforated elements in the kernel as shown in Fig. 5(b). The result of the multiplication between a feature map element and a perforated kernel element is 0, which can be skipped. Since every non-perforated kernel element and every feature map element are $+1$ and -1 respectively, multiplications between them always result in -1. Then, the output of the convolution can be obtained by counting the number of $+1$ elements in

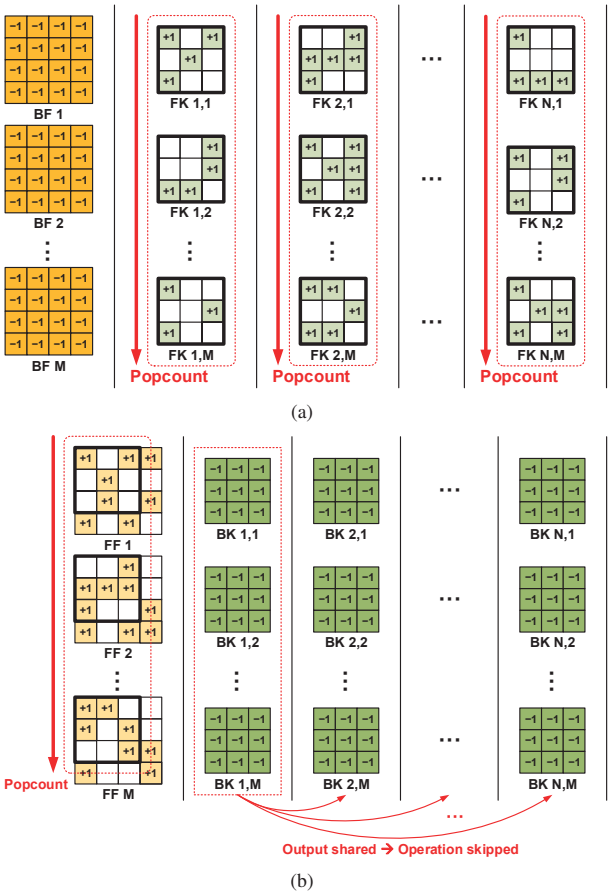


Figure 7. Operation minimization: (a) BF-FK convolution, (b) FF-BK convolution

the kernel and inverting the sign. Considering the multiple input channels, the range of the popcount expands to all kernels in the channel direction assigned to each output feature map as shown in Fig. 7(a). Like BF-BK convolutions, BF-FK convolutions also need not be processed during the runtime since they do not require any information of real-time feature maps. The results of BF-FK convolutions can be pre-computed on offline and merged into the bias elements.

4) Filtered Feature map & Base Kernel: Unlike BF-BK and BF-FK convolutions, sub-convolutions with filtered feature map cannot be skipped by pre-computing since they require information of real-time feature maps acquired during the runtime. In the sub-convolution between **F**iltered **F**eature map and **B**ase **K**ernel (**FF-BK**), the result of the multiplication is accumulated only when the feature map element is not perforated. For an efficient hardware mapping, this sub-convolution is simplified to counting the number of $+1$ elements in each kernel window of the feature map and inverting the sign as shown in Fig. 5(c). This operations are so simple, but processing them during runtime requires additional energy and latency. However, since all the base

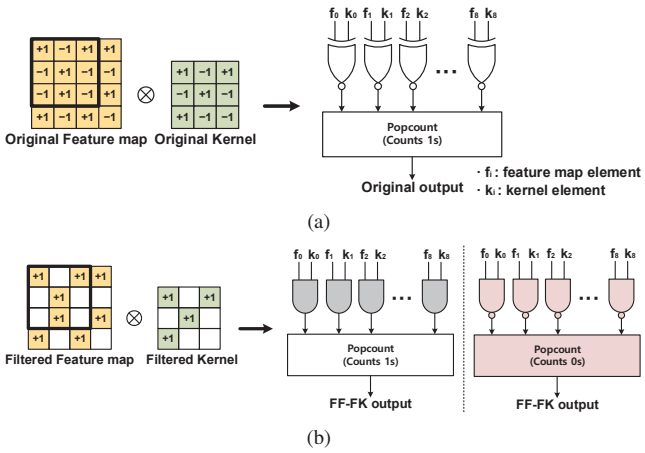


Figure 8. Operation conversion from XNOR to NAND: (a) Original convolution in XNOR-Net, (b) FF-FK convolution in NAND-Net

kernels are identical to each other, convolutions for all output channels except the first channel can be skipped as shown in Fig. 7(b), reducing the overhead significantly.

5) Filtered Feature map & Filtered Kernel: The sub-convolution between **F**iltered **F**eature map and **F**iltered **K**ernel (**FF-FK**) in Fig. 5(d) is the main part among the four sub-convolutions. In the FF-FK convolution, the perforated elements exist in both feature maps and kernels, so no offline pre-computing is possible for that. In addition, operation skipping by exploiting data reuse is not possible since all feature maps and all kernels are different to each other, respectively. All convolutions required for a given layer should be fully processed during the execution likewise the original convolution. The difference between the original convolution and FF-FK convolution is the actual value of the '0' element. Whereas '0' element in the original convolution stands for -1, it represents a perforated element which is zero in the FF-FK convolution. The results of the multiplication between '0' kernel element and '0' feature map element differ between them (e.g., 1 for the original convolution and 0 for FF-FK). Consequently, the bitwise XNOR operation of the original convolution can be replaced by a bitwise AND in the FF-FK convolution.

