BoolNet: Towards Energy-Efficient Binary Neural Networks Design and Optimization

Anonymous submission

Abstract

Binary Neural Networks (BNNs) have been advancing towards bridging the accuracy gap with their 32-bit counterparts, often by integrating additional 32-bit components. However, such designs, including the use of 32-bit values for feature maps and residual shortcuts, pose challenges for hardware accelerators with constraints on memory, energy, and computing resources. Addressing the critical question of balancing accuracy with energy consumption in BNNs, we introduce *BoolNet*, a novel BNN architecture minimizing the use of 32-bit components. *BoolNet* employs 1-bit values for storing feature maps, achieving a significant balance between efficiency and performance. Our experiments on the ImageNet dataset show that *BoolNet* attains 63.0% Top-1 accuracy, along with a 2.95-fold reduction in energy consumption compared to recent state-of-the-art BNNs. The code and trained models will be made available at: `(URL in final version)`.

Introduction

The rise of *Deep Neural Networks* (DNNs) marks a significant milestone in AI advancements. Yet, their demand for high-performance computing resources, such as GPUs and TPUs, leads to substantial energy consumption and CO_2 emissions. For instance, training OpenAI's GPT-3 (Brown et al. 2020) emits as much CO_2 as the lifetime emissions of 43 cars. Their computational requirements also limit their deployment in resource-constrained environments like mobile phones and IoT devices. Efforts to mitigate these challenges include network pruning (Han, Mao, and Dally 2015), knowledge distillation (Crowley, Gray, and Storkey 2018), compact networks (Howard et al. 2017), and low-bit quantization (Courbariaux, Bengio, and David 2015). Among these, Binary Neural Networks (BNNs) (Courbariaux et al. 2016) represent an extreme case of low-bit quantization, using only 1 bit for weights and activations.

Despite their potential for memory compression and CPU speedup, as shown in (Rastegari et al. 2016), BNNs historically suffer from accuracy deficits compared to 32-bit models. To address this, recent research has focused on architectural innovations (Liu et al. 2018; Bethge et al. 2020), training methodologies (Martinez et al. 2020), and leveraging neural architecture search (Bulat, Martinez, and Tzimiropoulos 2020). However, current efficiency analyses in BNN literature primarily consider theoretical instruction counts, overlooking practical aspects like memory usage, hardware efficiency, and energy consumption. Moreover, the frequent use of 32-bit components in BNNs for accuracy improvements results in hardware inefficiencies and increased power usage, as noted by Fromm et al. (2020).

This paper thoroughly explores the balance between accuracy and hardware efficiency in BNNs. We propose a novel architecture, BoolNet, which significantly reduces the reliance on 32-bit components (see Figure 1b). BoolNet exclusively utilizes binary feature maps and fuses BN layers into the Sign function during inference, thereby reducing the computational load. Additional modifications include eliminating components that require 32-bit operations, such as PReLU, average pooling, and binary downsampling convolutions. Our novel *Multi-slice strategy* helps mitigate representational capacity loss from these changes. Experimental results on ImageNet showcase BoolNet's superior energy efficiency and reasonable accuracy compared to existing BNNs, as summarized in Figure 1c and detailed in Section .

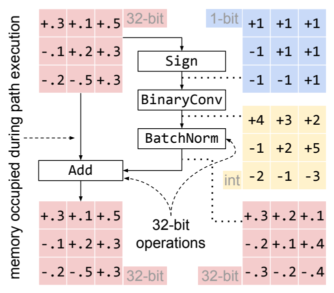
Our contributions are summarized as follows:

- Analyzing the impact of 32-bit layers in BNNs.
- Introducing BoolNet, a BNN architecture with minimal 32-bit components for enhanced efficiency.
- Proposing a Multi-slice strategy to offset accuracy loss from binary feature maps.
- Demonstrating a favorable balance between accuracy and energy consumption, with BoolNet outperforming existing BNNs in power efficiency and maintaining competitive accuracy.

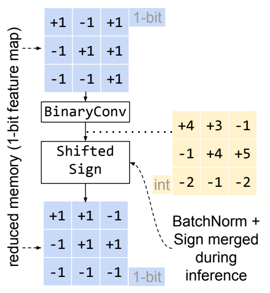
Related Work

The domain of *Efficient Deep Learning* has garnered considerable interest in recent years, with researchers exploring compact network designs (Howard et al. 2017; Zhang et al. 2018), knowledge distillation (Polino, Pascanu, and Alistarh 2018), network pruning (Han, Mao, and Dally 2015; Li et al. 2017), and low-bit quantization (Courbariaux, Bengio, and David 2015; Rastegari et al. 2016). The evolution of efficient models has progressed from manual designs to automated neural architecture search, aiming for optimal performance on specific hardware (Tan et al. 2019; Howard et al. 2019).

Binary Neural Networks (BNNs), introduced by Courbariaux et al. (2016), initially focused on smaller datasets



(a) Design in previous work.


reduced memory (1-bit feature map)
1-bit
BinaryConv
Shifted Sign
int
BatchNorm + Sign merged during inference
1-bit

(b) BoolNet design.

Method	Bitwidth (W/A/F)	Energy (mJ)	Top-1 Acc.	OPs $(\cdot 10^8)$
Bi-Real-Net	1/1/32	3.90	56.4%	1.63
BoolNet (ours)	1/1/4	1.33	63.0%	1.81
BaseNet (ours)	1/1/4	0.83	58.2%	1.54
BaseNet (ours)	1/1/1	0.70	53.3%	1.51

(c) By streamlining the feature map representation and memory access on hardware, BoolNet reduces the energy consumption by $2.9 - 5.57\times$ compared to Bi-RealNet by (Liu et al. 2018), while reaching similar or even higher (6.6%) Top-1 accuracy on ImageNet tasks.

Figure 1: The main differences between previous work and BoolNet. BoolNet uses 1-bit feature maps and a shifted sign function reducing memory requirements and the need for 32-bit operations.

like MNIST and CIFAR10. Subsequent developments, such as XNOR-Net (Rastegari et al. 2016), have strived to bridge the accuracy gap between BNNs and their 32-bit counterparts. Techniques like channel expansion in WRPN (Mishra et al. 2018), using multiple binary bases in ABC-Net (Lin, Zhao, and Pan 2017a), and real-valued shortcuts in Bi-RealNet (Liu et al. 2018) have significantly improved BNN accuracy. Recent approaches like MeliusNet (Bethge et al. 2020) and ReActNet (Liu et al. 2020b) have further advanced this field by achieving competitive accuracy with efficient architectures.

However, despite these advancements, the literature often overlooks the aspect of energy consumption in BNN design. Most studies focus on operation counts and memory usage, neglecting the practical energy efficiency crucial for real-world applications. This paper aims to address this oversight by exploring a balance between accuracy and energy efficiency in BNNs, emphasizing minimal reliance on 32-bit components to maintain the intrinsic efficiency of BNNs.

BoolNet

In this section, we first revisit the latest BNNs and recap how they enhanced the accuracy by adding more 32-bit components (in Section). Afterwards, we propose to replace most commonly used 32-bit components from current BNN designs and instead use a fully binary information flow in the network (in Section). However, abandoning 32-bit information flow results in a serious degradation of the representative capacity of the network. Thus, we also present our strategies to restore the representative capacity. The focus on boolean operations and binary feature maps leads to the name of our network: **BoolNet**.

