HyBNN: Quantifying and Optimizing Hardware Efficiency of Binary Neural Networks

GENG YANG and JIE LEI, Image Processing and Image Communication Laboratory, Xidian University, China
ZHENMAN FANG, School of Engineering Science, Simon Fraser University, Canada
YUNSONG LI, JIAQING ZHANG, and WEIYING XIE, Image Processing and Image Communication Laboratory, Xidian University, China

Binary neural network (BNN), where both the weight and the activation values are represented with one bit, provides an attractive alternative to deploy highly efficient deep learning inference on resource-constrained edge devices. However, our investigation reveals that, to achieve satisfactory accuracy gains, state-of-the-art (SOTA) BNNs, such as FracBNN and ReActNet, usually have to incorporate various auxiliary floating-point components and increase the model size, which in turn degrades the hardware performance efficiency. In this paper, we aim to quantify such hardware inefficiency in SOTA BNNs and further mitigate it with negligible accuracy loss. First, we observe that the auxiliary floating-point (AFP) components consume an average of 93% DSPs, 46% LUTs, and 62% FFs, among the entire BNN accelerator resource utilization. To mitigate such overhead, we propose a novel algorithm-hardware co-design, called *FuseBNN*, to fuse those AFP operators without hurting the accuracy. On average, FuseBNN reduces AFP resource utilization to 59% DSPs, 13% LUTs, and 16% FFs. Second, SOTA BNNs often use the compact MobileNetV1 as the backbone network but have to replace the lightweight 3×3 depth-wise convolution (DWC) with the 3×3 standard convolution (SC, e.g., in ReActNet and our ReActNet-adapted BaseBNN) or even more complex fractional 3×3 SC (e.g., in FracBNN) to bridge the accuracy gap. As a result, the model parameter size is significantly increased and becomes 2.25× larger than that of the 4-bit direct quantization with the original DWC (4-Bit-Net); the number of multiply-accumulate operations is also significantly increased so that the overall LUT resource usage of BaseBNN is almost the same as that of 4-Bit-Net. To address this issue, we propose *HyBNN*, where we binarize depth-wise separation convolution (DSC) blocks for the first time to decrease the model size and incorporate 4-bit DSC blocks to compensate for the accuracy loss. For the ship detection task in synthetic aperture radar imagery on the AMD-Xilinx ZCU102 FPGA, HyBNN achieves a detection accuracy of 94.8% and a detection speed of 615 frames per second (FPS), which is 6.8× faster than FuseBNN+ (94.9% accuracy) and 2.7× faster than 4-Bit-Net (95.9% accuracy). For image classification on the CIFAR-10 dataset on the AMD-Xilinx Ultra96-V2 FPGA, HyBNN achieves 1.5× speedup and 0.7% better accuracy over SOTA FracBNN.

CCS Concepts: • **Computing methodologies → Machine learning**; • **Hardware → Hardware accelerators**; **Hardware-software codesign**; • **Computer systems organization → Reconfigurable computing**; **Special purpose systems**.

Additional Key Words and Phrases: Binary Neural Network, Hardware Efficiency, FPGA Accelerator, Hybrid Design, Operator Fusion

Authors' addresses: Geng Yang, gengyang@stu.xidian.edu.cn; Jie Lei, jielei@mail.xidian.edu.cn, Image Processing and Image Communication Laboratory, Xidian University, China; Zhenman Fang, zhenman@sfu.ca, School of Engineering Science, Simon Fraser University, Canada; Yunsong li, ysli@mail.xidian.edu.cn; Jiaqing Zhang, jqzhang_2@stu.xidian.edu.cn; Weiying Xie, wyxie@xidian.edu.cn, Image Processing and Image Communication Laboratory, Xidian University, China.

1936-7406/2023/6-ART1 $15.00
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Geng Yang, Jie Lei, Zhenman Fang, Yunsong li, Jiaqing Zhang, and Weiying Xie. 2023. HyBNN: Quantifying and Optimizing Hardware Efficiency of Binary Neural Networks. *ACM Trans. Reconfig. Technol. Syst.* 1, 1, Article 1 (June 2023), 23 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

As an extreme model compression method, binary neural network (BNN) has recently presented a promising opportunity for deep learning inferences on resource-constrained edge devices. Using extreme data precision, i.e., 1-bit weight and 1-bit activation, BNN [4] not only significantly reduces the network memory footprint, but also trades massive multiply-accumulate (MAC) operations for much cheaper logical XNOR and population count (POPCNT) operations.

Despite BNN's attractive hardware-friendly properties, simply quantizing floating-point to binary representations for a BNN would damage its feature representation capability and lead to severe accuracy loss. For instance, the early-stage BNN called XNOR-Net [22] only achieves 51.2% top-1 accuracy on ImageNet classification [5], which leaves an 18% accuracy gap to its floating-point counterpart, ResNet-18 [9].

In the past few years, numerous studies have been carried out to narrow the accuracy gap between BNN and its corresponding floating-point convolutional neural networks (CNN) using various optimization strategies, including minimizing the quantization error, improving the network loss function, and reducing the gradient error [2, 15, 16, 18, 21, 26, 27, 29, 30, 34]. Promisingly, the latest ReActNet [15] and FracBNN [29] have achieved SOTA ResNet-18-level (69.4%) and MobileNetV2-level (71.8%) top-1 accuracy on ImageNet classification, respectively. In this paper (detailed in Section 2), we have adapted ReActNet [15] (called BaseBNN) for the ship detection task on synthetic aperture radar (SAR) imagery and achieved 94.9% detection accuracy, very close to its floating-point counterpart (96%).

Unfortunately, the accuracy improvements of these SOTA BNNs come at the cost of *various auxiliary floating-point (AFP) components* and *increased model size*, which harms their hardware efficiency when deployed on resource-constrained edge devices. ***The goal of this paper*** *is to quantify such hardware inefficiency in SOTA BNNs and further optimize the BNN hardware performance with negligible accuracy loss.*

Throughout this paper, we will mainly use our ReActNet-adapted BaseBNN for ship detection on SAR imagery as a case study, due to its high detection accuracy and representativeness of SOTA BNNs such as ReActNet and FracBNN. For the hardware efficiency analysis and optimization, we implement an optimized dataflow-style accelerator architecture for BaseBNN—where all network layers are streamlined on the FPGA chip, similar to SOTA BNN acceleration framework FINN [3, 25] and ultra low-bit CNN acceleration framework OSCAR-RT [28]—on the AMD-Xilinx ZCU102 embedded FPGA platform. The detailed setup is given in Section 2.

Inefficiency of auxiliary floating-point (AFP) components and our *FuseBNN* solution (Section 3): To boost the feature representation capability and thus the inference accuracy, SOTA BNNs have incorporated a variety of AFP operators as summarized in Fig. 1 (a) and (b) (in Section 3.1), including shortcut branch [16, 27], batch normalization (BatchNorm) [12, 21], biased parametric rectified linear unit (BPReLU) [15, 30], and biased sign function (BSign) [15, 29]. As a result, the entire BNN network works on floating-point operations except the binary convolution, which leads to significant hardware performance overhead. Our experiments show that, on average, these cascaded AFP components consume about 93% of DSPs, 46% of LUTs, and 62% of FFs, among the total resource utilization of the BNN accelerator on the FPGA. For about half of the BNN layers, they consume more than 93% of DSPs, 60% of LUTs, and 76% of FFs, significantly more than the resources that the actual binary convolutions consume.

To mitigate such hardware performance overhead, we propose *FuseBNN*, where we fuse those AFP operators into one threshold unit that includes only one multiplication, three additions, and several logical decisions. This is done via algorithmic transformations without hurting the accuracy. However, due to the AFP shortcut branch, the computation of the current fused unit depends on the intermediate floating-point result (e.g., from BPReLU) in the prior fused unit, thus preventing the complete fusion of all AFP operations. To address this challenge, we make a slight change in the network structure to connect the shortcut branch directly from the first BatchNorm operator in the prior layer to the addition operator in the next layer and fuse the first BatchNorm operator into the next layer (shown in Fig. 4(a)-(b) and Fig. 5, and detailed in Section 3.3). Lastly, we also quantize the fused operators into 24-bit integer precision and implement them on the FPGA. Experimental results show that FuseBNN slightly increases the model accuracy from 94.9% (BaseBNN) to 95% and significantly reduces the AFP resource utilization down to an average of 59% of DSPs, 13% of LUTs, and 16% of FFs.

Inefficiency of increased model size and our *HyBNN* solution (Section 4): In addition to AFP components, another essential strategy in SOTA BNNs to improve the model accuracy is to increase the scale of the binary convolution [2, 15, 27, 29]. Specifically, MobileNetV1 [10] is widely adopted as the backbone network in SOTA BNNs such as ReActNet [15], FracBNN [29], and our BaseBNN, due to its compact network structure. However, when MobileNetV1 is directly binarized with the aforementioned AFP components, there is a significant accuracy drop. For example, in our case study, the directly binarized MobileNetV1-SAR model (called BaseBNN-DWC) only achieves 88% detection accuracy, leaving a 8% accuracy gap to its floating-point counterpart. This is because the binarization of the highly sparse depth-wise separable convolution (DSC) in MobileNetV1 seriously hurts its feature extraction capability.

To compensate for the accuracy loss, SOTA BNNs increase the scale of the binary convolution. For example, ReActNet [15] and our BaseBNN replace the 3×3 depth-wise convolution (DWC) in the DSC block with the 3×3 standard convolution (SC). This boosts the detection accuracy of BaseBNN to 94.9%. Similarly, FracBNN [29] replaces the 3×3 DWC in the DSC block with a more complex fractional 3×3 SC that consists of a base 3×3 SC and a conditional 3×3 SC. The adapted FracBNN achieves 95.1% detection accuracy in our case study, uses the same amount of model parameters but 1.5× more MAC operations than BaseBNN. This is why we choose to adapt our BaseBNN based on ReActNet.