This AND operation can be further optimized by inverting the subsequent popcount operations. The original popcount operations count the number of 1s in a given bit sequence. If we change the function of them into counting the number of 0s, the AND operation can be converted to NAND as represented in Fig. 8, which is one of the simplest primitive logic operations from the perspective of hardware implementation.

As will be discussed in Section IV, NAND is more flexible and memory-friendly than XNOR. Therefore, the proposed conv decomposition optimizes the baseline XNOR-Net into NAND-Net, which is more suitable for in-memory processing.

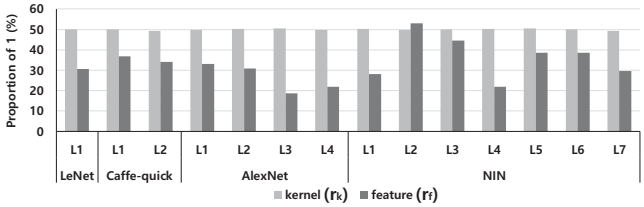


Figure 9. Proportions of 1s in the feature maps (r_f) and the kernels (r_k)

B. Popcount Compression

Converting XNOR to NAND brings an opportunity to optimize the subsequent popcounts. In this section, we propose two popcount compression techniques to reduce the hardware overhead induced by in-memory popcounts in DRAM and SRAM, respectively.

1) Target Bit Statistics: In general, a popcount operation counts the number of either 1s or 0s in a given bit sequence. We define 'target bit' as the symbol to be counted in the popcount (e.g., '1' or '0'). As described in Section III-A5, converting XNORs to NANDs inverts the target bit of XNOR-Net from 1 to 0 in NAND-Net. Accordingly, the patterns of the bit sequences fed to the popcounts are also changed since the preceding XNORs and NANDs generate different computation results.

If all the elements of the feature maps and kernels in a convolutional layer (L) are independent and uniformly distributed, the probabilities of occurrence of target bit would be 50% and 25% for XNOR-Net and NAND-Net, respectively. However, the actual distributions of them are correlated and not uniform, so those probabilities are incorrect when applied to real BNN models. For an in-depth analysis of them, we measured the proportions of 1 in the feature maps and kernels from our target BNN models, which are LeNet [1], Caffe-quick [33], AlexNet [2], and NIN [34].

As shown in Fig. 9, feature maps and kernels show different characteristics in magnitudes of their proportions. Whereas the proportions of kernels (r_k) always fall within $\pm$ 1% of the average 50%, those of the feature maps (r_f) vary widely from layer to layer. r_f is far below 50% in most of the layers except for the L2 in NIN. The average of them is 32.7%, which means that about two-thirds of the feature map elements are 0.

With these proportions, the target bit probabilities of XNOR-Net (p_{xnor}) and NAND-Net (p_{nand}) can be derived as follows.

$$p_{xnor} = r_f \cdot r_k + (1 - r_f) \cdot (1 - r_k) \tag{7}$$

$$p_{nand} = r_f \cdot r_k \tag{8}$$

Under the constraint that r_k is approximately 50%, p_{xnor} converges to 50% according to (7). This can be verified in real BNN models as shown by grey bars in Fig. 10(a). p_{xnor} is always 50% with a deviation of smaller than 0.5%

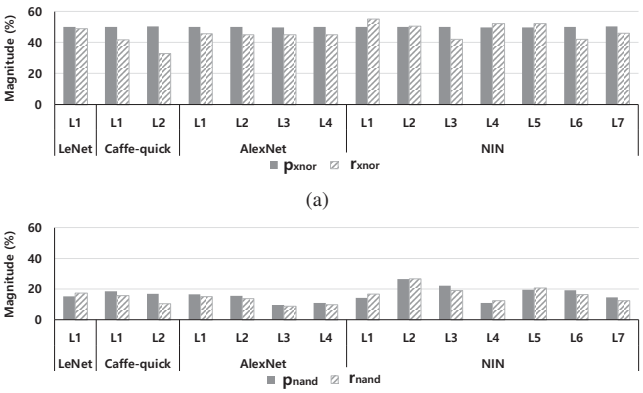


Figure 10. Target bit probability & Measured ratio of target bit: (a) XNOR-Net, (b) NAND-Net

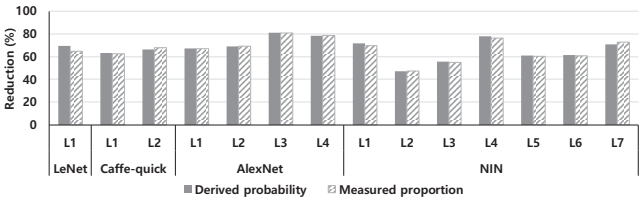


Figure 11. The reduction of the target bit by NAND-Net

for every layer in the target models. That is, the target bit probability of XNOR-Net is constant regardless of r_f.

On the other hand, p_{nand} is linearly proportional to r_f according to (8), and it converges to $\frac{1}{2}r_f$ when r_k goes to 50%. As shown in Fig. 10(b), it varies widely depending on r_f, and its magnitude is significantly smaller than that of p_{xnor}.