Improving Accuracy with Additional 32-bit Components

Recent works on BNNs have made promising progress in narrowing the gap to their 32-bit counterparts. The key intention is to enhance the representative capacity by fully exploiting additional 32-bit components. However, such additional 32-bit components significantly reduce the hardware efficiency (as shown by Fromm et al. (2020) and further discussed in Section). The following list summarizes the 32-bit components commonly used in the latest BNNs: (1) The **channel-wise scaling factor** was first proposed by Rastegari et al. (2016) for approximating the 32-bit parameters. It increases the value range of activation and weight. (2) Bi-RealNet (Liu et al. 2018) proposes to use a **32-bit shortcut** for enclosing each binary convolution. The key advantage is that the network can maintain an almost completely 32-bit information flow. (3) XNOR-Net (Rastegari et al. 2016) uses **32-bit 1×1 downsampling** convolutions, which is also used by many subsequent methods (Liu et al. 2018; Martinez et al. 2020; Bethge et al. 2020). Bethge et al. (2019) shows that this simple strategy can achieve about 3.6% Top-1 accuracy gains on ImageNet based on a binary ResNet-18 model. (4) Martinez et al. (2020); Bulat, Martinez, and Tzimiropoulos (2020, 2021) show that **PReLU activation** effectively improves accuracy of BNNs. ReActNet (Liu et al. 2020b) constructs the RPReLU activation function and uses it before every sign function. (5) Martinez et al. (2020) reuse the 32-bit activation in their Real-to-Binary Net after BN with a squeeze and excitation (SE) **attention mechanism**. This module can adaptively re-scale the outputs of each binary convolution but needs additional 32-bit operations.

Although these techniques can effectively improve the accuracy, they increase the number of 32-bit values and floating point operations, making them not particularly efficient on hardware accelerators. They are closer to mixed-precision neural networks rather than being highly efficient binary neural networks, as one might expect.

BaseNet: Replacing 32-bit Components with Boolean Operations

To better balance accuracy and efficiency, we rethink the additional 32-bit components (Batch Normalization, 32-bit feature maps, scaling factors and PReLU) elaborated in the previous section and propose to remove or replace them with more efficient operations. We further propose a new basic convolution block without 32-bit operations, where we rearranged the order of the convolution basic block as [BinaryConv, BatchNorm, Sign], so that all feature maps are binary. These general changes constitute our BoolNet baseline, in short *BaseNet*.

Integrating the BatchNorm into the Sign Function Umuroglu et al. (2017) suggested replacing the BatchNorm

(BN) with a thresholding operation during inference on FPGAs. However, their suggestion can not be applied to more recent work (Hubara et al. 2016; Rastegari et al. 2016; Liu et al. 2018, 2020b; Bethge et al. 2020), because the layer order in these works is [Sign, BinaryConv, BN] surrounded by 32-bit valued shortcuts. Instead, these recent works have kept the 32-bit BatchNorm layer in both the training and testing stages. However, using a 32-bit BN right after the 1-bit convolution layer decreases the computational efficiency of hardware, using more memory and energy. Thus, in the following, we propose to fuse the BN layer into the Sign function during the inference stage and do not use the 32-bit output of BN layer for shortcut connection.

During the training phase, the batch normalization layer normalizes feature maps with a running mean μ and a running variance σ. For inference, it utilizes the constant statistic mean and variance instead, which in result can be reformulated as a linear process, expressed as:

$$\begin{aligned} y_i &= \gamma \frac{x_i - \mu}{\sqrt{\|\sigma^2 + \epsilon\|}} + \beta \\ &= \frac{\gamma}{\sqrt{\|\sigma^2 + \epsilon\|}} x_i + \left(\beta - \frac{\gamma\mu}{\sqrt{\|\sigma^2 + \epsilon\|}}\right) \end{aligned} \tag{1}$$

where x_i and y_i represent the N-dimensional input and output of a BN layer. γ and β are trainable scale and shift parameters, which are constant during the inference. $||\dots||$ is the absolute function. We can therefore simplify the formula as follows:

$$y_i = ax_i + b \;=\; a\left(x_i + \frac{b}{a}\right) = a\left(x_i + c\right) \;, \tag{2}$$

where a, b, and c denote constants in the formula. By transforming a into its sign and its absolute value, we have

$$y_i = ||a|| \circledast \text{Sign}\left(a\right) \odot \left(x_i + c\right), \tag{3}$$

As arranged in our basic block, Equation (3) is followed by a sign function, and Sign(y_i) only depends on Sign(a) and $(x_i + c)$. We thus derive a parameterized sign function as:

$$\text{Sign}(y_i) = \text{XNOR}(\text{Sign}(a),\, \text{Sign}(x_i + c)) \tag{4}$$

We further replace $\odot$ by using XNOR operator so that only bit-wise operations are adopted in the inference.

Further Reducing 32-bit Operations We rarely use the PReLU activation function, which is commonly used in the recent literature (Liu et al. 2018; Martinez et al. 2020; Bulat, Martinez, and Tzimiropoulos 2021) and brings a lot of extra overhead to the hardware implementation (it is only used once before the final dense layer). We also decided not to use scaling factors as suggested by Liu et al. (2018); Bethge et al. (2019). There are two components that use 32-bit operations and parameters in previous work, which are kept in 32-bit in BoolNet: the first convolution and the last dense layer. Directly replacing them with binary versions leads to a severe accuracy loss (Rastegari et al. 2016), thus we leave the investigation of alternatives for these special layers for future work.

BoolNet: Enhancing Binary Information Flow

The network design changes explained in the previous section, constitute our BoolNet baseline, called *BaseNet*. Although it uses a completely binary information flow which minimizes the energy and memory consumption, the representative capacity of BaseNet is drastically degraded compared to its 32-bit counterparts and previous BNN methods, which accumulate information in 32-bit values. To counter this reduction of representative capacity, we propose the following three ideas, which constitute our proposed **BoolNet**.

Multi-slice Binary Convolution (MS-BConv). In contrast to standard BNNs that use a single 1-bit value per 32-bit value, our multi-slice approach employs multiple 1-bit values (k slices) to diminish information loss typically associated with the sign function. This strategy is analogous to multi-slice imaging in MRI, enhancing the richness of the feature maps.