Unfortunately, such accuracy optimization also leads to the model size increase from two perspectives. First, the model parameter size is increased by 6× from 0.15MB (BaseBNN-DWC) to 0.9MB (BaseBNN), which is 2.25× larger than that of the 4-bit direct quantization with the original DWC (4-Bit-Net: 0.4MB parameters, 95.9% accuracy). Second, the number of MAC operations is increased by 8.5× from 2.5GOP (giga operations, BaseBNN-DWC and 4-Bit-Net) to 21.2GOP (BaseBNN). Although each binary operation is much cheaper than a 4-bit operation, the overall LUT resource usage of BaseBNN becomes almost the same as that of 4-Bit-Net. **In summary, BaseBNN loses its advantage over 4-Bit-Net.**

To avoid this excessive increase in BNN model size, we propose *HyBNN*, where we use a hybrid of FuseBNN-DWC+ blocks to keep the model size small with depth-wise convolution (and other common optimizations) and 4-Bit-Net blocks to compensate the accuracy loss. For the first time, we directly binarize the DSC blocks in the MobileNetV1 backbone to keep its compact model size. To compensate for the accuracy loss, we use the 4-Bit-Net blocks—which are also quantized on the original DSC blocks—to replace the early network layers. Via a design space exploration, we find that, using 6 4-Bit-Net blocks in the upper network and 5 FuseBNN-DWC+ blocks in the down network, our *HyBNN-U6D5* achieves the best accuracy and performance tradeoff. Overall, HyBNN-U6D5 achieves a detection accuracy of 94.8% and a detection speed of 615 FPS, which is 6.8× faster than FuseBNN+ (94.9% accuracy) and 2.7× faster than 4-Bit-Net (95.9% accuracy).

Generality of our analysis and optimizations (Section 5): To validate the generality of our analysis and optimizations, we also apply them to the image classification task on the CIFAR-10 dataset using a different BNN network, HyBNN-U5D3, and a different FPGA platform, AMD-Xilinx Ultra96-V2 FPGA platform. Our *HyBNN-U5D3* achieves 89.8% top-1 accuracy and 4,302 FPS classification speed, which is 0.7% more accurate and 1.5× faster than SOTA FracBNN [29], both running on the same AMD-Xilinx Ultra96-V2 FPGA platform.

In summary, this paper makes the following contributions:

- A quantitative evaluation of hardware inefficiency caused by auxiliary floating-point components and increased model size in SOTA BNNs for compensating the accuracy.
- FuseBNN, a novel algorithm/hardware co-design to fuse AFP operators in BNNs without hurting the accuracy.
- HyBNN, the first hybrid BNN (FuseBNN) and 4-Bit-Net design that directly binarizes (and quantizes) the original DSC blocks to keep compact model size and high model accuracy.
- Experimental results to demonstrate superior hardware performance and promising model accuracy of HyBNN for ship detection on SAR imagery (94.8% detection accuracy and 615 FPS on ZCU102 FPGA) and image classification on CIFAR-10 (89.8% top-1 accuracy and 4,302 FPS on Ultra96-V2 FPGA).

The rest of this paper is organized as follows. Section 2 illustrates the baseline BNN and detailed experimental settings for our primary case study of SAR ship detection, serving as a starting point for the quantitative evaluation of the algorithmic performance and hardware efficiency of SOTA BNNs. Sections 3 and Sections 4 analyze the hardware inefficiency caused by AFP components and increased model size, and provide our algorithm-hardware co-design solutions to address these challenges. Section 5 evaluates the generalization of our proposed solutions on the CIFAR-10 image classification task. Section 6 reviews related work on model design and FPGA acceleration of BNNs. Finally, we conclude this paper in Section 7.

2 BASELINE BNN AND EXPERIMENTAL SETUP

2.1 Baseline BNN for Ship Object Detection in SAR Imagery

In this paper, we choose the ship detection task in SAR imagery as our major case study, where BNN is able to achieve a high detection accuracy of 94.9%, which is much higher than that in ImageNet classification (69.4% and 71.8% for SOTA ReActNet [15] and FracBNN [29]). To ensure our analysis and optimizations for BNNs are representative for SOTA BNNs, we choose SOTA BNN ReActNet [15] as our baseline and adapt it to achieve a high detection accuracy for SAR image ship detection; the adapted ReActNet model is called BaseBNN. In Section 4.1, we will compare it to the adapted version based on FracBNN [29] and explain why we use the ReActNet-based BaseBNN throughout this paper.

In BaseBNN, the compact MobileNetV1 structure [10] is retained as the backbone. Shown in Table 1, MobileNetV1 uses depth-wise separation convolution (DSC), which is composed of 3×3 depth-wise convolution (DWC) and point-wise convolution (i.e., 1×1 standard convolution (SC)). Compared to the 3×3 SC, this reduces the computation by 8× to 9× while imposing minor impact on accuracy. Note that we employ the model adapting strategy proposed in OSCAR-RT[28] to adjust the number of channels and layers of the original MobileNetV1 to better detect small and clustered ship targets in SAR imagery, called MobileNetV1-SAR. As listed in Table 1, in addition to the input SC layer (the first row) and the output detection SC layer (the second last row), the backbone of MobileNetV1-SAR is composed of 11 DSC blocks, where each DSC block includes a

Table 1. Configuration of backbone network MobileNetV1-SAR. Each row describes a sequence of n (last column) repeated identical layers. The first column shows the input feature map size for each operator (second column). c and s denote the number of output channels and stride of each operator.

Input	Operator		c	s	n
416×416×1	3×3 SC		32	2	1
208×208×32	DSC	3×3 DWC	32	1	1
		1×1 SC	64	1	
208×208×64	DSC	3×3 DWC	64	2	1
		1×1 SC	128	1	
104×104×128	DSC	3×3 DWC	128	1	1
		1×1 SC	128	1	
104×104×128	DSC	3×3 DWC	128	2	1
		1×1 SC	256	1	
52×52×256	DSC	3×3 DWC	256	1	5
		1×1 SC	256	1	
52×52×256	DSC	3×3 DWC	256	1	1
		1×1 SC	512	1	
52×52×512	DSC	3×3 DWC	512	1	1
		1×1 SC	512	1	
52×52×512	1×1 SC		25	1	1
52×52×25	detector		-	-	-

3×3 DWC and a 1×1 SC. The detector (the last row) employs the YOLOv2 back-end which uses five k-means clustering anchor boxes[23].

To compensate for the accuracy loss encountered by extreme data precision, each DSC block in the BaseBNN backbone is replaced by a normal or downsample binary SC block shown in Fig. 1 (a) and (b), depending on whether the numbers of input and output channels of the DSC block are equal. First, both the normal and downsample SC blocks incorporate various AFP components, which will be detailed and analyzed in Section 3. Second, both SC blocks replace the 3×3 DWC with a 3×3 SC to increase the model size, which will be detailed and analyzed in Section 4. Moreover, for the downsample SC block, similar to FracBNN [29], we replace the original activation duplicate in ReActNet [15] with a channel duplicate to reduce the 1×1 SC computation. As shown in Table 2, the final adapted BaseBNN achieves a high average precision of 94.9%, comparable to the corresponding floating-point version MobileNet-SAR (96%) in our case study.

Table 2. Accuracy of ship detection on SAR image dataset SSDD.

Model	AP (%)
MobileNetV1-SAR (floating-point)	96
BaseBNN	94.9

2.2 All On-Chip Dataflow-Style BNN Accelerator on FPGA

The extreme 1-bit weights and activations in BNNs facilitate the highly-efficient hardware implementation of all the network layers on-chip, thus eliminating the expensive off-chip data movement between layers. Inspired by SOTA BNN accelerator design FINN [3, 25] and low-bit CNN accelerator design OSCAR-RT [28], this paper implements all the studied networks on FPGA using the

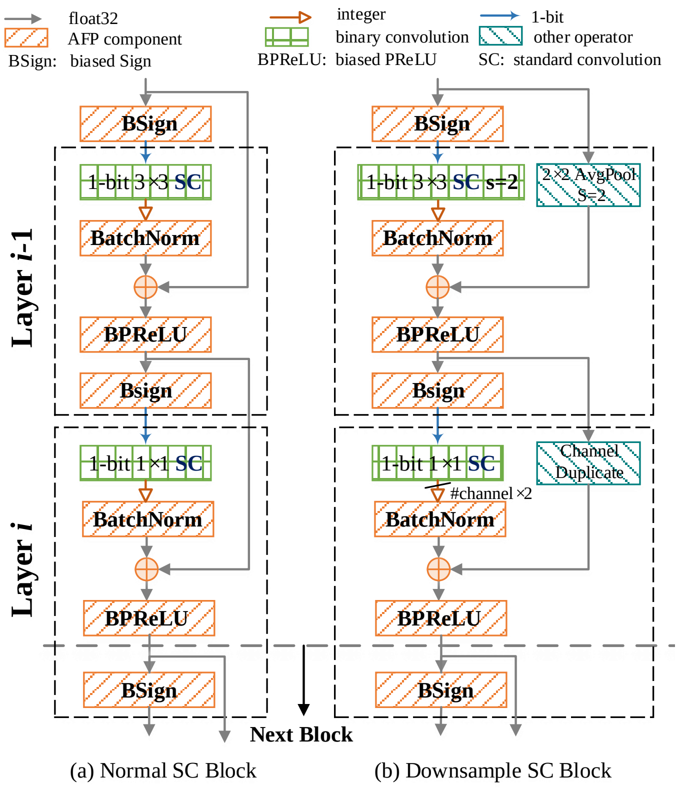

Fig. 1. Basic binary standard convolution (SC) blocks in BaseBNN, which replaces each DSC block in MobileNetV1-SAR (Table 1) with (a) or (b). When the numbers of input and output channels of the DSC block are equal, the normal SC block is used; otherwise, the downsample SC block is used.

all-on-chip dataflow-style accelerator design, which maps all layers of the model to the FPGA chip and executes all layers on-chip concurrently in the deeply pipelined streaming fashion. We leave the detailed hardware implementation of each layer to Section 3.3.3 and 4.4.1. At a high level, for each on-chip convolution layer, the parallelism unrolled on the input channel dimension of the input feature map (Inp) and the parallelism unrolled on the output channel dimension of the output feature map (Oup) can be flexibly configured by the interlayer matching strategy proposed in [28]. This fine-grained layer-by-layer customization allows us to set appropriate parallelism for layers of different types and sizes to achieve the specified overall dataflow throughput.

2.3 Detailed Experimental Setup

Dataset. For our case study, we use the widely used SAR ship detection dataset (SSDD) constructed by Li *et al.* [13]. Each input image in SSDD is a resized grayscale SAR image with a size of 416×416. The widely used average precision (AP) based on an intersection over union (IOU) of 0.5 is applied to quantity the ship detection accuracy. The division of training set and test set in SSDD is the same as that in [28].