The dashed bars in Fig. 10(a) and 10(b) are the measured proportions of the target bit in XNOR-Net (r_{xnor}) and NAND-Net (r_{nand}). These proportions do not match exactly to the derived expectations in (7) and (8) due to the correlations among the feature maps and kernels. However, those correlations are negligible as shown by the slight differences between the grey bar and the dashed bar for each column in Fig. 10. They can be neglected without affecting the statistical characteristics of the target bits.

As can be proved by comparing Fig. 10(a) and 10(b), the occurrence of target bit is reduced by more than half in most of the layers. The amounts of these reductions are shown in Fig. 11. The grey and the dashed bars represent the reductions of the number of target bit derived from probabilities (p_{xnor} & p_{nand}) and those based on the measured proportions (r_{xnor} & r_{nand}), respectively. Both of them show the same tendency although slight differences exist between them due to the correlations. The average of the reductions based on the measured proportions is 66.9%, which means that two-thirds of the target bits are eliminated by converting XNORs to NANDs.

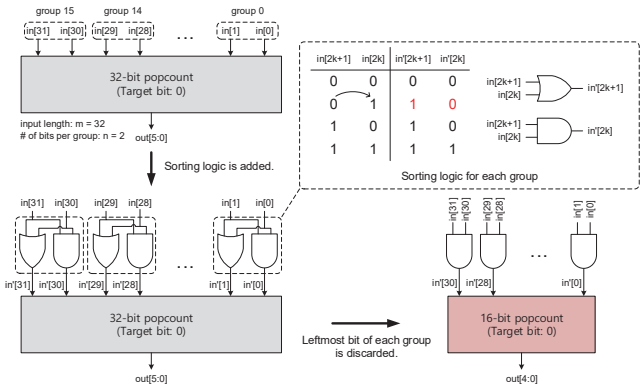
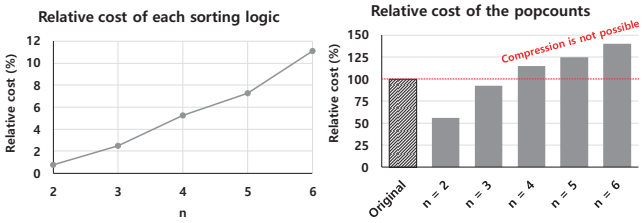


Figure 12. Popcount compression in DRAM for $m = 32$, $n = 2$ (m: input length, n: number of bits per group)



(a) (b)

Figure 13. (a) The relative size (number of cascading stages) of the sorting logic, (b) Relative size of the popcounts

2) Compressing Popcount in DRAM: As mentioned in Section II-C, in-DRAM popcount is performed by serially cascading the implemented primitive logic operations. This serial manner severely degrades the overall throughput when processing large-scale BNN models with billions of popcounts. To mitigate the degradation, we minimized the number of cascading stages of popcounts by exploiting the sparse occurrence of target bit in NAND-Net discussed in Section III-B1.

If the probability of target bit (p_{nand}) is sufficiently small, the probability of that at least one bit in a group of n bits is not the target bit (p'_{nand}) would be close to 1, which is calculated by (9).

$$p'_{nand} = 1 - (p_{nand})^n \tag{9}$$

Once an input sequence fed to m-bit popcount is divided into groups of n bits ($m > n$), at least $\lfloor \frac{m}{n} \rfloor$ bits would not be the target bit if p'_{nand} is approximated to 1. These non-target bits are not counted by the popcount, so they can be discarded from the input sequence, resulting in a reduction of the popcount size. To discard only the non-target bits, we inserted a sorting logic in front of each group as shown in Fig. 12. It gathers every target bit in the group to the right side so that all the bits in the group are sorted. Then, the leftmost bit of every group would always be a non-target bit. By discarding those bits from all the groups, only the non-target bits can be eliminated from the

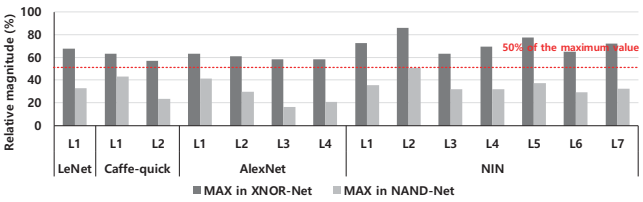
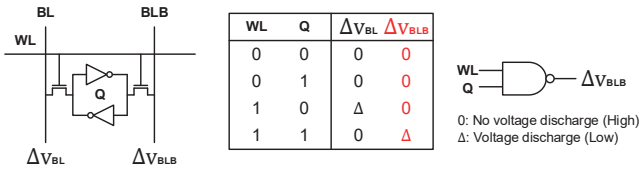


Figure 14. The relative magnitude of the maximum popcount results



WL	Q	ΔV_{BL}	ΔV_{BLB}
0	0	0	0
0	1	0	0
1	0	Δ	0
1	1	0	Δ

Figure 15. NAND operation implemented in the conventional 6T cell of SRAM

bit sequence. Consequently, the original m-bit popcount can be compressed to a $(m - \lfloor \frac{m}{n} \rfloor)$-bit popcount at the cost of some sorting logic blocks.

The efficiency of this compression technique is maximized when $n = 2$ since the relative size of each sorting logic to that of the original popcount grows rapidly as shown in Fig. 13(a). When $n = 2$, the length of the bit sequence is reduced by half at the cost of just a couple of AND operations (e.g., 16 ANDs for 32-bit popcount) as shown in Fig. 12. In this case, the overall reduction of the number of cascaded stages of the popcount is 44.1%. It rather increases when n is larger than 3 as shown in Fig. 13(b).