We redefine the binarization process as a multi-slice projection, allowing for the retention of more relative magnitude information. The modified sign function is formulated as:

$$x_i^b = \text{Sign}(x_i, b_n), \tag{5}$$

where b_n, a set of constant biases, is defined as:

$$b_n = \frac{\pm 2n}{k}, \quad \text{where } n = 0, 1, \dots, k/2. \tag{6}$$

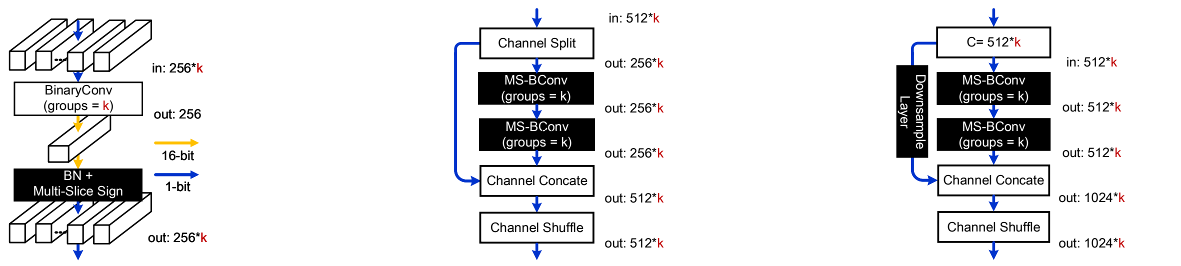
This approach effectively expands the channel dimension without altering the convolution's parameter count or operation amount, as the number of groups in the convolution matches k. The binary projection output x_i^b, with dimension $[N, C * k, H, W]$, is then fed into the subsequent binary convolution layer.

Contrasting our work with related studies (Lin, Zhao, and Pan 2017b; Zhuang et al. 2019; Zhu, Dong, and Su 2019), we note that while these studies increase both the feature map and weight bit-width to enhance accuracy, thus multiplying computation and memory costs, MS-BConv strategically reduces memory requirements by replacing 32-bit values with k 1-bit values, maintaining a constant operation count through grouped convolutions.

Enhancing MS-BConv further, we introduce a **Local Adaptive Shifting** module, inspired by FReLU (Ma, Zhang, and Sun 2020). This module, comprising a binary depth-wise 3×3 convolution and batch normalization layer, adaptively adjusts pixel zero points, thereby refining the feature extraction process. The detailed block design of MS-BConv is illustrated in Figure 2a.

Strengthening Information Propagation in BoolNet. Deep neural networks derive their robust representational capacity from layer-by-layer feature extraction and accumulation. Traditional methods use 32-bit addition operations for this accumulation, leading to substantial 32-bit data flow, which contradicts the goal of developing hardware-efficient binary neural networks. To address this, we enhance BoolNet's *BaseNet* structure by reusing binary features, thus avoiding 32-bit data flow.

Inspired by ShuffleNet-V2's (Ma et al. 2018) approach to information fusion and retention, where the input tensor is split and one part is used for feature extraction while the other is directly concatenated with the extracted features, we



(a) MS-BConv module (b) BoolNet basic block (c) BoolNet downsample block

Figure 2: The detailed architecture of BoolNet. To enhance the information flow, we modify the baseline architecture in two aspects: a) Reducing information loss with our multi-slices binary convolution. b) Strengthening the information propagation by using split and concatenate operations.

adopt a similar strategy in BoolNet. This design, illustrated in Figure 2b, involves splitting the feature extraction branch into two MS-BConv modules, with one branch remaining as the identity. The branches are then concatenated and undergo channel shuffling to ensure uniform feature distribution across layers.

The BoolNet downsampling block, depicted in Figure 2c, is uniquely designed to double the number of channels without requiring channel splitting. This novel approach to information accumulation differentiates BoolNet from *BaseNet* and is central to our proposed architecture (as detailed in Section).

Rethinking the Downsample Branch. Furthermore, we modify the downsampling block compared to previous methods, which usually use the layers [2×2 AvgPool (Stride 2), 32-bit 1×1 Conv, BN] in this branch (Liu et al. 2018; Martinez et al. 2020). Instead, we propose to use the layers [1-bit 1×1 Conv, 2×2 MaxPool (Stride 2), BN, Sign] and replace the 1-bit 3×3 Conv (stride 2) in the main branch with [1-bit 3×3 Conv, 2×2 MaxPool (stride 2)]. Overall, this removes all 32-bit operations and 32-bit parameters from the downsample block of BoolNet, but due to space limitations, we discuss the details in the appendix (including alternatives and experimental results).

Training with Progressive Weight Binarization

Though we intend to build highly efficient BNNs with a fully binary information flow, this strategy makes the network more sensitive to weight initialization during training. Traditional methods tried to alleviate similar problems through two-stage training (Martinez et al. 2020; Liu et al. 2020b), which makes the training more complicated. In this paper, we adopt a progressive binarization technique based on the traditional Hardtanh-STE method (Courbariaux et al. 2016). This can be understood as a smooth version of a multi-stage training approach. Specifically, in the training phase, a differentiable function $F(x)$ is used to replace the sign function. The slope of this function is adjusted by a single scalar λ. As the slope increases, the weight gradually changes from 32-bit to 1-bit. During backward propagation, we approximate $F(x/\lambda)$ with $F(x/1)$, to avoid the problem of gradients clipping as λ decreases. In the testing phase, we use the regular sign function for inference. The whole process can be formulated as:

$$F(x, \lambda) = \lim_{\lambda \to 0} \text{Hardtanh}\left(\frac{x}{\lambda}\right) \simeq \text{Sign}(x). \tag{7}$$

To smooth the weight binarization process, we schedule λ during training with an exponential decay strategy $\lambda_t = \sigma^{(t)}$, where $\sigma < 1$ is the exponential decay rate of λ.

Experiments

We evaluated our baseline network *BaseNet* and our proposed *BoolNet* (as described in Sections and , respectively) on the ImageNet dataset (Deng et al. 2009). In the following section, we first present the training details for our experiments. Afterwards, we conduct an ablation study on our proposed network design changes and the *Multi-slice* convolution (in Section and), analyze the energy consumption of BoolNet and other recent work on BNNs (in Section), and compare our model accuracy to state-of-the-art BNN models (in Section).

Training Details

Our training methodology largely follows the guidelines set by Bethge et al. (2020), with a few modifications. We extend the training duration to 256 epochs and adopt the knowledge distillation approach from Liu et al. (2020b), using a 32-bit ResNet-34 (He et al. 2016) as the teacher model. Complete hyperparameter specifications, additional details, and code can be found in the appendix and supplementary material. In place of the conventional two-stage training employed in Liu et al. (2020b); Martinez et al. (2020), we introduce progressive weight binarization (refer to Section , Equation 7). This approach utilizes a decay factor $\sigma = 0.965$, with λ set as $\lambda = 0.965^t$, where t denotes the number of processed samples divided by 256,000 and rounded down. During validation, the progressive weight binarization is substituted with the sign function for consistency. Comparative testing between the two-stage training strategy and our method was conducted using a ResNet-like model with binary feature maps. The two-stage training, encompassing 60 epochs for each stage (totaling 120 epochs), yielded an accuracy of 49.60%. In contrast, our progressive weight binarization method achieved 48.39% accuracy over 60 epochs, and

Table 1: Our ablation study on ImageNet ((Deng et al. 2009)) regarding accuracy, number of 32-bit operations (FLOPs), 1-bit operations (BOPs), and model size. (OPs = FLOPs + $^1/_{64}\cdot$BOPs)