Model training. We use Pytorch [20] to specify all models and training scripts on the Nvidia RTX 2080Ti and 3090 GPUs. Adam optimizer with a linear learning rate decay scheduler is used. The initial learning rate of all models is set to $5e^{-4}$. The batch size and number of epochs are 10 and 1000, respectively.

For all BNN training, the standard two-step training strategy [18] is applied. First, the activation is binarized while the weight remains floating-point. Next, the model is initialized by weights obtained in step one, and the weights and activation are binarized. The training settings are identical for both steps, except that the weight decay is $1e^{-5}$ in step one and 0 in step two. For additional 4-Bit-Net, it inherits the floating-point weights of MobileNetV1-SAR to accelerate convergence and adopts uniform quantization aware training [33] to reduce the precision of weight and activation to 4-bit. Note that prior studies have shown that no less than 4-bit quantization precision can achieve comparable accuracy to floating-point models with appropriate training strategies [6, 24, 28].
Hardware setup. For hardware deployment and analysis, we implement the hardware accelerator designs in AMD-Xilinx Vitis HLS and run them on the AMD-Xilinx ZCU102 embedded FPGA board. The board has 274,080 LUTs, 548,160 FFs, 912 BRAMs, and 2,520 DSPs. The hardware architectures are synthesized, implemented, and tested using AMD-Xilinx Vitis HLS, Vivado, and Vitis version 2020.2. Hardware resource utilization is obtained from the Vivado post-implementation report. All the designs run at 300MHz unless otherwise specified.

3 ANALYSIS OF VARIOUS AUXILIARY FLOATING-POINT COMPONENTS AND OUR SOLUTION FUSEBNN

3.1 Auxiliary Floating-Point (AFP) Components

Summarized in Fig. 1 (a) and (b), to improve BNN accuracy, various AFP components have been incorporated into SOTA BNNs.
Shortcut branch. Since the extreme 1-bit precision inevitably shrinks the expression range of data, the shortcut branch is proposed in Bi-Real Net [16] to connect and add the current 32-bit activation to the activation of the following block to enhance the representation capability. Due to its simplicity yet effectiveness, the shortcut branch and its variants have been widely adopted in most SOTA BNNs[2, 11, 18, 27].
Batch normalization (BatchNorm). Prior work has demonstrated that BatchNorm [12] helps BNNs smoothly improve convergence and generalization, from the theoretical and empirical perspectives [17]. It can be described as follows:

$$y_j = k_j x_j + b_j \tag{1}$$

$$k_j = \frac{\gamma_j}{\sqrt{||\sigma^2{}_j + \varepsilon||}}, \quad b_j = \beta_j - \frac{\gamma_j \mu_j}{\sqrt{||\sigma^2{}_j + \varepsilon||}} \tag{2}$$

x_j and y_j denote the input and output of the j-th channel in BatchNorm layer. μ_j and σ_j are the running mean and running variance required to normalize the output feature map. γ_j and β_j are the learning scale and shift parameters. Note that k_j and b_j can be pre-computed once a network is trained.
Biased parametric rectified linear unit (BPReLU) and biased sign (Bsign). As the accuracy is sensitive to the activation distribution variations, BPReLU and Bsign are proposed in ReActNet [15] to adopt channel-wise reshaping and shifting to obtain a more enriched and subtle distribution of the feature map in a learnable fashion. They are defined as:

$$BPReLU(m_j) = \begin{cases} m_j + \phi_j + \xi_j & \text{if } m_j > -\phi_\text{j} \\ \lambda_j(m_j + \phi_j) + \xi_j & \text{if } m_j \leq -\phi_\text{j} \end{cases} \tag{3}$$

where m_j represents the input for BPReLU on the j-th channel. ϕ_j and ξ_j are learning bias. λj is a learning coefficient that controls the slope of the negative part. Similarly, PokeBNN [30] employs a variant of BPReLU—dynamic PReLU, which supplements the learnable slope for the positive part.

Algorithm 1: Hardware implementation pseudocode for cascaded AFP components to obtain the 1-bit output activation of one convolution layer

```
Input: the XNOR-POPCNT integer result of the previous convolution layer: $in$; the BPReLU
       floating-point result form the shortcut branch: $in\_shortcut$; the total number of
       input pixels for one output activation: $N$
Output: the 1-bit output activation of the current convolution layer: $res$
  // parameters for each layer, defined in Section 3.1
  float $k$, $b$, $\phi$, $\lambda$, $\xi$, $\omega$;
  // step 1: obtain the actual result of binary convolution after bitwise XNOR-POPCNT
 operations
  int $sc\_res = 2 \times in - N$;
  // step 2: BatchNorm with Eq. (1)
  float $tmp = k \times sc\_res + b$;
  // step 3: shortcut-add
  $tmp = tmp + in\_shortcut$;
  // step 4: BPReLU with Eq. (3)
  $tmp = tmp + \phi$;
  if $(tmp \leq 0)$ $tmp = tmp \times \lambda$;
  $tmp = tmp + \xi$ ;
  // step 5: Bsign with Eq. (4)
  $tmp = tmp + \omega$;
  if $(tmp \leq 0)$ $res = 0$;
  else $res = 1$;
  return $res$;
```

$$x_j^b = Bsign(x_j^r) = \begin{cases} +1 & \text{if } x_j^r > -\omega_\text{j} \\ -1 & \text{if } x_j^r \leq -\omega_\text{j} \end{cases} \tag{4}$$

where x_j^r and x_j^b are real-value input and binary output on the j-th channel. ω_j is a learning bias for the threshold.

3.2 Impact on Hardware Performance

As shown in Fig. 1 (a) and (b), these AFP components keep the entire BNN network working on floating-point operations except the binary convolutions. To assess their actual hardware cost, we implement them on FPGA using Vitis HLS [1]. Algorithm 1 shows the pseudocode to implement the four cascaded AFP operators presented in Section 3.1. Similar to FINN [3, 25], the binary convolutions are implemented using logic XNOR and POPCNT operations. After the intermediate results (32-bit integer) are obtained from a binary convolution layer (step 1), it performs BatchNorm, shortcut add, BPReLU, and Bsign (steps 2 to 5) all in floating-point, to get the final 1-bit output activation. Eq. (2) is not calculated since k_j and b_j are pre-computed once a network is trained. We call Algorithm 1 as a floating-point threshold unit (TU), which is integrated with each binary convolution layer. Fig. 2 reports the FPGA resource consumption percentage of the floating-point TUs in each binary convolution layer of BaseBNN (shown in Table 1, excluding the first and last SC layer). On average, these floating-point TUs consume about 46% of LUTs, 62% of FFs, and 93% of DSPs, among the entire BNN accelerator resource utilization. For about half of the layers, they

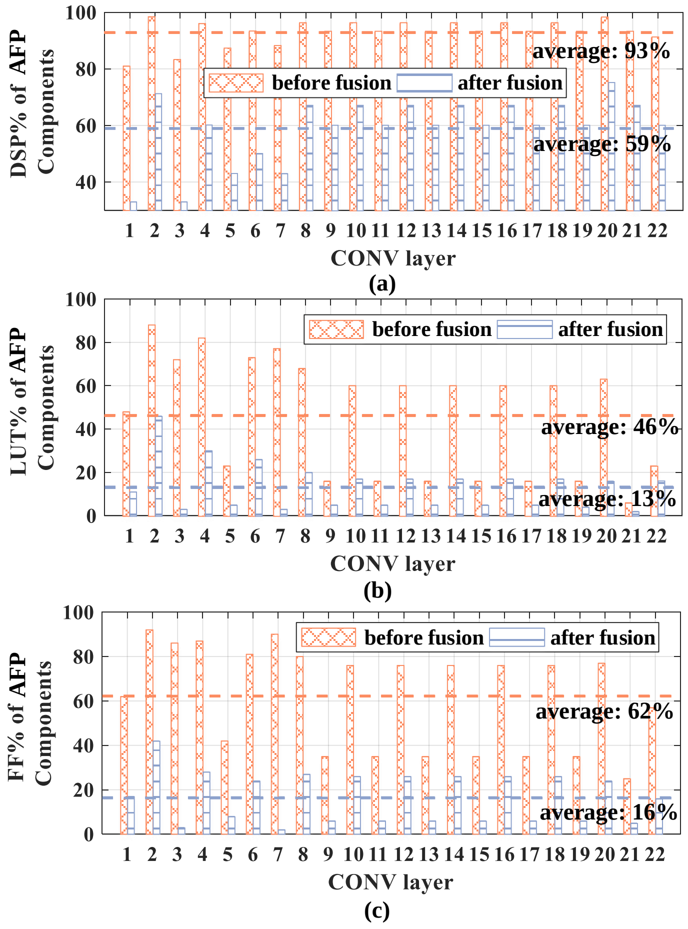

Fig. 2. Percentage of AFP component resource consumption in each convolution layer of BaseBNN's (before fusion) and FuseBNN's (after fusion) binary SC blocks.

consume more than 60% of LUTs, 76% of FFs, and 93% of DSPs, significantly more than the resources that those actual binary convolutions consume.

3.3 FuseBNN: AFP Component Fusion for BNN

To reduce the hardware resource usage of AFP components and reserve more resource for binary convolutions, and thus to improve the overall BNN performance, we propose FuseBNN to fuse those AFP components.

3.3.1 Challenges in AFP Component Fusion. In the widely used BNN acceleration framework FINN [3, 25], it proposed a simple fusion technique to accelerate early-stage BNNs (with low accuracy) that only include floating-point BatchNorm and sign operations, shown in Fig. 3.