Applying this technique to NAND-Net, however, causes the results of the popcount to be incorrect since the actual value of p'_{nand} is smaller than 1. These errors are accumulated through the network, degrading the accuracy of the model. We minimized the effect of them by retraining the network after the popcount compression. Thus, the accuracy drops are reduced to marginal levels as will be shown in Section V and Table IV. These marginal drops are reasonably negligible considered that the bottleneck by the in-memory popcount is significantly reduced.

3) Compressing Popcount in SRAM: In-memory popcount in SRAM is processed by accumulating the discharged voltages from the bit cells along the common bit line. It can be processed in a single cycle regardless of the length of the input bit sequence, unlike the case of DRAMs. Thus, popcount compression in Section III-B2 is less effective for SRAMs. In addition, the sorting logic in that technique rather increases the hardware overhead since they require bitwise AND operations among the bit cells, which cannot be implemented without modifying the conventional cell array.

Instead, we focused on the ADCs, which produce digital outputs from the accumulated voltage discharges. These ADCs impose huge overhead to the SRAM since their hardware cost in terms of area and energy is significantly larger than that of 1-bit sense amplifiers. It even grows

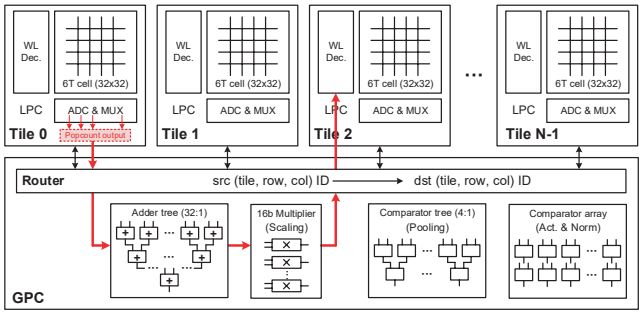


Figure 16. In-memory accelerator for NAND-Net in SRAM

exponentially as the resolution of the ADCs increases, so the overall performance of in-memory popcounts is significantly degraded.

To lower the overhead, we decreased the resolution of each ADC by exploiting the sparse occurrence of the target bits in NAND-Net. As can be seen in Fig. 11, the number of the generated target bits is considerably reduced by converting XNORs to NANDs. The magnitude of the result of popcounts also decreases accordingly. Fig. 14 represents the relative magnitude of the maximum results of the popcount compared to the upper limit value that can be expressed at the given resolution. For XNOR-Net, the relative magnitude is always bigger than 50% regardless of the layers. That is, the full voltage range of the ADC is utilized for producing popcount results. On the other hand, the relative magnitudes in the case of NAND-Net are far below the half of the upper limit except for L2 of NIN, which is 50.6%. That is, they are produced by using mostly the lower half of the full voltage range in the ADC, and the upper half is rarely utilized. The unused upper range can be eliminated by reducing the resolution of each ADC by one bit. Then, the hardware overhead of the ADCs can be alleviated, reducing the cost of them effectively. Although some value errors may occur due to the diminished resolution, the occurrence of them is extremely low as shown in Fig. 14. Their effects on the network accuracy can be neglected after the retraining process.

IV. Hardware Implementation

A. NAND-Net in SRAM

1) NAND in 6T Bit Cells: NAND operations in NAND-Net can be implemented in the conventional 6T bit cells of SRAM without any modifications to them unlike XNORs in XNOR-Net. In Fig. 15, the feature map and kernel elements are represented by word line (WL) and cell data (Q), respectively, and the voltage discharges on BL and BLB are denoted by Δv_{BL} and Δv_{BLB}. Those values vary depending on the combination of WL and Q. As represented in Fig. 15, Δv_{BLB} is generated only when both WL and Q are 1. Therefore, the NAND operation is effectively performed by sensing the voltage on BLB. Multiple voltage discharges generated by the cells in the same column are accumulated

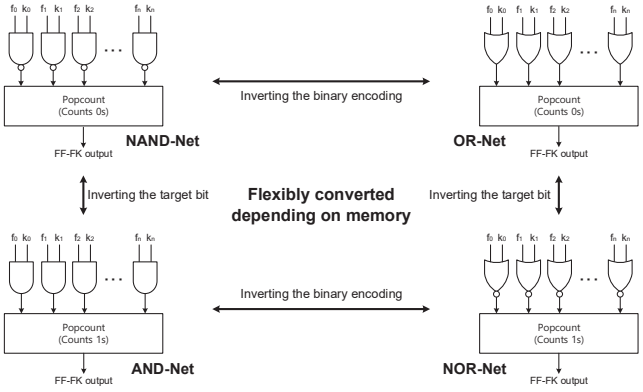


Figure 17. Flexible Conversion of NAND-Net

Table I
DRAM SPECIFICATIONS

Model	4GB DDR4-2400 (JEDEC)
Config. Info.	16 banks, 2KB row buffer
Timing Param.	tRAS: 32ns, tRP: 14.16ns
Width Info.	Word-length: 64B
Requirements	Support of Ambit [26]

Table II
SRAM SPECIFICATIONS

Model	2KB Standard 6T (ITRS) - 65nm CMOS
Config. Info.	16 tiles (128B per each), Word: 16B
Core Param.	V_{dd}: 1.1V, Clk: 10ns
Cell Tr. Width (F)	access/p-down/p-up: 1.31/1.23/2.08
Cell Sizing (F^2)	Area: 146, AR = 1.46
ADC	4-bit Multi-level-SA [25]

along the common BLB, and the result of the accumulation is sensed by the ADC located at the bottom of the column, resulting in the output of the popcounts.