Network Configuration	BaseNet					BoolNet				
	Top 1 Acc.	FLOPs $(\cdot 10^8)$	BOPs $(\cdot 10^9)$	OPs $(\cdot 10^8)$	Model Size	Top 1 Acc.	FLOPs $(\cdot 10^8)$	BOPs $(\cdot 10^9)$	OPs $(\cdot 10^8)$	Model Size
Baseline (60 epochs, CE Loss)	47.69%	1.22	1.68	1.49	3.47 MB	54.07%	2.78	1.85	3.07	5.05 MB
+ Multi-Slice strategy (k=4)	52.27%	1.22	1.69	1.49	3.47 MB	56.84%	2.78	1.86	3.07	5.05 MB
+ (1) Modified downsample branch	(BaseNet has no downsample branch)					58.66%	1.23	2.48	1.62	3.84 MB
+ (2) Local Adaptive Shifting[†]	52.08%	1.25	1.69	1.51	3.47 MB	59.56%	1.26	2.48	1.65	3.84 MB
+ (3) MaxPool instead of stride	55.14%	1.23	2.21	1.57	3.47 MB	59.98%	1.26	3.53	1.81	3.84 MB
+ (4) Knowledge distillation[‡]	56.84%	1.23	2.21	1.57	3.47 MB	61.98%	1.26	3.53	1.81	3.84 MB
+ Long training (256 epochs)	58.20%	1.23	2.21	1.57	3.47 MB	63.00%	1.26	3.53	1.81	3.84 MB

[†] Local Adaptive Shifting is not used for the subsequent BaseNet experiments [‡] Replaces the cross-entropy loss with the distributional loss by (Liu et al. 2020b)

k	BaseNet baseline + (3) + (4)							BoolNet baseline + (1) + (2) + (3) + (4)						
	Top 1 Acc.	Δ	Top 5 Acc.	FLOPs $(\cdot 10^8)$	BOPs $(\cdot 10^9)$	OPs $(\cdot 10^8)$	Model Size	Top 1 Acc.	Δ	Top 5 Acc.	FLOPs $(\cdot 10^8)$	BOPs $(\cdot 10^9)$	OPs $(\cdot 10^8)$	Model Size
1	51.74%	-	75.39%	1.23	2.20	1.57	3.47 MB	57.62%	-	80.47%	1.26	3.05	1.74	3.71 MB
2	55.75%	+4.01	79.08%	1.23	2.20	1.57	3.47 MB	60.57%	+3.95	82.56%	1.26	3.21	1.76	3.76 MB
4	56.84%	+1.09	79.85%	1.23	2.21	1.57	3.47 MB	61.98%	+1.41	83.75%	1.26	3.53	1.81	3.84 MB
8	57.19%	+0.35	80.33%	1.23	2.22	1.58	3.47 MB	62.54%	+0.56	84.14%	1.26	4.16	1.91	4.01 MB

50.19% over 120 epochs. This demonstrates that our training strategy effectively eliminates the need for a two-stage approach while delivering comparable or slightly superior performance within a similar total training duration.

Ablation on Network Design and Training

In our ablation study, detailed in Table 1, we evaluated the impact of various design changes and training techniques on BaseNet and BoolNet. The Multi-Slice strategy with $k = 4$ notably improved accuracy by 4.58% for BaseNet and 2.77% for BoolNet, compared to the baseline models trained for 60 epochs using cross-entropy loss. Transitioning from *BaseNet to BoolNet* resulted in an accuracy boost of 4.57% and 6.38% (with and without the Multi-Slice strategy, respectively). However, BoolNet's design inherently involves a higher count of 32-bit operations. By *modifying the downsample branch* (discussed in Section), these additional 32-bit operations were eliminated, leading to a further 1.82% increase in BoolNet's accuracy. The *Local Adaptive Shifting* module, enhancing BoolNet's accuracy by 0.90%, did not show similar benefits for BaseNet, suggesting its limited impact in models lacking traditional shortcut branches in the downsampling block. In contrast, employing a *MaxPool* layer instead of strided convolutions significantly aided BaseNet in information retention. Regarding training techniques, switching from cross-entropy loss to knowledge distillation, as proposed by Liu et al. (2020b), added 1.70% to BaseNet's and 2.00% to BoolNet's accuracy. Extending the training duration from 60 to 256 epochs further augmented the accuracy by 1.36% for BaseNet and 1.02% for BoolNet. In summary, our results affirm that the proposed network modifications, coupled with knowledge distillation, effectively enhance the accuracy of both BaseNet and BoolNet.

Ablation on the Multi-slice Binary Convolution

The previous section has already shown that our Multi-slice Binary Convolution (see Section) can reduce the accuracy loss caused by using 1-bit feature maps. However, we further evaluated the influence of the number of slices k in these convolutions based on the best configuration of the previous section but only training for 60 epochs (see the lower half of Table 1). Our results show that the final accuracy increases with each increase of k, but there are diminishing returns. Doubling k from 1 to 2, from 2 to 4, and from 4 to 8 increases accuracy by 3.95%, 1.41%, and 0.56% respectively (for BoolNet). Based on the increase of the number of operations, model size, and projected memory consumption, we use $k = 4$ for the best trade-off between accuracy and energy efficiency for our following experiments.

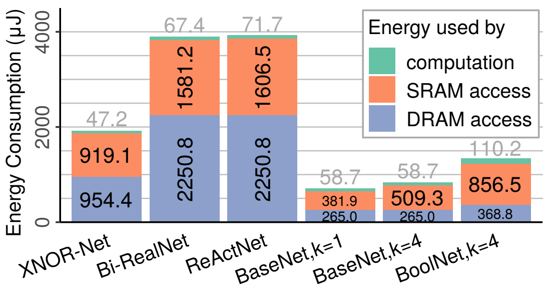
Energy Consumption Evaluation

This section aims to efficiently compare the energy consumption of BoolNet to classic BNN architectures under strictly fair design conditions. We thus implemented five BNN accelerators (BaseNet, BoolNet, XNOR-Net, Bi-RealNet, ReActNet (based on a Bi-RealNet backbone)). Considering the scope of this work, we leave the details of further hardware optimization of individual accelerators for future work.

We designed the five accelerators in the RTL language. Then, the power and area of computing circuits is given by Design Compiler (DC) with a TSMC 65nm process and 1GHz clock frequency. We refer to the design and implementation methods of computing units of Conti, Schiavone, and Benini (2018); Zhang et al. (2021). For a fair comparison between the different BNNs, we keep the design of architecture, data stream, the parallelism of computing units, and total on-chip cache (192KB for feature maps and 288KB for weights) consistent and only change the bit-width of the data stream and computing units. More specifically, the parallelism of binary convolution is 64×64, and the parallelism of other units is 64 in all accelerators (except the IntConv module is 8×64). These modules are pipelined and run at 1GHz. When DRAM bandwidth can be fully utilized, the performance depends on the parallelism and is bounded by the convolution,

Methods	Bitwidth (W/A/F)	Energy Consumption	Top-1 Acc.	OPs ($\cdot 10^8$)
ReActNet (Bi-Real)†	1/1/32	3.93mJ	65.9%	1.63
Bi-RealNet	1/1/32	3.90mJ	56.4%	1.63
XNOR-Net	1/1/32	1.92mJ	51.2%	1.59
BoolNet, k=4 (ours)	1/1/4	1.33mJ	63.0%	1.64
BaseNet, k=4 (ours)	1/1/4	0.83mJ	58.2%	1.54
BaseNet, k=1 (ours)	1/1/1	**0.70mJ**	53.3%	1.51

(a) The advantage of BoolNet is reduced energy consumption. †The ReActNet result based on a Bi-RealNet backbone is stated on the official Github repository (Liu et al. 2020a).