The fused BatchNorm and sign equation is as follows:

$$o_{i,j} = \begin{cases} 1 & if\ a_{i,j} > T_{i,j} \\ 0 & otherwise \end{cases}, \quad T_{i,j} = -\frac{b_{i,j}}{2k_{i,j}} + \frac{N_i}{2} \tag{5}$$

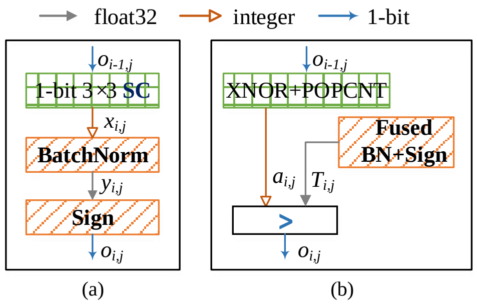

Fig. 3. Network structure (a) and hardware implementation (b), using a simple floating-point component fusion technique, for binary convolution blocks in the early-stage BNNs (FINN [3, 25]) with low accuracy.

where $o_{i,j}$ denotes the 1-bit output activation on the j-th output channel of the i-th layer L_i. $a_{i,j}$ indicates the intermediate binary convolution (i.e., XNOR and POPCNT) result. $T_{i,j}$ is called the threshold value. N_i denotes the total number of input activations ($o_{i-1,j}$) required to calculate one output activation. The definitions of $k_{i,j}$ and $b_{i,j}$ are described in Eq. (2). As N_i, $k_{i,j}$ and $b_{i,j}$ are constant once a network is trained, $T_{i,j}$ can be pre-computed, which eliminates floating-point operations.

As illustrated in Eq. (5), *the unique sign function gives the early-stage BNNs an ingenious fusion opportunity* to obtain the binary output activation ($o_{i,j}$) by a simple on-chip comparison between the intermediate result of XNOR-POPCNT ($a_{i,j}$) and the channel-wise threshold coefficient ($T_{i,j}$).

However, this simple fusion strategy no longer works well in SOTA BNNs, due to the introduction of the AFP shortcut branch for accuracy gains. Take our BaseBNN as an example, shown in Fig. 1(a), we regard the part in the dotted box as a convolution layer. To obtain the binary output activation of the current layer i, it first needs to perform an floating-point addition between the current BatchNorm and the result of BPReLU in layer i-1. Therefore, it is inevitable to perform the expensive floating-point BPReLU computation in layer i-1 and buffer its result on-chip, thus losing the opportunity of offline fusion with the Bsign function. Similarly, the floating-point shortcut branches in the downsample SC block shown in Fig. 1(b) also impede efficient floating-point component fusion.

3.3.2 Hardware-Friendly Fusion Algorithm. To address the fusion challenge, we first make slight changes in the binary SC blocks shown in Fig. 4(a) and (b), without hurting the accuracy. Specifically, for the normal SC block, the starting position of the floating-point shortcut branch is moved to right after the BatchNorm in the previous layer (the bold line in Fig 4(a)). Then all AFP components in layer i, together with the BatchNorm in layer i-1, in the yellow shadow box, are fused. The fusion strategy is the same for layer i-1.

In the downsample SC block, the starting position of the floating-point shortcut branch in layer i is moved to right after BatchNorm in layer i-1 (the bold line in Fig 4(b)). Moreover, we change the shortcut branch in layer i-1 into an integer shortcut branch, by changing the input of AvgPool (average pooling) from the BPReLU result to the 1×1 SC result and adding an integer BatchNorm after AvgPool. Then, all AFP components in layer i, together with AvgPool and two BatchNorms in layer i-1, in the yellow shadow box, are fused. For fusion in layer i-1, we fuse all its AFP components and there is no floating-point shortcut branch.

Next we explain how algorithmic transformations are done to fuse AFP components. For the sake of simplicity, we take the updated normal SC block (the yellow shadow box of Fig 4(a)) as an example for a detailed description. The downsample SC block can be derived in a similar way.

Assume $a_{i-1,j}$ and $a_{i,j}$ denote the XNOR-POPCNT results on the j-th output channel in the i-1-th and i-th SC layers L_{i-1} and L_i. Firstly, the actual MAC result of the 1-bit SC in L_{i-1} and L_i, $x_{i-1,j}$

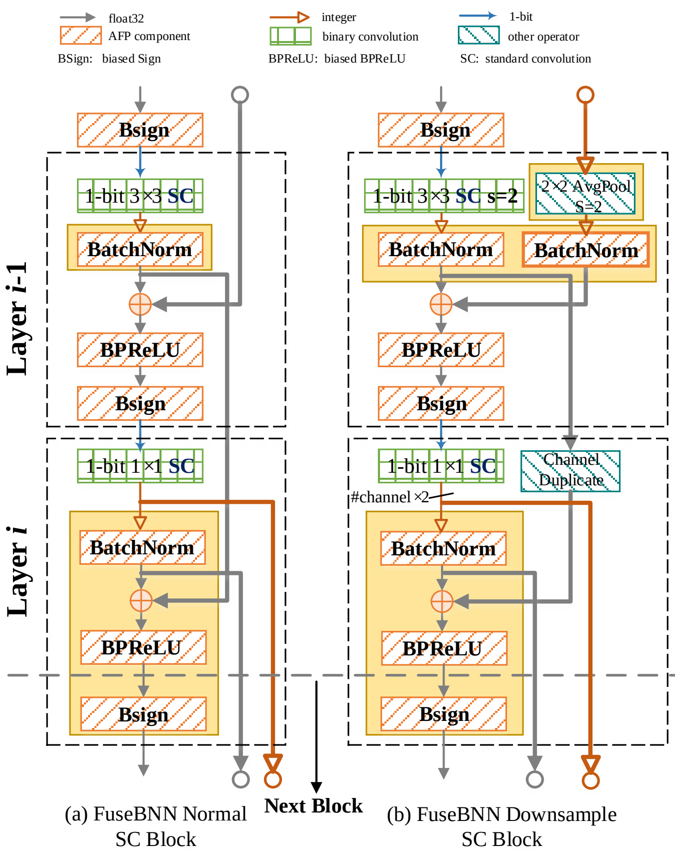

Fig. 4. Our proposed FuseBNN with updated binary SC blocks to support hardware-friendly AFP component fusion.

and $x_{i,j}$, can be calculated as:

$$x_{i-1,j} = 2a_{i-1,j} - N_{i-1}, \quad x_{i,j} = 2a_{i,j} - N_i \tag{6}$$

where N_{i-1} and N_i denote the total number of input activations to calculate one output activation for L_{i-1} and L_i.

Shown in Fig 4(a), the following AFP operators are then performed to obtain the binary output activation $o_{i,j}$ for L_i. All coefficients are defined in Eq. (1)-(4).

BatchNorm operator for L_{i-1} and L_i:

$$y_{i-1,j} = k_{i-1,j}x_{i-1,j} + b_{i-1,j}, \quad y_{i,j} = k_{i,j}x_{i,j} + b_{i,j} \tag{7}$$

Point-wise shortcut branch addition operator for L_i:

$$s_{i,j} = y_{i-1,j} + y_{i,j} \tag{8}$$

BPReLU operator for L_i:

$$r_{i,j} = \begin{cases} s_{i,j} + \phi_{i,j} + \xi_{i,j} & \text{if } \mathrm{s_{i,j}} > -\phi_{\mathrm{i,j}} \\ \lambda_{i,j}(s_{i,j} + \phi_{i,j}) + \xi_{i,j} & \text{if } \mathrm{s_{i,j}} \leq -\phi_{\mathrm{i,j}} \end{cases} \tag{9}$$

BSign operator for L_i:

$$o_{i,j} = \begin{cases} +1 & \text{if } \mathrm{r_{i,j}} > -\omega_{\mathrm{i,j}} \\ -1 & \text{if } \mathrm{r_{i,j}} \leq -\omega_{\mathrm{i,j}} \end{cases} \tag{10}$$

Benefiting from the aforementioned network changes, we can fuse Eq. (6)-(10) into:

$$o_{i,j} = \begin{cases} 1 & \text{if } ((\text{cond1}\&\text{cond2}) \mid (\text{cond3}\&\text{cond4})) \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

$$\begin{aligned} \text{cond1}: &\quad \mathrm{a_{i,j}} > \eta_{\mathrm{i,j}}\mathrm{a_{i-1,j}} + \theta_{1_{\mathrm{i,j}}} \\ \text{cond2}: &\quad \mathrm{a_{i,j}} > \eta_{\mathrm{i,j}}\mathrm{a_{i-1,j}} + \theta_{2_{\mathrm{i,j}}} \\ \text{cond3}: &\quad \mathrm{a_{i,j}} \leq \eta_{\mathrm{i,j}}\mathrm{a_{i-1,j}} + \theta_{1_{\mathrm{i,j}}} \\ \text{cond4}: &\quad \mathrm{a_{i,j}} > \eta_{\mathrm{i,j}}\mathrm{a_{i-1,j}} + \theta_{3_{\mathrm{i,j}}} \end{aligned} \tag{12}$$

where $\eta_{i,j}$, $\theta_{1_{i,j}}$, $\theta_{2_{i,j}}$ and $\theta_{3_{i,j}}$ are four constant coefficients and can be pre-computed as:

$$\begin{aligned} \eta_{i,j} &= -\frac{k_{i-1,j}}{k_{i,j}} \\ \theta_{1_{i,j}} &= \frac{N_{i,j}}{2} + \frac{k_{i-1,j}N_{i-1,j} - b_{i-1,j} - b_{i,j} - \phi_{i,j}}{2k_{i,j}} \\ \theta_{2_{i,j}} &= \frac{N_{i,j}}{2} + \frac{k_{i-1,j}N_{i-1,j} - b_{i-1,j} - b_{i,j} - \phi_{i,j} - \xi_{i,j} - \omega_{i,j}}{2k_{i,j}} \\ \theta_{3_{i,j}} &= \frac{N_{i,j}}{2} + \frac{k_{i-1,j}N_{i-1,j} - b_{i-1,j} - b_{i,j} - \phi_{i,j}}{2k_{i,j}} - \frac{\xi_{i,j} + \omega_{i,j}}{2\lambda_{i,j}k_{i,j}} \end{aligned} \tag{13}$$

Note that the signs of $k_{i,j}$ and $\lambda_{i,j}$ are assumed to be positive, and the negative sign case can be similarly deduced.

As shown in Eq. (11) and (12), after the fusion, to get the binary output activation $o_{i,j}$, we only need two variables $a_{i-1,j}$ and $a_{i,j}$, i.e., the XNOR-PONCNT results from the prior and current SC layers; all other coefficients $\eta_{i,j}$, $\theta_{1_{i,j}}$, $\theta_{2_{i,j}}$ and $\theta_{3_{i,j}}$ are pre-computed constants for a network. Moreover, we quantize all these floating-point numbers into 24-bit integers to further reduce their resource usage.