2) Overall Architecture: The overall architecture of the in-memory accelerator for NAND-Net in SRAM is shown in Fig. 16. It is an array of tiles which consist of 6T cell array and local peripheral circuits (LPC) including a WL decoder and ADCs. ADCs in the LPC are multi-level sense amplifiers exploited in [25], which are optimized by the proposed popcount compression. Each tile processes the proposed NAND-based binary convolutions (FF-FK), and the results of them are transmitted to the global peripheral circuit (GPC) which connects multiple tiles.

In the GPC, the adder tree sums all the intermediate results of the popcounts to produce the final output, and the multipliers perform the scaling operations. The 4:1 comparator trees and the comparator array are used to process subsequent pooling, activation and normalization layers which can be computed by simple compare operations as used in [21]. Data movements between the tiles are mediated by the router which exploits multicast communication based on src and dst IDs. Each ID is composed of three indexes which indicate tile, row, and column, respectively.

In Fig. 16, once src and dst IDs are fetched to the router, the popcount outputs from the source tile are transmitted to one of the adder tree, comparator tree, and comparator array depending on the layer type. After the target computation, the final output is transmitted to the WL decoder of the destination tile through the router. Regarding the data partitioning, we adopted the same strategy in [25] to map arbitrary shapes of the network.

The results of BF-BK and BF-FK convolutions are pre-added to the bias parameters on offline, so adding them to the results of FF-FK convolutions are automatically processed when bias additions occur in GPC. Computing FF-BK convolutions is processed by a dummy tile in which all the cell data are set to 1. The results of them can be reused by the other tiles through the GPC.

B. NAND-Net in DRAM

1) Baseline Architecture: The baseline architecture of our in-DRAM accelerator for NAND-Net is Ambit [26], which implemented bitwise AND, OR, and NOT operations without modifying the conventional 1T1C cells. Theoretically, any complex logic can be processed in this architecture by serially cascading the implemented primitive operations, but the bottleneck is the throughput which is severely degraded as the size of the logic enlarges. We minimized such throughput bottleneck by reducing the number of the stages of the operation cascading and by utilizing the peripheral area for processing non-binary operations. The cell area consists of 16 banks with 2KB row buffers, and digital blocks in the peripheral area are identical to those of GPC in Fig. 16.

2) Utilizing Flexibility of NAND-Net: NAND operations in NAND-Net can be flexibly converted to the other primitive operations such as AND or NOR. In Section III-A5, it was shown that the NAND operation followed by popcount with target bit 0 is logically identical to the AND operation followed by popcount with target bit 1. Simply inverting the target bit of the popcount can convert NAND to AND. Not only that, inverting the binary encoding of feature maps and kernels enables conversions from NAND to NOR and OR as shown in Fig. 17. For instance, if we encode $+1$ elements in feature maps and kernels to 0 and -1 elements to 1, NAND followed by popcount with target bit 0 can be converted to OR followed by popcount with target 0. This OR-based convolution is flexibly converted to NOR-based convolution by inverting the target bit likewise the case of NAND to AND.

In Ambit-based architecture, AND is more advantageous than NAND in terms of the latency cost since NAND is processed by cascading AND and NOT. Thus, AND-based convolution is more efficient when mapping NAND-Net to Ambit. In the case of DRISA [22] which implemented NOR as the basic operation, NOR-based convolution is more efficient. In other words, NAND-Net can be flexibly switched to the other version (e.g., AND-Net or NOR-Net)

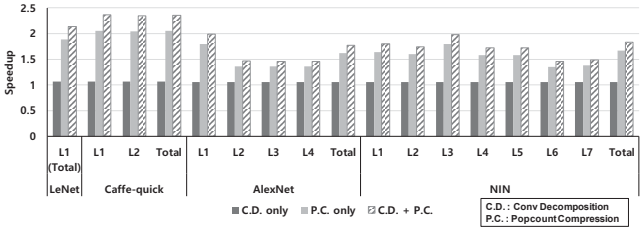


Figure 18. Throughput improvement in DRAM

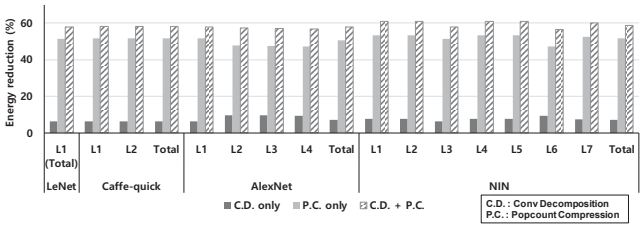


Figure 19. Energy reduction in DRAM

depending on the characteristics of memory.