(b) Energy consumption regarding computations and access to DRAM/SRAM.

Figure 3: Comparison between BoolNet and state-of-the-art BNNs. The energy consumption is calculated through hardware simulations.

so each accelerator has the same peak performance for convolution, i.e., 4096 GOPs/s. Therefore, we achieve similar throughputs between 2044 and 2237 samples per second and it is reasonable to compare the energy consumption of the whole inference process.

DC can provide the hierarchical circuit area and power of computing units, including static power (P_s) and dynamic power (P_d). For each layer of the network, we know the amount (A) of each operation. According to the circuit parallelism (P_a), we can calculate the required number of cycles (C_n = A / P_a) and then the energy consumption according to the frequency and power ($E_c = C_n \times (P_s + P_d) / 10^{-9}$). For the operations with fewer cycles, the energy consumption waiting for other units is estimated by static power ($E_s = (C_n^{max} - C_n) \times P_s / 10^{-9}$). Similar to Wang, Zhang, and Han (2021); Jiang and Zokaee (2021); Li et al. (2021), we evaluate the energy consumption of on-chip SRAM access and off-chip DRAM access by using CACTI 6.5 (CACTI) and the power calculator of DDR provided by Micron (Micron). According to the cycle accurate simulation above, we can get the access amount of SRAM and DRAM, then get the energy consumption. The above components sum the overall energy consumed by a single inference pass. More details of simulation and energy estimation can be found in .

Memory access and computation significantly influence the energy efficiency of hardware accelerators. While traditional BNN efficiency analyses primarily consider theoretical instruction counts, the role of memory access is often overlooked. Studies by Yang et al. (2017) and Han et al. (2015) reveal that computation energy forms only a fraction (approximately 10%) of the total energy consumption in 32-bit networks, with memory access, especially SRAM and DRAM, being substantially more energy-intensive. In contrast to existing BNNs, BoolNet extensively utilizes 1-bit data streams, reducing memory requirements, particularly in the network's early stages (as detailed in Figure 6b in the appendix). This reduction in memory access contributes significantly to BoolNet's enhanced energy efficiency. Unlike other BNNs that rely on 32-bit feature map storage, BoolNet's Shifted Sign design allows immediate quantization of convolution accumulation results to 1 bit, bypassing the need for extensive cache or DRAM access. This stark difference in data bit-width and read/write volume leads to notable energy savings, as depicted in Figure 7. Our results (Figure 3b) indicate a greater disparity between computing and memory access energy consumption than previously reported. The bitwise operations `XNOR` and `popcount` in BoolNet further reduce computational energy share, underscoring the importance of memory optimization in BNNs for creating highly energy-efficient AI accelerators.

Comparison to State-of-the-Art BNNs

In our comparison with leading BNNs such as ReActNet, Bi-RealNet, and XNOR-Net (Liu et al. 2018, 2020b; Rastegari et al. 2016), BoolNet demonstrated significant energy savings. We based our comparison on the ReActNet performance, as reported on its official Github repository (Liu et al. 2020a), using a similar backbone. Our experiments revealed that by removing 32-bit components, BaseNet with k=1 achieved an energy reduction up to 5.6×, albeit with a 12.6% drop in accuracy. Implementing our Multi-slice strategy (k=4) in BoolNet mitigated this accuracy decrease by 4.9%, while still realizing a 4.7× energy reduction. Compared to Bi-RealNet (Liu et al. 2018), BoolNet with k=4 not only decreased energy consumption by 2.95× but also improved accuracy by 6.6%. These results showcase BoolNet's and BaseNet's capability to significantly reduce energy consumption with a relatively small trade-off in accuracy, marking a notable advancement in the efficiency of BNN architectures.

Conclusion

This paper presents BoolNet, a novel approach in Binary Neural Networks (BNNs) focused on optimizing energy efficiency while maintaining accuracy. By eliminating the reliance on 32-bit components, BoolNet demonstrates significant improvements in computational and energy efficiency, as validated through ImageNet experiments and hardware simulations. This work challenges traditional metrics of efficiency based on theoretical operation counts, offering a practical perspective towards developing ultra-energy-efficient BNNs. Our contributions mark a significant step in advancing BNN technology, aligning it more closely with its core principles of efficiency.

References

Bethge, J.; Bartz, C.; Yang, H.; and Meinel, C. 2020. MeliusNet: Can Binary Neural Networks Achieve MobileNet-level Accuracy? *arXiv preprint arXiv:2001.05936*.

Bethge, J.; Yang, H.; Bornstein, M.; and Meinel, C. 2019. BinaryDenseNet: developing an architecture for binary neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 0–0.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language Models are Few-Shot Learners. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 1877–1901. Curran Associates, Inc.

Bulat, A.; Martinez, B.; and Tzimiropoulos, G. 2020. Bats: Binary architecture search. In *European Conference on Computer Vision*.

Bulat, A.; Martinez, B.; and Tzimiropoulos, G. 2021. High-Capacity Expert Binary Networks. In *International Conference on Learning Representations*.

CACTI. ???? CACTI. Accessed: 2021-09-29.

Conti, F.; Schiavone, P. D.; and Benini, L. 2018. XNOR neural engine: A hardware accelerator IP for 21.6-fJ/op binary neural network inference. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11): 2940–2951.

Courbariaux, M.; Bengio, Y.; and David, J.-P. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, 3123–3131.

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

Crowley, E. J.; Gray, G.; and Storkey, A. J. 2018. Moonshine: Distilling with cheap convolutions. In *Advances in Neural Information Processing Systems*, 2888–2898.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 248–255. Ieee.

Fromm, J.; Cowan, M.; Philipose, M.; Ceze, L.; and Patel, S. 2020. Riptide: Fast end-to-end binarized neural networks. *Proceedings of Machine Learning and Systems*, 2: 379–389.

Guo, J.; He, H.; He, T.; Lausen, L.; Li, M.; Lin, H.; Shi, X.; Wang, C.; Xie, J.; Zha, S.; Zhang, A.; Zhang, H.; Zhang, Z.; Zhang, Z.; and Zheng, S. 2019. GluonCV and GluonNLP: Deep Learning in Computer Vision and Natural Language Processing. *arXiv preprint arXiv:1907.04433*.

Han, S.; Mao, H.; and Dally, W. J. 2015. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. Cite arxiv:1510.00149Comment: Published as a conference paper at ICLR 2016 (oral).

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

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Howard, A.; Sandler, M.; Chu, G.; Chen, L.-C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; et al. 2019. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*, 1314–1324.

Howard, A. G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; and Adam, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; and Bengio, Y. 2016. Binarized neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 4114–4122.

Jiang, L.; and Zokaee, F. 2021. EXMA: A Genomics Accelerator for Exact-Matching. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 399–411. IEEE.

Li, H.; Kadav, A.; Durdanovic, I.; Samet, H.; and Graf, H. P. 2017. Pruning Filters for Efficient ConvNets. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Li, J.; Louri, A.; Karanth, A.; and Bunescu, R. 2021. CSCNN: Algorithm-hardware Co-design for CNN Accelerators using Centrosymmetric Filters. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 612–625. IEEE.