3.3.3 Hardware Implementation of Fused Threshold Unit. Fig. 5 describes the detailed hardware structure of the normal SC block on FPGA after fusion; the hardware structure of the downsample SC block is similar and omitted. Similar to FINN [3, 25], the 3×3 SC and 1×1 SC are implemented using XNOR-POPCNT; their results are buffered as inputs for the fused threshold unit (TU, highlighted in the yellow box of Fig. 5 (b)), which implements Eq. (11) and (12). All data use 24-bit integer precision by shifting $a_{i,j}$ and pre-shifting all constant coefficients in Eq. (13). The proposed TU only needs one multiplication (for $\eta_{i,j} \times a_{i-1,j}$), three additions (addition with $\theta_{1_{i,j}}$, $\theta_{2_{i,j}}$, $\theta_{3_{i,j}}$), and several logical decisions, all in integer precision.

3.4 Accuracy and Hardware Performance of FuseBNN

Accuracy of FuseBNN. Our FuseBNN does not hurt the accuracy and actually slightly increases the accuracy from 94.9% (in BaseBNN) to 95%. This is because FuseBNN still retains all AFP components to enrich the expression of the binary network.
Hardware resource of single TU. As shown in Table 3, the DSPs, FFs and LUTs consumed by our fused TU are reduced by 5×, 20.5× and 6.5× compared to the one before fusion.

Table 3. Resource utilization of one threshold unit (TU) before and after AFP component fusion on ZCU102 with 300MHz clock frequency.

Threshold Unit	DSP	FF	LUT
Before fusion	5	820	517
After fusion	1	40	80

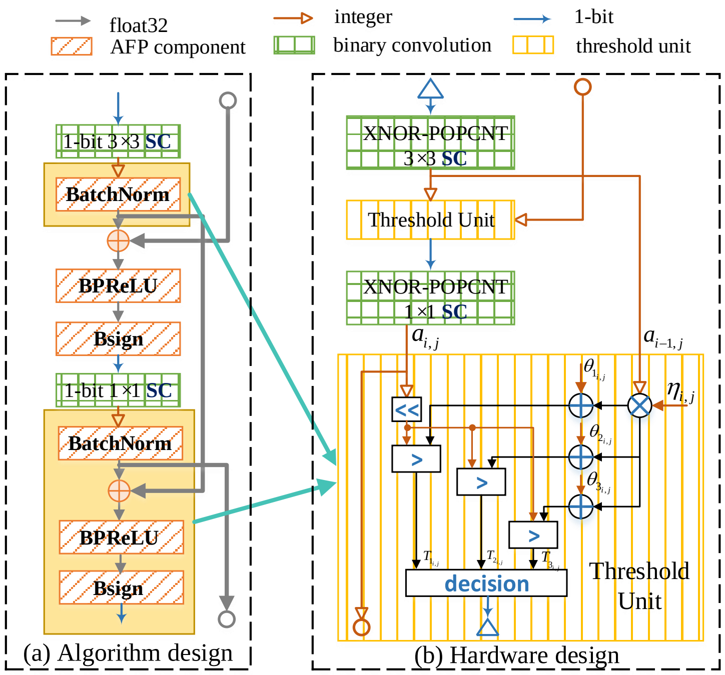

Fig. 5. Hardware implementation of the normal SC block in FuseBNN.

Hardware resource % of fused TU in each SC layer. Fig. 2 shows the percentage of LUTs, FFs and DSPs consumed by our fused TU in each SC layer of FuseBNN. Compared with BaseBNN, on average, these percentage numbers are significantly reduced down to 13%, 16% and 59%.

4 ANALYSIS OF INCREASED MODEL SIZE AND OUR SOLUTION HYBNN

4.1 Model Size Evaluation of SOTA BNNs

Another essential strategy to compensate for the accuracy loss caused by binary representations is to increase the BNN model size. In general, the compact MobileNetV1 is widely used as the backbone in SOTA BNNs, such as ReActNet [15], FracBNN [29], and our BaseBNN. Since MobileNetV1 uses the compact depth-wise separable convolution (DSC) block, which is composed of 3×3 depth-wise convolution (DWC) and 1×1 standard convolution (SC), it requires 8× to 9× less computation than one 3×3 SC. However, as shown in Fig. 6, if we apply direct binarization to the compact MobileNetV1-SAR with the 3×3 DWC in the DSC block (called BaseBNN-DWC), the detection accuracy drops to 88%.

To improve the accuracy, similar to ReActNet [15], the basic binary SC blocks in BaseBNN replaces the 3×3 DWC with the 3×3 SC to increase the model size, as shown in Fig. 1 and discussed in Section 2.1. Compared with BaseBNN-DWC, BaseBNN improves the accuracy by 6.9% to 94.9%; However, it also increases the number of operations (GOP) and parameter size (MB) by 8.5× and 6×, shown in Fig. 6.

Comparison to FracBNN [29]. Similarly, FracBNN replaces the 3×3 DWC with a more complex fractional 3×3 SC that consists of a base 3×3 SC and a conditional 3×3 SC; it also replaces the 1×1 SC with a fractional 1×1 SC. As a result, FracBNN brings the detection accuracy to 95.1%, slightly higher than 94.9% of BaseBNN. However, FracBNN requires 1.5× more operations than BaseBNN. This is why we choose to adapt our BaseBNN based on ReActNet for our case study.

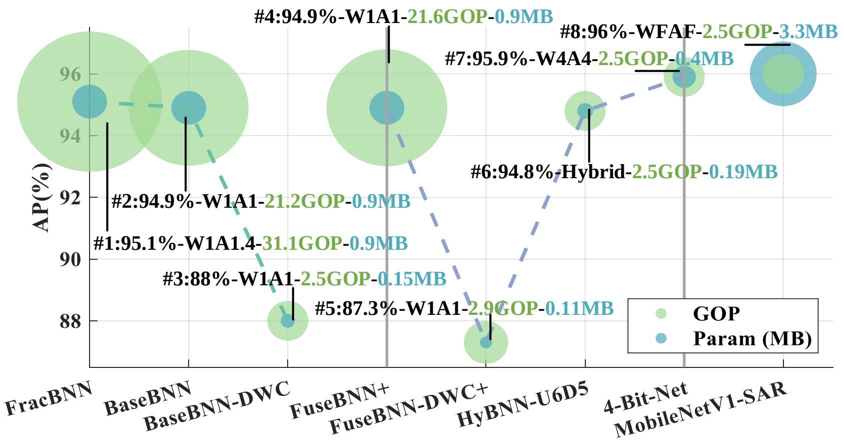

Fig. 6. Accuracy, operation number (GOP, giga multiply-accumulate operations, *not considering bit-width*) and parameter size (MB) comparison for different models. For example, 94.9%-W1A1-21.2GOP-0.9MB for BaseBNN denotes 94.9% accuracy, 1-bit weight and 1-bit activation, 21.2GOP, and 0.9MB parameters. WFAF denotes 32-bit floating-point weight and activation.

Losing advantage over 4-Bit-Net. Unfortunately, due to the model size increase, the model parameter size of BaseBNN (0.9MB), is actually 2.25× larger than that of the 4-bit direct quantization with the original DWC (called 4-Bit-Net in Fig. 6: 0.4MB parameters, 95.9% accuracy). Moreover, the number of operations for BaseBNN (21.2GOP) is 8.5× larger than that of 4-Bit-Net (2.5GOP). Although each binary operation is much cheaper than a 4-bit operation, we will show in Section 4.2 that the overall LUT resource usage of FuseBNN (even more optimized than BaseBNN) is almost the same as that of 4-Bit-Net. Such observations remain almost the same when we compare the more optimized FuseBNN+ (defined in Section 4.2, plotted in Fig. 6) over 4-Bit-Net.

4.2 Impact of Model Size on Accuracy and Hardware Performance with Common Hardware-Friendly Optimizations

To address the model size increase issue in SOTA BNNs, we first apply several widely used (minor) model size optimizations on top of our FuseBNN presented in Section 3, for the sake of comprehensiveness. Fig. 7 and 8 shown their impact on the accuracy and hardware resource.

(1) Quantization of the input and detection convolution layers: Prior studies find that the bit-width of the first input and the last detection convolution layers is more sensitive to the detection accuracy of BNNs [22]. Their direct binarization brings a dramatic accuracy drop. Instead, they can be quantified from floating-point to 8-bit integer without accuracy loss (so we still call it FuseBNN, 95% accuracy in Fig. 7).
(2) Binarization of the input layer (BI): To simplify the structure of the first input layer and binarize it, FracBNN proposes a thermometer encoding-based strategy [29]. We use the same strategy to expand the single-channel grayscale SAR image to 32 binary channels. This slightly reduces the BRAM and DSP usage by 0.3% and 0.2%, while slightly decreases the accuracy to 94.9%, shown as FuseBNN-BI.
(3) Replacement of average pooling with max pooling (MP): This is a common hardware-friendly strategy to avoid the addition and division involved in the window averaging operation. Unfortunately, this results in a 0.3% accuracy drop with minor resource consumption reduction, shown as FuseBNN-BI-MP in Fig. 7 and 8. This is because our proposed fusion strategy transfers the division involved in the average pooling to the coefficient calculated offline.

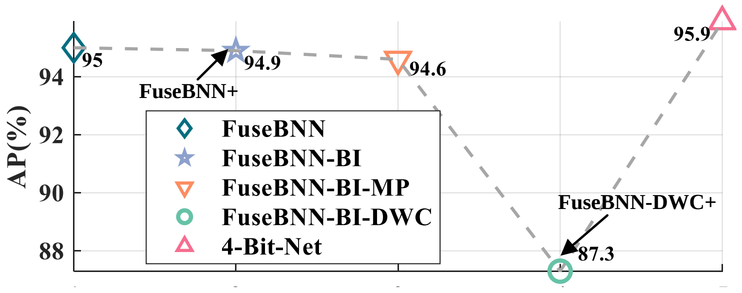

Fig. 7. Accuracy impact of common model size optimizations on FuseBNN.