V. Evaluation

A. Experimental Setup

Two in-memory accelerators for NAND-Net are implemented in DRAM and SRAM, respectively. The memory specifications are summarized in Table I and II. Regarding DRAM, we built an in-house simulator based on CACTI [35] for modeling and extracting the DRAM parameters. For SRAM, we customized NeuroSim [36] to implement the design in Fig. 16. ADC modeling is conducted by Cadence tools, and all the digital blocks located at the peripheral area are designed and synthesized using Synopsys Design Compiler under Samsung CMOS 65nm library. The baseline of DRAM is Ambit-based processing of XNOR-Net, and that of SRAM is 8T cell-based processing in [25]. For the experiments, we trained and validated four BNN models using PyTorch [37] and Caffe [38] APIs, which are LeNet, Caffe-quick, AlexNet, and NIN.

B. Results of NAND-Net in DRAM

1) Throughput: Fig. 18 shows the throughput improvements by NAND-Net in DRAM. We made three comparison groups which are 'C.D. only' (only conv decomposition is applied), 'P.C. only' (only popcount compression is applied), and 'C.D.+P.C.' (Both are applied). The latency bottleneck of processing BNN in DRAM is mainly caused by in-memory popcounts which require a large number of operation cascading. Popcount compression effectively alleviates this bottleneck by a factor of more than 1.6x as shown by the middle bars in each column of Fig. 18. The speedup even reaches up to 2x in Caffe-quick, which implies that the number of operation cascading is reduced by half. Conv decomposition improves the latency of binary multiplication by 41.5%. Although its contribution to the overall speedup

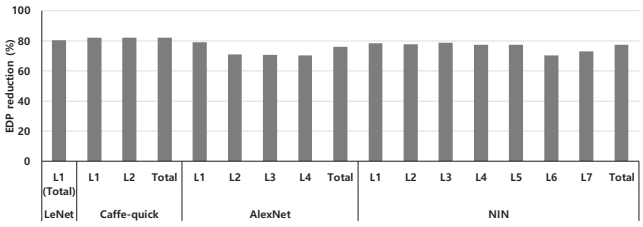
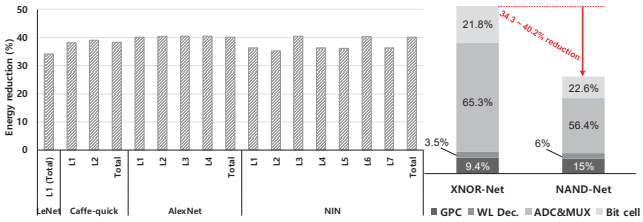


Figure 20. EDP reduction in DRAM (C.D. + P.C.)



(a) (b)

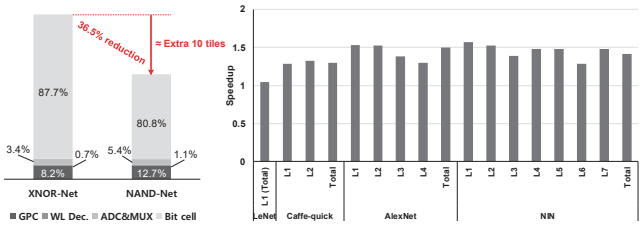
Figure 21. (a) Energy reduction in SRAM, (b) Energy breakdown of SRAM

is quite small (around 1.07x), it is essential for reducing the occurrence of target bit so that the proposed popcount compression can be applied to BNN.

In total, NAND-Net with both techniques achieves a speedup of 2.14x, 2.36x, 1.77x, and 1.84x for LeNet, Caffe-quick, AlexNet, and NIN, respectively. The effect of NAND-Net is more substantial in LeNet and Caffe-quick of which latency for data allocation is negligible due to the relatively small parameter size, but the difference is not huge. As proved by the magnitude of the speedup, NAND-Net effectively improves the latency bottleneck mentioned above regardless of the network.

2) Energy Consumption & EDP: Fig. 19 shows the energy reduction by NAND-Net in DRAM. Since in-memory computations are performed by activating triple rows simultaneously as mentioned in Section II-C, they consume a lot of energy compared to the normal data allocation process. Thus, the overall energy consumption is dominated by the in-memory computations, which implies that optimizing these computations can increase the energy-efficiency effectively. The total energy reduction of each model falls within $\pm$ 1% of the average 58.15% regardless of the network size, proving the effectiveness of NAND-Net. The reduction by conv decomposition is around 5-6%, and popcount compression contributes to the remaining portion of the total reduction.

Since NAND-Net considerably reduces both the latency and the energy consumption regardless of the model, EDP is also significantly improved as shown in Fig. 20. NAND-Net achieves an EDP reduction of 80.4%, 82.2%, 76.1%, and 77.5% for each of the target models, respectively. This implies that the proposed techniques improve both the latency and the energy at the same time without any trade-off between them.



(a) (b)

Figure 22. (a) Area breakdown of SRAM, (b) Speedup in SRAM with additional tiles

Table III
OVERHEAD CAUSED BY FF-BK CONVOLUTION

Model	LeNet	Caffe-quick	AlexNet	NIN
Memory footpr. (%)	2	2.35	0.33	0.61
Latency (%)	0.67	0.13	0.08	0.22

C. Results of NAND-Net in SRAM

1) Energy Consumption: Fig. 21(a) shows the energy reduction by NAND-Net in SRAM. NAND-Net can be processed in the conventional 6T SRAM without any customizations to the bit cell such as the 8T cell in the baseline. Since the 6T cell consumes 40% less energy compared to the 8T cell, NAND-Net considerably reduces the energy consumption of both the in-memory computing and the data allocation. Reducing the resolution of ADCs also saves 50% energy since the number of sensing stages is reduced by half. Moreover, the sparse occurrence of the target bit in NAND-Net further reduces the dynamic power consumption of both the bit cells and the ADCs. In total, NAND-Net results in 34.3%, 38.4%, 40.2%, and 40.1% reduction of the energy consumption for the target models, respectively.