Lin, X.; Zhao, C.; and Pan, W. 2017a. Towards Accurate Binary Convolutional Neural Network. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Lin, X.; Zhao, C.; and Pan, W. 2017b. Towards accurate binary convolutional neural network. *arXiv preprint arXiv:1711.11294*.

Liu, L.; Jiang, H.; He, P.; Chen, W.; Liu, X.; Gao, J.; and Han, J. 2019. On the Variance of the Adaptive Learning Rate and Beyond. *arXiv preprint arXiv:1908.03265*.

Liu, Z.; Shen, Z.; Savvides, M.; and Cheng, K. 2020a. Official source code of ReActNet on Github. Accessed: 2021-10-01.

Liu, Z.; Shen, Z.; Savvides, M.; and Cheng, K. 2020b. ReActNet: Towards Precise Binary Neural Network with Generalized Activation Functions. In Vedaldi, A.; Bischof, H.; Brox, T.; and Frahm, J., eds., *Computer Vision - ECCV 2020*

- *16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XIV*, volume 12359 of *Lecture Notes in Computer Science*, 143–159. Springer.

Liu, Z.; Wu, B.; Luo, W.; Yang, X.; Liu, W.; and Cheng, K.-T. 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.

Ma, N.; Zhang, X.; and Sun, J. 2020. Funnel activation for visual recognition. *arXiv preprint arXiv:2007.11824*.

Ma, N.; Zhang, X.; Zheng, H. T.; and Sun, J. 2018. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In *European Conference on Computer Vision*.

Martinez, B.; Yang, J.; Bulat, A.; and Tzimiropoulos, G. 2020. Training binary neural networks with real-to-binary convolutions. In *International Conference on Learning Representations*.

Micron. ???? Micron. Accessed: 2021-09-29.

Mishra, A.; Nurvitadhi, E.; Cook, J. J.; and Marr, D. 2018. WRPN: Wide reduced-precision networks. In *International Conference on Learning Representations (ICLR)*.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, 8026–8037.

Polino, A.; Pascanu, R.; and Alistarh, D. 2018. Model compression via distillation and quantization. In *ICLR (Poster)*. OpenReview.net.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, 525–542. Springer.

Tan, M.; Chen, B.; Pang, R.; Vasudevan, V.; Sandler, M.; Howard, A.; and Le, Q. V. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2820–2828.

Umuroglu, Y.; Fraser, N. J.; Gambardella, G.; Blott, M.; Leong, P.; Jahre, M.; and Vissers, K. 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.

Wang, H.; Zhang, Z.; and Han, S. 2021. SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 97–110. IEEE.

Yang, T.-J.; Chen, Y.-H.; Emer, J.; and Sze, V. 2017. A method to estimate the energy consumption of deep neural networks. In *2017 51st asilomar conference on signals, systems, and computers*, 1916–1920. IEEE.

Zhang, X.; Zhou, X.; Lin, M.; and Sun, J. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 6848–6856.

Zhang, Y.; Pan, J.; Liu, X.; Chen, H.; Chen, D.; and Zhang, Z. 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.

Zhu, S.; Dong, X.; and Su, H. 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.

Zhuang, B.; Shen, C.; Tan, M.; Chen, P.; Liu, L.; and Reid, I. 2019. Structured binary neural networks for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 413–422.

Appendix

Before we present further details in the following sections, we present an overview on the total amount of computation that was used during this work. We measured the total GPU hours for the three experiments in Section of our paper. These experiments (BaseNet k=1, BaseNet k=4, BoolNet k=4) required 192, 256, and 336 GPU hours respectively, in total: 784 GPU hours.

We have recorded more than 7191 GPU hours for our ablation studies and our intermediate, initial, or discarded experiments (some of which were not presented in the paper), but estimate that a further 1500-2000 hours were needed in experiments before we started measuring the GPU runtime.

Training Details and Further Experimental Results

The training strategy is mostly based on Bethge et al. (2020). More specifically, we use the RAdam optimizer by Liu et al. (2019) with a learning rate of 0.002 without weight decay, use the *cosine learning rate decay* by Guo et al. (2019), and train with a batch size of 256 for 60 epochs. We only use random flipping and cropping of images to a resolution of 224×224 for augmentation and finally normalize the data according to the mean and standard deviation of the dataset. During validation we resize the images to 256×256, and then crop the center with a size of 224×224 (and normalize in the same manner as during training). Our implementation is based on PyTorch (Paszke et al. 2019), and the code can be found in the supplementary material ZIP archive. The implementations of many previous works can not be sped up with `XNOR` and `popcount` (also observed by Fromm et al. (2020)), since they use padding with zeros, which introduces a third value ($\{-1, 0, +1\}$) in the feature map. To circumvent this issue, we use *Replication* padding, which duplicates the outer-most values of the feature map, thus the values are limited to $\{-1, +1\}$. A further difference to previous work, is our progressive weight binarization technique to remove the need for two-stage trainings, as discussed in the following Section.

Progressive Weight Binarization vs. Two-Stage Training We have introduced the progressive weight binarization strategy in Section , Equation 7 and discussed the results briefly in Section . As presented in our main paper, training with progressive weight binarization leads to a higher accuracy, if we train for the same total number of epochs. However, we also conducted an experiment using a linear increase ($\lambda'_t = 1 - t + \epsilon$, $\epsilon = 10^{-6}$) instead of our proposed exponential increase ($\lambda_t = \sigma^t$) of the slope (see Figure 4). We chose σ, so the final λ values are equal, i.e. if $t_{\max}$ represents the final epoch, then $\lambda_{t_{\max}} = \lambda'_{t_{\max}} = 10^{-6}$. The learning curves show that our progressive weight binarization gains the largest advantage by only "initializing" the values during a brief initial phase of the training.

Ablation Study on the Downsample Structure

As described in Section , we modify the 1×1 convolution in the downsampling branch in contrast to many previous works (Rastegari et al. 2016; Liu et al. 2018, 2020b; Martinez et al. 2020). While being helpful for accuracy, the 32-bit 1×1 convolution involves extra computing, memory and energy consumption, which is in conflict with our motivation. Using our multi-slice strategy with $k = 8$, the number of input channels for the 1×1 convolution also increases by the same factor of 8. To counter this increase of 32-bit operations, it could be an option to use 8 groups in the convolution, which would keep the number of 32-bit operations constant, compared to previous work. However, this strategy still conflicts with our motivation to remove most 32-bit operations. Furthermore, the average pooling layer used in previous work, requires additional 32-bit addition and division operations, which could be reduced with either using a max pooling layer or a stride of 2.

Therefore, to find a good downsample module with binary data flow, we first design the downsample template as [Conv$_y$, x, BN, Sign]. In this template, x indicates the different candidate downsample operations (e.g., average pooling, max pooling, or adding stride=2 to the convolution) and y the number of bits used for weights and activations in the convolution.

We conducted a detailed ablation study on the CIFAR100 dataset for both $k = 1$ and $k = 8$ (see Table 2). The results show, that max pooling combined with 1-bit 1×1 convolution (groups = 1) has the same Top-1 accuracy as average pooling combined with 32-bit 1×1 convolution (groups = 8). Thus, we decide to use max pooling instead of average pooling, since it does not involve any 32-bit operations, such as addition and division.