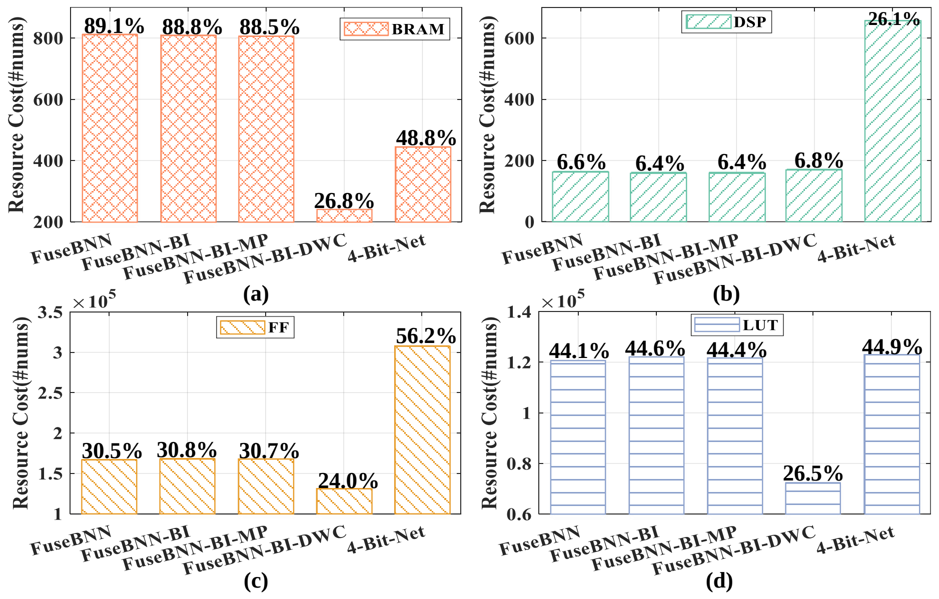

Fig. 8. Resource impact of several common model size optimizations described in Fig. 7, assuming all models have similar inference performance.

In the rest of this paper, we retain two effective optimizations for FuseBNN, i.e., the binarization of the input layer and the 8-bit quantization of the last detection layer, and refer this optimized model FuseBNN-BI as FuseBNN+ in Fig. 7.

FuseBNN+ vs. 4-Bit-Net. As shown in Fig. 6, FuseBNN+ (94.9% accuracy) has 21.6GOP and 0.9MB parameter size, which are 8.6× and 2.25× larger than those of 4-Bit-Net (95.9% accuracy), respectively. Note that FuseBNN+ has a slightly higher GOP than BaseBNN due to the slight network change presented in Section 3.3.2; however, it uses much less hardware resource than BaseBNN as presented in Section 3.4. As shown in Fig. 8(a) and Fig. 8(d), due to the increased model size, FuseBNN+ consumes 88.8% BRAMs and 44.6% LUTs of the entire FPGA, which uses 1.8× more BRAMs and almost the same amount of LUTs than 4-Bit-Net.

Comparison to FuseBNN-DWC+. Finally, we also compare with FuseBNN-DWC+, whose only difference to FuseBNN+ is that it directly binarizes the 3×3 DWC in the DSC blocks without increasing the model size. Shown in Fig. 6, the drawback is that FuseBNN-DWC+ drops the accuracy to 87.3%; but, it has a much smaller model size of 2.9GOP and 0.11MB parameter size. Shown in Fig. 8, FuseBNN-DWC+ uses significantly less resource than FuseBNN+ and 4-Bit-Net.

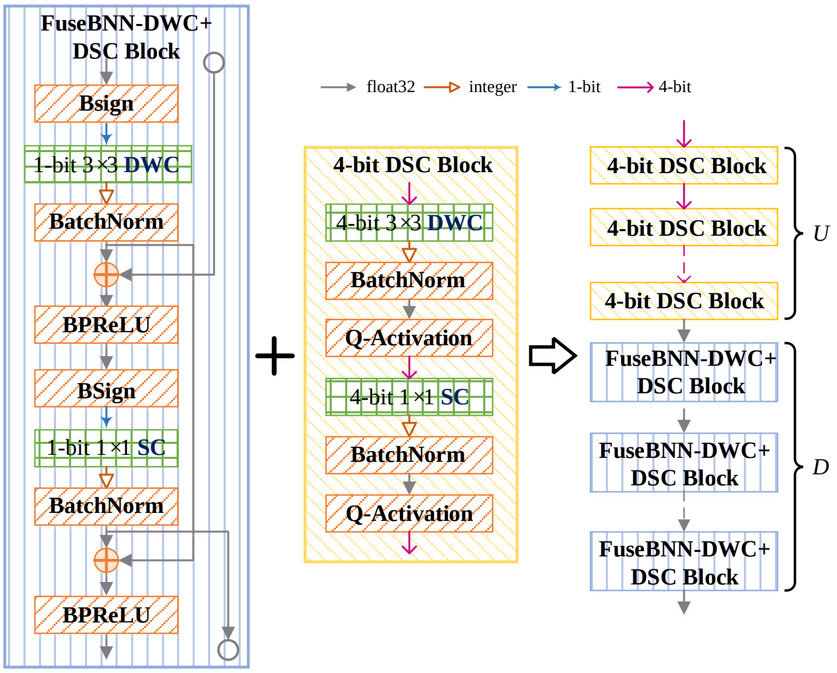

Fig. 9. Our proposed HyBNN, using a hybrid of U number of 4-Bit-Net DSC blocks and D number of FuseBNN-DWC+ DSC blocks.

4.3 HyBNN: Hybrid FuseBNN-DWC+ and 4-Bit-Net Depth-Wise Separable Convolution (DSC) Blocks

Considering the tradeoff of detection accuracy and resource overhead betwween FuseBNN+, FuseBNN-DWC+, and 4-Bit-Net, we propose HyBNN, where we use a hybrid of binary DSC blocks in FuseBNN-DWC+ to decrease model size and 4-bit DSC blocks in 4-Bit-Net to compensate for accuracy loss. Shown in Fig. 9, we replace the bottom D number of DSC blocks in MobileNetV1-SAR (in Table 1) with FuseBNN-DWC+ DSC blocks, and replace the up U number of DSC blocks with 4-Bit-Net DSC blocks. Q-Activation in the 4-bit DSC block indicates that the activation function, i.e., ReLU, is uniformly quantified by 4-bit. Potentially, a higher U leads to a higher accuracy, but a higher resource utilization and thus a lower inference throughput.

4.4 Accuracy and Hardware Performance of HyBNN

4.4.1 Hardware Implementation of HyBNN Blocks. Based on the all-on-chip dataflow-style hardware architecture described in Section 2.2, here we implement the primitive HyBNN blocks with configurable parameters, which can reused for different layers by simply configuring the block parameters to enable fast deployment of HyBNN on a target FPGA.

For the 4-bit DSC block, we use similar hardware blocks as those proposed in [28], including two major configurable blocks: the matrix-vector-activation unit (MVAU) for DWC layer (D-MVAU) shown in Fig. 10(a) and the MVAU for SC layer (S-MVAU) shown in Fig. 10(d). Both D-MVAU and S-MVAU have α number of PEs (processing elements). Each PE has β number of PUs (parallel units in D-MVAU) or OPUs (output parallel units in S-MVAU), one non-linear unit (NLU, i.e., merged BatchNorm and Q-Activation for 4-bit DSC block), one DownSizer and Upsizer to address the rate-mismatch problem between PUs/OPUs and the NLU.

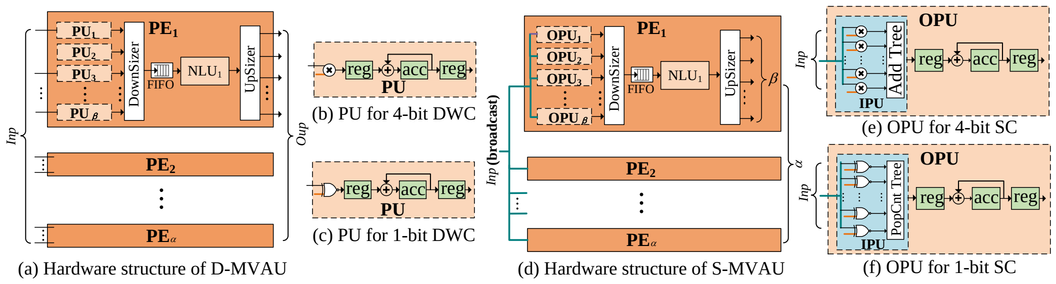

Fig. 10. Major HyBNN hardware blocks: D-MVAU (Matrix-Vector-Activation Unit for DWC) and S-MVAU (MVAU for SC). *Inp* and *Oup* denote the input channel parallelism for input feature map and the output channel parallelism for output feature map, which can be customized for each HyBNN layer.

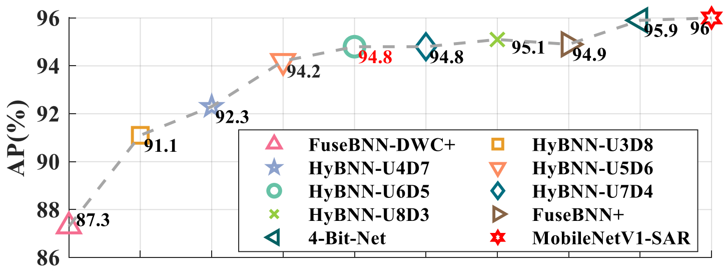

Fig. 11. Accuracy of different U and D settings for HyBNN on SAR imagery.

Both D-MVAU and S-MVAU have an input channel parallelism *Inp* for the input feature map and an output channel parallelism *Oup* for the output feature map, where $Oup = \alpha \times \beta$. For D-MVAU that performs DWC, shown in Fig. 10(a), $Inp = Oup = \alpha \times \beta$ and each PU works on one multiply, shown in Fig. 10(b). For S-MVAU that performs SC, shown in Fig. 10(d), $Inp \neq Oup$ and *Inp* input neurons are broadcast to each OPU. Each OPU processes *Inp* inputs in parallel with *Inp* multipliers and an adder tree, shown in Fig. 10(e). Both PU and OPU need an accumulator to accumulate partial results, as shown in Fig. 10(b) and 10(e), since both D-MVAU and S-MVAU are able to process a part of the inputs on hardware at a time. We refer interested audience to [28] for more details.

For the FuseBNN-DWC+ DSC block, the D-MVAU and S-MVAU are reused by redesigning PU, OPU and NLU. Shown in Fig. 10(c) and 10(f), the multipliers and the adder tree involved in the PU and OPU are replaced by simple XNOR gates and the POPCNT tree. Moreover, the NLU is updated to our fused threshold unit presented in Section 3.3.3.