Fig. 21(b) illustrates the energy breakdown of SRAM. It proves that NAND-Net effectively reduces the energy consumptions of two major components which are the bit cell arrays and the ADCs.

2) Throughput & Area Reduction: If the number of tiles in Fig. 16 is fixed, the execution time of NAND-and-popcount in NAND-Net is identical to that of XNOR-and-popcount in XNOR-Net since no operation cascading is required unlike the case of DRAM.

On the other hand, replacing 8T bit cell to 6T by applying NAND-Net reduces the size of each bit cell by 45%, which results in 36.5% reduction of the total area as shown in Fig. 22(a). This area reduction can be exploited to boost the overall throughput. By increasing the number of tiles, more computations can be processed in parallel, resulting in effective speedup. In our accelerator with 16 tiles, the amount of the reduced area is translated to the area of additional 10 tiles as shown in Fig. 22 (a). Therefore, the augmented accelerator with 26 tiles can improve the throughput as represented in Fig. 22 (b). The total speedups of NAND-Net are 1.04x, 1.3x, 1.5x, and 1.41x for the target models, respectively. The speedup for LeNet is slight since the throughput bottleneck for this small-scale model is mainly caused by the allocation of data rather than computations.

Table IV
THE EFFECT OF P.C. ON THE ACCURACY

Model	LeNet	Caffe-quick	AlexNet	NIN
Original acc. (%)	99.23	73.92	44.87	86.28
Retrained acc. (%)	99.30	73.50	43.77	86.02
Acc. drop (%)	−0.07	0.42	1.10	0.26

D. Overhead Analysis

To hide the latency overhead of FF-BK, we assign a dummy tile for processing FF-BK in parallel along with FF-FK as explained in Section IV-A2. The lifetime of the dummy tile is extremely short since the FF-BK convolution is processed only for the first output feature map. Table III shows the proportion of the required memory footprint for allocating the dummy tile and the resulting latency overhead. Considering the short lifetime of them, these overheads can be neglected reasonably.

Table IV shows the effect of the popcount compression on the network accuracy. Retraining the network recovers the original accuracy for every model with a marginal drop which is at most 1.1%.

VI. RELATED WORKS

A variety of works have been proposed to accelerate BNNs on various platforms. XNORBIN [39] implemented BNN accelerators built in ASIC. They used loop unrolling and data-reuse techniques to exploit the inherent parallelism of BNNs, but their designs are kinds of a straightforward mapping of BNNs into hardwares, so their state-of-the-art results mainly come from the algorithmic efficiencies of BNNs.

Li et al. [28] designed an FPGA-based BNN accelerator by exploiting abundant LUT components. They merged the normalization and the binarization layer into a single comparison operation to efficiently implement them with LUTs. They also conducted a design space exploration for the throughput modeling to maximize the performance. Although the resource utilization is maximized upon LUT based computing, its performance is limited by the von Neumann bottleneck of BNNs.

The emerging in-memory processing architectures partially address this bottleneck. Liu et al. [25] implemented BNN processings in the SRAM arrays to exploit the high parallelism of them. Each 6T bit cell is customized to 8T to support XNOR operations, and sense amplifiers are replaced by multi-bit ADCs, which are further optimized by quantizations. Though the feasibility of this work was verified by a fabricated chip, the design overhead due to the customized 8T cells causes the scalability and the reliability issues when dealing with large-scale network models.

Ambit [26] exploited the analog operation of DRAMs to perform bitwise operations fully inside DRAM. They activated three rows simultaneously to perform AND and OR operations and modified the sense amplifiers at a modest level to perform NOT operations. Complex logic functions are processed by cascading AND, OR and NOT operations. The limitation of this approach lies on the latency issues when implementing complex logic consisting of a massive number of the primitive logic operations. The approach of DRISA [22] is similar to Ambit, but they implement only NOR as the base primitive operation. In DRISA, 1T1C cells are augmented to 3T1C to support NOR, and shifting logics are implemented to route the internal data movements efficiently. They further optimized the latency issues by reorganizing the DRAM hierarchy. However, their work suffers from the process variations and the design overhead due to the customized cells and the hierarchy.

VII. Conclusion

In this paper, we propose NAND-Net, an efficient architecture targeting high-performance in-memory processing for BNNs in both DRAM and SRAM to address the overhead issues in the prior in-memory processing architectures. The proposed scheme optimizes the computations of BNNs to fit them in the conventional memory rather than modifying the memory architecture which was the approach used by the prior works. All the operations in each binary convolution are modified by two techniques to remove the redundant and unnecessary parts from them. Bitwise XNOR operations are converted to NANDs by decomposing the original convolution into four sub-convolutions and minimizing them. In addition, the derived statistics of the target bit in NAND-Net are utilized to reduce the size of the popcounts. This popcount compression technique appropriately reduces the overhead of the in-memory popcounts in DRAM and SRAM. Applying NAND-Net to the conventional in-memory architecture achieves 1.04-2.36x speedup, 34-59% energy saving, and 41.5% area reduction. Our approach effectively alleviates the inefficiencies in the prior works in terms of latency and energy consumption. Moreover, NAND-Net can be compatibly applied to the other types of memory and even to other implementations including ASIC and FPGA, thus making it a flexible solution to perform high-performance BNNs in an energy-efficient way regardless of the platforms.