Based on the above analysis, we suggest using the [32-bit Conv (groups = k), AvgPool2d, BN, Sign] structure for the downsample branch if we want to increase accuracy. However, if we intend to build a fully binary data flow, we suggest using the [1-bit Conv (groups = 1), MaxPool2d, BN, Sign] structure (independent of k) instead to balance the accuracy and hardware efficiency. The latter is also the structure we used for our experiments in the main paper.

More Details About the Energy Consumption Simulation

In Table 3, we give an example of calculating the memory consumption among different stages of our network. Compared with regular BNNs with mixed precision data flow, the fully binary representation of BoolNet significantly lowers the memory consumption during inference process. This change leads to less memory access operations to DRAM which has a much higher power consumption than the on-chip SRAM. To the best of our knowledge, our work is the first one to study the impact of memory access on energy consumption. The details of simulation and energy estimation are introduced as follow.

Overall hardware architecture. An illustrative graph on the data flow between the hardware components is provided in Figure 7. In the typical BNN Bi-RealNet, only the convolution is binary, the shortcut branch adopts high precision, and other calculations adopt high precision, too. The corresponding accelerators we designed have different computing modules (but their parallelisms are the same, that is, the computing time of the whole block is roughly the same, and the binary convolution units are exactly the same). In addition,

			AvgPool Acc. (%)		MaxPool Acc. (%)		Stride=2 Acc. (%)	
k	Bits	Groups	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
21	32	1	63.5	87.8	63.0	87.7	60.7	86.4
	1	1	63.1	88.0	62.5	87.2	60.9	86.7
38	32	1	66.0	89.4	67.0	90.0	63.4	87.9
	32	8	65.0	88.0	65.3	88.9	62.2	87.0
	1	1	64.1	88.5	65.0	89.0	62.6	87.3

Table 2: Our ablation study on CIFAR100 regarding different downsampling methods. The number of bits refers to both the input activation and weight binarization of the 1×1 convolution in the shortcut branch.

Table 3: Theoretical minimum memory requirement of all convolution blocks (can differ depending on the implementation). k is the number of slices. The stages have different input size and thus lead to different memory requirements.

Memory	Stage 1 with $64 \times 56 \times 56$			Stage 2 with $128 \times 28 \times 28$		
Usage of	BoolNet (k=1)	BoolNet (k=4)	Regular BNN	BoolNet (k=1)	BoolNet (k=4)	Regular BNN
Weights	36,864	36,864	36,864	147,456	147,456	147,456
Activation	200,704·1 = 200,704	200,704·4 = 802,816	200,704·1 = 200,704	100,352·1 = 100,352	100,352·4 = 401,408	100,352·1 = 100,352
Output & Features	2·200,704·1 = 401,408	2·200,704·4 = 1,605,632	2·200,704·32 = 12,845,056	2·100,352·1 = 200,704	2·100,352·4 = 802,816	2·100,352·32 = 6,422,528
Total	**638,976**	**2,445,312**	**13,082,624**	**448,512**	**1,351,680**	**6,670,336**
Memory	Stage 3 with $256 \times 14 \times 14$			Stage 4 with $512 \times 7 \times 7$		
Usage of	BoolNet (k=1)	BoolNet (k=4)	Regular BNN	BoolNet (k=1)	BoolNet (k=4)	Regular BNN
Weights	589,824	589,824	589,824	2,359,296	2,359,296	2,359,296
Activation	50,176·1 = 50,176	50,176·4 = 200,704	50,176·1 = 50,176	25,088·1 = 25,088	25,088·4 = 100,352	25,088·1 = 25,088
Output & Features	2·50,176·1 = 100,352	2·50,176·4 = 401,408	2·50,176·32 = 3,211,264	2·25,088·1 = 50,176	2·25,088·4 = 200,704	2·25,088·32 = 1,605,632
Total	**740,352**	**1,191,936**	**3,851,264**	**2,434,560**	**2,660,352**	**3,990,016**

for fair comparison, these accelerators have the same size of on-chip memory (192KB for feature map and 288KB for weight) and the same off-chip memory.

Computing unit. The binary convolution units of different BNN accelerators are exactly the same, but other calculation units of BoolNet are simpler. The first is the shortcut branch of downsample blocks. The shortcut branch of traditional BNNs are high-precision, and the high-precision convolution downsampling is adopted. Although the convolution on the shortcut branch accounts only for a small amount of calculation, the power consumption of a high-precision convolution is 37 times that of a binary convolution, and the extra convolution unit also increases the complexity of the circuit. Secondly, regarding batch normalization and binarization, since the shortcut branch has changed from high-precision to binary, the aggregation position of the shortcut branch and the main branch has also changed, so that the binarization and batch normalization can be simplified together, while the calculation of typical BNN can not be simplified, and their power consumption is high. In addition, there is a difference in the complexity of the aggregation operation itself (boolean logic operation vs. 32-bit addition) and the computational overhead of non-linear functions (i.e. RPReLU) added in networks such as ReActNet. These aspects show the efficiency of BoolNet.

Energy consumption per unit. The energy consumption per unit operation of some commonly used components is shown in Figure 6a. However, since the units are not operated the same number of times, the total energy consumption during one inference graph is different. For instance, the energy consumption of Int8 downsampling convolution is $37\times$ larger than binary downsampling[1]. Surprisingly, per unit operation, a 32-bit RPReLU consumes 26% more energy than a binary convolution, Int8 BN consumes about half of a binary convolution, and those two components are commonly used in conjunction with binary convolutions in existing BNNs.

On-chip memory. We use CACTI 6.5 to simulate the power of on-chip SRAM. According to the requirements of the computing unit, we configure the on-chip SRAM to meet the parallelism of the corresponding data reading bandwidth (64 bits for BoolNet and 2048 bits for traditional BNNs), while keeping the total storage unchanged. In addition, we split a large SRAM into multiple SRAMs to meet the requirement that the read time is less than the clock cycle (1ns) of the computing unit. Finally, the simulation software can give the energy consumption of one read or one write of each SRAM unit. For each layer of the network, we know the total number of operations for each type of operation. According to the circuit parallelism, we can calculate the number of cycles. Then, according to the amount of data that needs to be read from (or written to) SRAM in each cycle, we can

[1] 37=504$\times$8/108.8, where Int8 Conv has only 1/8 of the parallel capability of BConv.

get the energy that the accelerator spends to access on-chip SRAM.

Off-chip memory. Due to the limited amount of on-chip memory, it is inevitable to save some data to (or read from) off-chip DRAM in BNN computing. In our BoolNet design, due to the large total number of weights, all BNN accelerators need to read weights from DRAM and write to SRAM before the computation of each layer. In addition, for traditional BNN, the intermediate feature maps are larger, which cannot be completely cached on-chip. It is also necessary to save the extra part to DRAM, to read it back in the next layer. With the amount of read-write operations of data to (and from) DRAM and SRAM, the power consumption data of DRAM read-write operations (SRAM has been given by the CACTI simulation in the previous step) is also needed to estimate the overall energy consumption. We use the DDR4 Power Calculator provided by Micron, to configure a DDR UDIMM module composed of four 8Gb x16 chips, which adopts the speed grade of -075E, and the maximum transmission rate is 2666MT/s. The calculator gives the average energy consumption of reading and writing data with 64 bits parallelism.