A different instance of D-MVAU or S-MVAU can be easily customized for each layer by configuring its *Inp* and *Oup*, following the interlayer matching strategy in [28]. Moreover, the separation design of PU/OPU and NLU has a high flexibility for extension of different AFP components.

4.4.2 Results for Model Accuracy. Fig. 11 presents the detection accuracy of HyBNN under different U and D settings, where $U + D = 11$, since there are 11 DSC blocks in the backbone network as listed in the last column of Table 1. Even when $U = 3$, i.e., three 4-bit DSC blocks are integrated into HyBNN, the accuracy of HyBNN-U3D8 (91.1%) becomes 3.8% higher than that of FuseBNN-DWC+. When U gets larger, the accuracy gets higher. When $U = 6$, the accuracy of HyBNN-U6D5 increases to 94.8%, very close the that of FuseBNN+ (94.9%). Further increasing U to 7 and 8 gets marginal accuracy improvement (94.8% and 95.1% accuracy, respectively). Considering that increasing U also adds more hardware resource overhead, we choose HyBNN-U6D5 for the final hardware deployment on board.

Table 4. Overall performance, resource utilization, and accuracy comparison for ship detection on AMD-Xilinx ZCU102 FPGA with 300MHz.

Model	AP	Resource Cost				Peak FPS	FPS/BRAM	FPS/DSP	FPS/kLUTs
	(%)	BRAM	DSP	FF	LUT				
BaseBNN	94.9	1,032.5 (113.2%)	1,146 (45.5%)	472,534 (86.2%)	399,544 (145.8%)	failed	-	-	-
FuseBNN+	94.9	810 (88.8%)	162 (6.4%)	168,688 (30.8%)	122,129 (44.6%)	90	0.11	0.56	0.74
4-Bit-Net* (250MHz)	95.9	466 (51.1%)	1,997 (79.2%)	367,112 (67.0%)	192,157 (70.1%)	230	0.49	0.12	1.20
HyBNN-U6D5	94.8	555 (60.9%)	1,662 (66.0%)	276,583 (50.5%)	152,687 (55.7%)	615	1.11	0.37	4.03

4.4.3 Results for Model Size. Shown in Fig. 6, HyBNN-U6D5 has a parameter size of 0.19MB, which is 4.7× smaller than that of FuseBNN+ (0.9MB) and 2.1× smaller than that of 4-Bit-Net (0.4MB). Moreover, it has 2.5GOP, the same as 4-Bit-Net, since all DSC blocks are retained; this is 8.6× smaller than that of FuseBNN+ (21.6GOP).

4.4.4 Results for Final Hardware Performance. Table 4 compares the peak inference FPS (frames per second), resource utilization, and FPS per resource utilization, as well as accuracy, between BaseBNN, FuseBNN+, 4-Bit-Net, and HyBNN-U6D5. FPS measures the model inference throughput on FPGA with a batch size of 25. All designs are run on the AMD-Xilinx embedded ZCU102 FPGA platform at 300MHz, except 4-Bit-Net that is only able to run at 250MHz.

(1) BaseBNN without our optimizations has heavy LUT and BRAM consumption which goes beyond the total amount of resources. Therefore, it cannot be deployed on the FPGA.
(2) FuseBNN+, with our AFP component fusion, achieves 90 FPS. It dramatically reduces the DSP, FF, and LUT usage compared to BaseBNN, thanks to our fusion optimization. However, its performance is bottlenecked by the BRAM resource consumption (88.8%).
(3) 4-Bit-Net achieves 230 FPS, which is 2.6× faster than FuseBNN+, due to the increased model size of FuseBNN+.
(4) HyBNN-U6D5 achieves 615 FPS, which is 6.8× faster than FuseBNN+ and 2.7× faster than 4-Bit-Net. It has a detection accuracy of 94.8%, which is very close to that of BaseBNN and FuseBNN+ (94.9%) and only 1.1% lower than that of 4-Bit-Net (95.9%). It enjoys not only the benefits of small model size, simple XNOR-POPCNT operations, and AFP component fusion from FuseBNN-DWC+, but also the accuracy compensation from 4-Bit-Net. Compared with 4-Bit-Net, HyBNN-U6D5 also has 2.3×, 3.1×, and 3.4× higher FPS/BRAM, FPS/DSP, FPS/kLUTs efficiency.

5 GENERALIZATION STUDY ON CIFAR-10

To validate the generality of our approach, we further apply it to image classification on the CIFAR-10 dataset using a MobileNetV1-like backbone with 8 DSC blocks ($U + D = 8$) summarized in Table 5. All model training settings are the same as those described in Section 2.3, except that the number of epochs and batch size are adjusted to 300 and 128.

Results for model accuracy. Shown in Fig. 12, our FuseBNN achieves 90% top-1 accuracy, slightly (0.2%) higher than BaseBNN. Compared with FuseBNN-DWC+ (77.5%), introducing three 4-bit DSC blocks (HyBNN-U3D5: 87.7%) brings a significant accuracy gain of 10.2%. With five 4-bit DSC

Table 5. Configuration of backbone network MobileNetV1-CIFAR-10.

Input	Operator	c	s	n
32×32×3	SC 3×3	32	1	-
32×32×32	DWC 3×3	32	1	1
	SC 1×1	64	1	
32×32×64	DWC 3×3	64	2	1
	SC 1×1	128	1	
16×16×128	DWC 3×3	128	1	2
	SC 1×1	128	1	
16×16×128	DWC 3×3	128	2	1
	SC 1×1	256	1	
8×8×256	DWC 3×3	256	1	3
	SC 1×1	256	1	
8×8×256	Max_Pool	256	-	3
1×1×256	FC	10	-	-
1×1×10	Softmax	-	-	-

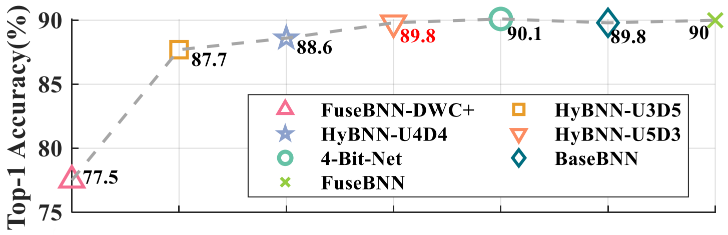

Fig. 12. Accuracy of different U and D settings for HyBNN on CIFAR-10.

Table 6. Operation number (GOP, *not considering bit-width*), parameter size (MB) and accuracy comparison for image classification on CIFAR-10.

Model	Param (MB)	GOP	Top-1 (%)
BaseBNN	0.32	0.25	89.8
FuseBNN			90
FuseBNN-DWC+	0.04	**0.03**	77.5
HyBNN-U5D3	**0.07**		**89.8**
4-Bit-Net	0.14		90.1
FracBNN[29]	0.03	0.07	89.1

blocks, the accuracy of HyBNN-U5D3 rises to the BaseBNN level (89.8%), which is only 0.3% lower than 4-Bit-Net.

Results for model size. Shown in Table 6, the parameter size of HyBNN-U5D3 is 0.07MB, 4.6× and 2× smaller than that of FuseBNN and 4-Bit-Net. Its number of MAC operations is 0.03GOP, the same as that of FuseBNN-DWC+ and 4-Bit-Net but 8.3× smaller than that of FuseBNN.

Comparison to FracBNN [29]. Compared to SOTA FracBNN that uses ResNet-20 as its backbone [29], our HyBNN-U5D3 achieves an accuracy gain of 0.7% and a 2.3× GOP reduction, shown in Table 6. Summarized in Table 7, running on the same AMD-Xilinx Ultra96-V2 FPGA, our HyBNN-U5D3 design (300MHz) achieves 4,302 FPS, which is 1.5× faster than FracBNN (250MHz). Moreover,

Table 7. Overall performance, resource utilization, and accuracy comparison for CIFAR10 classification on AMD-Xilinx Ultra96-V2 FPGA.

Model	Top-1 (%)	Freq (MHz)	Resource Utilization BRAM	DSP	FF	LUT	Peak FPS	FPS/BRAM	FPS/DSP	FPS/kLUTs
HyBNN-U5D3	**89.8**	**300**	94.5 (43.8%)	205 (56.9%)	99,074 (70.2%)	51,927 (73.6%)	**4302**	45.5	21	82.8
FracBNN[29]	89.1	250	212 (98.1%)	126 (35%)	39,618 (28.1%)	51,444 (72.9%)	2806	13.2	22.3	54.5

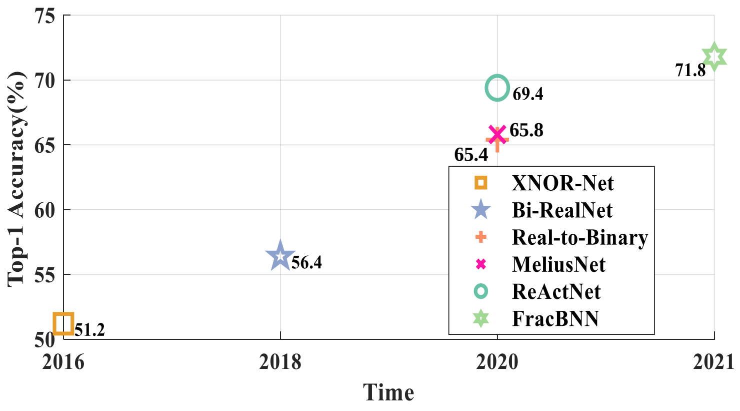

Fig. 13. Top-1 ImageNet classification accuracy of representative BNNs.

HyBNN-U5D6 achieves 1.5× and 3.4× higher FPS/kLUTs and FPS/BRAM efficiency. Such gains mainly come from the GOP reduction and the all on-chip dataflow accelerator design in HyBNN.

6 RELATED WORK

6.1 Recent Advances of Binary Neutral Networks

Fig. 13 illustrates the recent development of representative BNNs for the well-known ImageNet classification task over the past few years. The early-stage representative BNN XNOR-Net [22] demonstrated that the binarization of the weight and activation enjoys 32× memory savings and 58× convolution speedup on CPU. It also introduced the real-value scaling factor for the weight and activation to compensate for the information loss. However, it only achieved 51.2% top-1 accuracy on the ImageNet dataset, due to the naive straight-through structure illustrated earlier in Fig. 3(a).