Acknowledgment

This research was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (NO. 2017R1A2B2009380).

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[3] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 806–813, 2014.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[6] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2, pp. 1150–1157, Ieee, 1999.

[7] J. Sim, J.-S. Park, M. Kim, D. Bae, Y. Choi, and L.-S. Kim, "14.6 a 1.42 tops/w deep convolutional neural network recognition processor for intelligent ioe systems," in *Solid-State Circuits Conference (ISSCC), 2016 IEEE International*, pp. 264–265, IEEE, 2016.

[8] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[9] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in *ACM SIGARCH Computer Architecture News*, vol. 43, pp. 92–104, ACM, 2015.

[10] B. Moons, R. Uytterhoeven, W. Dehaene, and M. Verhelst, "14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi," in *Solid-State Circuits Conference (ISSCC), 2017 IEEE International*, pp. 246–247, IEEE, 2017.

[11] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 764–775, IEEE, 2018.

[12] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen, "Cambricon-x: An accelerator for sparse neural networks," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, p. 20, IEEE Press, 2016.

[13] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," *arXiv preprint arXiv:1709.01507*, vol. 7, 2017.

[14] M. Chiang and T. Zhang, "Fog and iot: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, 2016.

[15] T. X. Tran, A. Hajisami, and D. Pompili, "Cooperative hierarchical caching in 5g cloud radio access networks," *IEEE Network*, vol. 31, no. 4, pp. 35–41, 2017.

[16] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[17] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*, pp. 525–542, Springer, 2016.

[18] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," in *Advances in Neural Information Processing Systems*, pp. 345–353, 2017.

[19] J. Zhang, Z. Wang, and N. Verma, "In-memory computation of a machine-learning classifier in a standard 6t sram array.," *J. Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, 2017.

[20] K. Ando, K. Ueyoshi, K. Orimo, H. Yonekawa, S. Sato, H. Nakahara, S. Takamaeda-Yamazaki, M. Ikebe, T. Asai, T. Kuroda, *et al.*, "Brein memory: A single-chip binary/ternary reconfigurable in-memory deep neural network accelerator achieving 1.4 tops at 0.6 w," *IEEE Journal of Solid-State Circuits*, 2017.

[21] L. Jiang, M. Kim, W. Wen, and D. Wang, "Xnor-pop: A processing-in-memory architecture for binary convolutional neural networks in wide-io2 drams," in *Low Power Electronics and Design (ISLPED, 2017 IEEE/ACM International Symposium on*, pp. 1–6, IEEE, 2017.

[22] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A dram-based reconfigurable in-situ accelerator," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 288–301, ACM, 2017.

[23] O. Villa, D. R. Johnson, M. O'Connor, E. Bolotin, D. Nellans, J. Luitjens, N. Sakharnykh, P. Wang, P. Micikevicius, A. Scudiero, *et al.*, "Scaling the power wall: a path to exascale," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 830–841, IEEE Press, 2014.

[24] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pp. 10–14, IEEE, 2014.

[25] R. Liu, X. Peng, X. Sun, W.-S. Khwa, X. Si, J.-J. Chen, J.-F. Li, M.-F. Chang, and S. Yu, "Parallelizing sram arrays with customized bit-cell for binary neural networks," in *Proceedings of the 55th Annual Design Automation Conference*, p. 21, ACM, 2018.

[26] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 273–287, ACM, 2017.

[27] H. Kim, J. Sim, Y. Choi, and L.-S. Kim, "A kernel decomposition architecture for binary-weight convolutional neural networks," in *Proceedings of the 54th Annual Design Automation Conference 2017*, p. 60, ACM, 2017.

[28] Y. Li, Z. Liu, K. Xu, H. Yu, and F. Ren, "A 7.663-tops 8.2-w energy-efficient fpga accelerator for binary convolutional neural networks," in *FPGA*, pp. 290–291, 2017.

[29] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[30] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in *ACM SIGARCH Computer Architecture News*, vol. 44, pp. 367–379, IEEE Press, 2016.

[31] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, pp. 1135–1143, 2015.

[32] Y.-B. Kim and T. W. Chen, "Assessing merged dram/logic technology," *INTEGRATION, the VLSI journal*, vol. 27, no. 2, pp. 179–194, 1999.

[33] M. D. McDonnell and T. Vladusich, "Enhanced image classification with a fast-learning shallow convolutional neural network," *arXiv preprint arXiv:1503.04596*, 2015.

[34] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[35] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-3dd: Architecture-level modeling for 3d die-stacked dram main memory," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 33–38, EDA Consortium, 2012.

[36] P.-Y. Chen, X. Peng, and S. Yu, "Neurosim: A circuit-level macro model for benchmarking neuro-inspired architectures in online learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[37] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[38] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 675–678, ACM, 2014.

[39] A. Al Bahou, G. Karunaratne, R. Andri, L. Cavigelli, and L. Benini, "Xnorbin: A 95 top/s/w hardware accelerator for binary convolutional neural networks," in *Low-Power and High-Speed Chips (COOL CHIPS), 2018 IEEE Symposium in*, pp. 1–3, IEEE, 2018.