Detailed throughput. The detailed throughput of the accelerators for BaseNet, BoolNet, BiRealNet, ReActNet, XnorNet are 2125, 2044, 2237, 2237, 2237 samples per second, respectively.

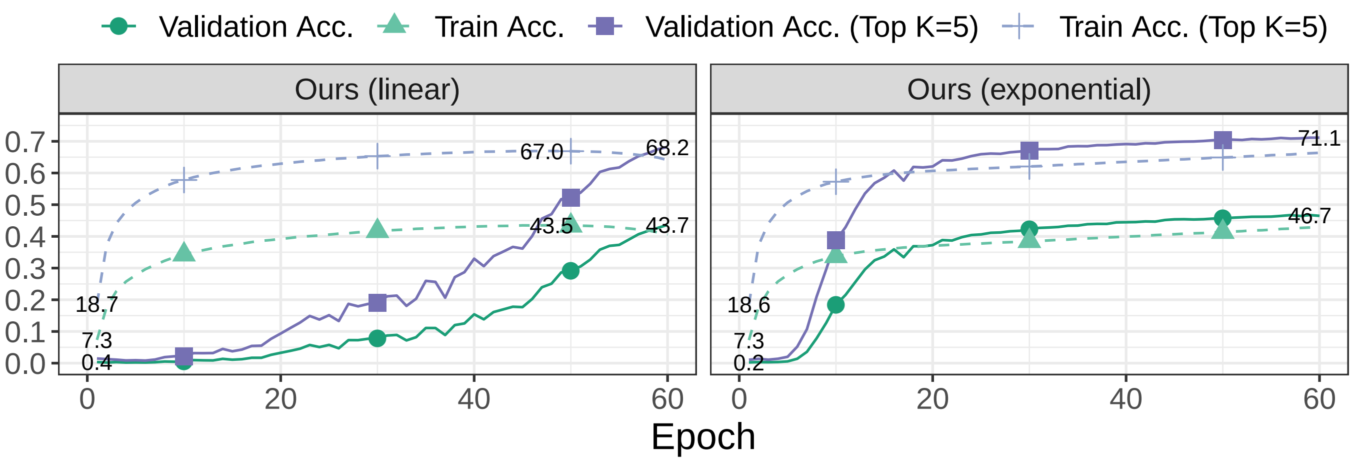


Figure 4: The training and validation accuracy curves of our proposed *Progressive Weight Binarization*. An exponential increase of the slope leads to much better results, than a linear increase.

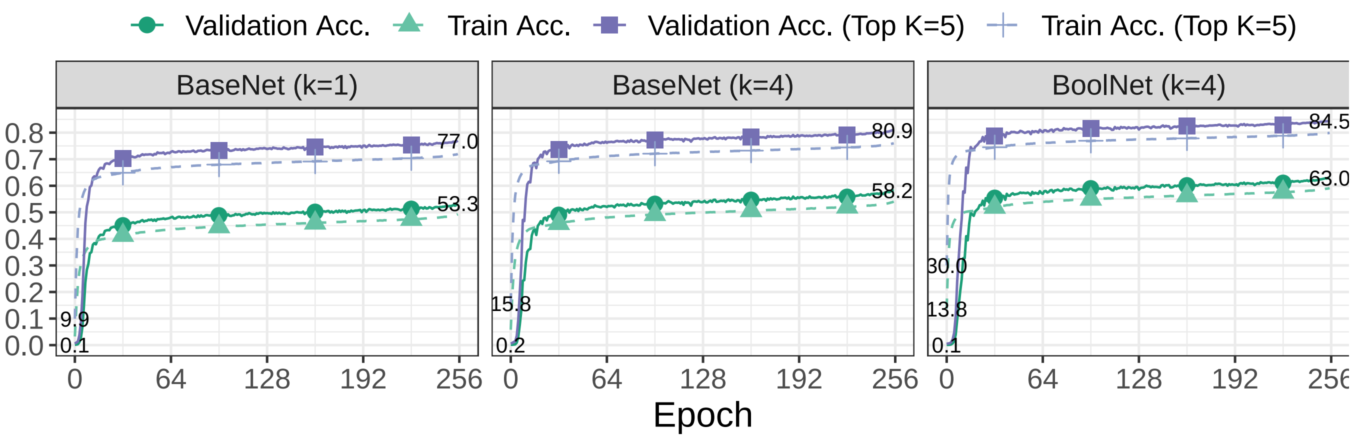
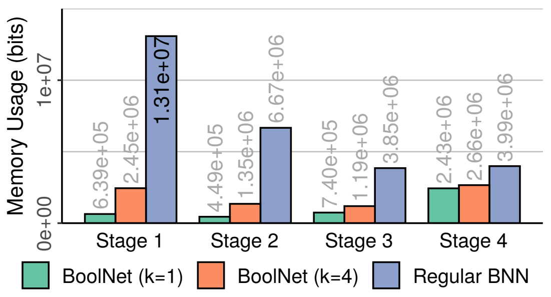


Figure 5: The training and validation accuracy curves of our trainings discussed in the comparison to the state-of-the-art BNNs in Section .

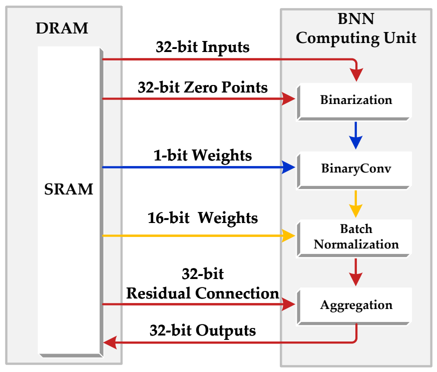
Operation	Power (mw)	Area (um^2)	Operation	Power (mw)	Area (um^2)
BConv	108.8	131737	Int8 Conv(1/8)	504	836269
-	-	-	Int Agg	43.5	53238
16-bit Sign	1.4	7956	32-bit Sign	3.3	13548
32-bit RPReLU	137.6	310671	Int8 BN	50.1	274606

(a) Energy consumption per unit operation and circuit area of commonly used components.

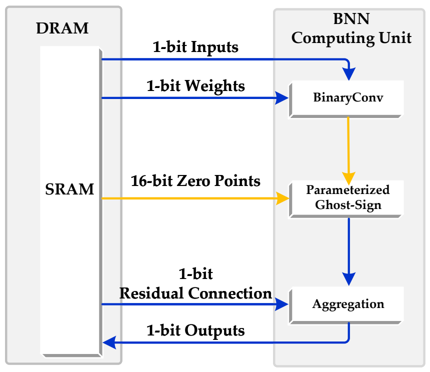


(b) Memory usage comparison between blocks of different stages.

Figure 6: A theoretical memory usage comparison of one convolution block between BoolNet and previous work. Actual numbers can differ during implementation, but BoolNet shows significantly lower memory usage, especially in early stages, even when using our Multi-slice strategy with $k = 4$.



(a) Bi-RealNet Data Flow on Hardware



(b) BoolNet Data Flow on Hardware

Figure 7: Hardware data flow comparison between Bi-RealNet and BoolNet.