In recent years, considerable efforts have been made to gradually narrow the accuracy gap [2, 15, 16, 18, 26, 27, 29, 30, 34]. Bi-RealNet [16] supplemented XNOR-Net with identity shortcuts to maintain the continuous propagation of floating-point information, which further increased top-1 accuracy to 56.4%. Due to the low theoretical computational complexity, this floating-point shortcut or more complex improved versions can often be found in later BNN designs. For instance, Real-to-Binary Net [18] used 32-bit channel-wise scale factors calculated by real-value activations of the previous layer before binarization to rescale the output of the current binary convolution. As a result, richer floating-point branch information improved the Real-to-Binary Net accuracy by 9% over Bi-RealNet. In MeliusNet [2], a binary variant of the DenseNet structure [11], the dense block concatenated the current 32-bit input feature map with the new features calculated by binary convolution to increase feature capacity. Furthermore, it enhanced the feature quality by adding additional 32-bit residual connections on the dense block. Such customized block design improved the accuracy of MeliusNet to 65.8%, 0.4% higher than Real-to-Binary Net.

More recently, ReActNet[15] reached ResNet-18-level accuracy, i.e., 69.4%, for the first time, which is considered as one of the SOTA BNNs. This was achieved by using two major optimizations. First, ReActNet increased the model size of the backbone network MobileNetV1 by replacing the DWC with the SC. Second, in addition to the widely used floating-point identify shortcut, ReActNet introduced BPReLU and Bsign functions to adjust the activation distribution in the model to learn more representative features.

FracBNN [29] is another SOTA BNN, which achieved a groundbreaking MobileNetV2-level accuracy of 71.8% for ImageNet classfication. Its core innovation is to replace ReActNet's binary SC with a more complex fractional SC with higher MAC operations. More precisely, it used the precision gating strategy[31] to calculate 2-bit activations and split a 2-bit activation into the high-bit activation (HBA) and the low-bit activation (LBA). In the base phase, it performs a regular binary SC using HBA. In a conditional updating phase (based on a learnable threshold), it performs an additional binary SC using LBA.

In this paper, we choose SOTA BNN ReActNet as our baseline and adapt it to BaseBNN for our case study; we explained why we did not choose FracBNN in Section 4.1. All of our analysis and optimizations also apply to FracBNN.

6.2 FPGA Acceleration for Binary Neural Networks

Due to the hardware-friendly features of BNNs, there are also an active body of research studies [3, 7, 8, 14, 19, 25, 32] to accelerate BNNs on the FPGA. Indeed, the early study in [19] showed the superior efficiency of BNN acceleration on FPGA than that on CPU and GPU. Among them, one of the most representative efforts is FINN [3, 25], which provided an automation framework for fast deployment of the early-stage BNN on FPGA and implemented an all-on-chip dataflow-style BNN inference accelerator (this inspired our accelerator design). Inherited from FINN's hardware design, ReBNet[7] employed multi-level residual binary convolutions to improve the accuracy, which is still below 42% though.

Besides FINN, Zhao et al. [32] introduced a BitSel module and variable-width line buffers to accelerate the inference of early-stage BNNs on FPGA. FP-BNN [14] only binarized the weights in the network. FBNA [8] binarized all network layers, including the first and last layers, but only targeted very small datasets such as SVHN and CIFAR-10.

Noted that all these aforementioned efforts accelerated early-stage BNNs that achieved low accuracy for ImageNet classification (if supported) and did not support complex AFP components discussed in this paper. The only exception is the recent FracBNN[29] that we compared to in Section 5: our HyBNN-U5D3 on CIFAR-10 dataset achieved higher clock frequency and throughput on the same FPGA, higher FPS/BRAM and FPS/kLUTs efficiency, and better top-1 accuracy.
Summary: Different to all prior studies, this is the first study to quantify and mitigate the hardware inefficiency in SOTA BNNs caused by various AFP components and increased model size for compensating the accuracy loss.

7 CONCLUSION AND FUTURE WORK

This work is the first to quantify and mitigate the hardware inefficiency in SOTA BNNs, which are mainly caused by various AFP components and increased model size that were proposed for accuracy gains. To eliminate the hardware overhead caused by AFP components without accuracy loss, we have proposed a novel algorithm/hardware co-design called *FuseBNN* to fuse cascaded AFP components. To alleviate the hardware inefficiency caused by increased model size, we have proposed *HyBNN*, where we use a hybrid of FuseBNN-DWC+ DSC blocks to keep compact model size and 4-Bit-Net DSC blocks to compensate the accuracy loss. Experimental results have demonstrated promising accuracy and superior hardware performance of HyBNN for SAR ship detection (94.8% detection accuracy and 615 FPS on AMD/Xilinx ZCU102 FPGA) and CIFAR-10 classification (89.8% top-1 accuracy and 4,302 FPS on AMD/Xilinx Ultra96-V2 FPGA) on two FPGA platforms. We plan to explore more SOTA BNNs, datasets, and FPGAs in future work.

REFERENCES

[1] AMD-Xilinx. 2021. Introduction to Vitis HLS. https://docs.xilinx.com/r/2020.2-English/ug1399-vitis-hls/Introduction-to-Vitis-HLS Last accessed July 28, 2022.

[2] Joseph Bethge, Christian Bartz, Haojin Yang, Ying Chen, and Christoph Meinel. 2020. Meliusnet: Can binary neural networks achieve mobilenet-level accuracy? *arXiv preprint arXiv:2001.05936* (2020).

[3] Michaela Blott, Thomas B Preußer, Nicholas J Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. 2018. FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 11, 3 (2018), 1–23.

[4] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. https://doi.org/10.48550/ARXIV.1602.02830

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255.

[6] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. 2020. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proc. IEEE* 108, 4 (2020), 485–532.

[7] Mohammad Ghasemzadeh, Mohammad Samragh, and Farinaz Koushanfar. 2018. ReBNet: Residual binarized neural network. In *2018 IEEE 26th annual international symposium on field-programmable custom computing machines (FCCM)*. IEEE, 57–64.

[8] Peng Guo, Hong Ma, Ruizhi Chen, Pin Li, Shaolin Xie, and Donglin Wang. 2018. FBNA: A Fully Binarized Neural Network Accelerator. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. 51–513.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[10] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[11] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4700–4708.

[12] Sergey Ioffe and Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*. PMLR, 448–456.

[13] Jianwei Li, Changwen Qu, and Jiaqi Shao. 2017. Ship detection in SAR images based on an improved faster R-CNN. In *2017 SAR in Big Data Era: Models, Methods and Applications (BIGSARDATA)*. IEEE, 1–6.

[14] Shuang Liang, Shouyi Yin, Leibo Liu, Wayne Luk, and Shaojun Wei. 2018. FP-BNN: Binarized neural network on FPGA. *Neurocomputing* 275 (2018), 1072–1086.

[15] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. 2020. Reactnet: Towards precise binary neural network with generalized activation functions. In *European conference on computer vision*. Springer, 143–159.

[16] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. 2018. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In *Proceedings of the European conference on computer vision (ECCV)*. 722–737.

[17] Ping Luo, Xinjiang Wang, Wenqi Shao, and Zhanglin Peng. 2018. Towards understanding regularization in batch normalization. *arXiv preprint arXiv:1809.00846* (2018).

[18] Brais Martinez, Jing Yang, Adrian Bulat, and Georgios Tzimiropoulos. 2020. Training binary neural networks with real-to-binary convolutions. *arXiv preprint arXiv:2003.11535* (2020).

[19] Eriko Nurvitadhi, David Sheffield, Jaewoong Sim, Asit Mishra, Ganesh Venkatesh, and Debbie Marr. 2016. Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC. In *2016 International Conference on Field-Programmable Technology (FPT)*. 77–84.

[20] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA, Article 721, 12 pages.

[21] Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe. 2020. Binary neural networks: A survey. *Pattern Recognition* 105 (2020), 107281.

[22] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*. Springer, 525–542.

[23] Joseph Redmon and Ali Farhadi. 2017. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7263–7271.

[24] Mengshu Sun, Zhengang Li, Alec Lu, Yanyu Li, Sung-En Chang, Xiaolong Ma, Xue Lin, and Zhenman Fang. 2022. FILM-QNN: Efficient FPGA Acceleration of Deep Neural Networks with Intra-Layer, Mixed-Precision Quantization. In *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 134–145.

[25] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*. 65–74.

[26] Peisong Wang, Xiangyu He, Gang Li, Tianli Zhao, and Jian Cheng. 2020. Sparsity-inducing binarized neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 12192–12199.

[27] Qi Wang, Nianhui Guo, Zhitong Xiong, Zeping Yin, and Xuelong Li. 2022. Gradient Matters: Designing Binarized Neural Networks via Enhanced Information-Flow. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 11 (2022), 7551–7562.

[28] Geng Yang, Jie Lei, Weiying Xie, Zhenman Fang, Yunsong Li, Jiaxuan Wang, and Xin Zhang. 2022. Algorithm/Hardware Codesign for Real-Time On-Satellite CNN-Based Ship Detection in SAR Imagery. *IEEE Transactions on Geoscience and Remote Sensing* 60 (2022), 1–18.

[29] Yichi Zhang, Junhao Pan, Xinheng Liu, Hongzheng Chen, Deming Chen, and Zhiru Zhang. 2021. FracBNN: Accurate and FPGA-efficient binary neural networks with fractional activations. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 171–182.

[30] Yichi Zhang, Zhiru Zhang, and Lukasz Lew. 2022. PokeBNN: A Binary Pursuit of Lightweight Accuracy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 12475–12485.

[31] Yichi Zhang, Ritchie Zhao, Weizhe Hua, Nayun Xu, G. Edward Suh, and Zhiru Zhang. 2020. Precision Gating: Improving Neural Network Efficiency with Dynamic Dual-Precision Activations. In *International Conference on Learning Representations*. https://openreview.net/forum?id=SJgVU0EKwS

[32] Ritchie Zhao, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng-Hau Lin, Mani Srivastava, Rajesh Gupta, and Zhiru Zhang. 2017. Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '17)*. Association for Computing Machinery, New York, NY, USA, 15–24.

[33] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).

[34] Shilin Zhu, Xin Dong, and Hao Su. 2019. Binary ensemble neural network: More bits per network or more networks per bit?. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4923–4932.