BoolNet: Minimizing the Energy Consumption of Binary Neural Networks

Nianhui Guo*[1], Joseph Bethge*[1], Haojin Yang[1], Kai Zhong[2],
Xuefei Ning[2], Christoph Meinel[1] and Yu Wang[2]
[1] Hasso Plattner Institute, Germany
[2] Department of Electronic Engineering, Tsinghua University, China
`{nianhui.guo,joseph.bethge,haojin.yang,christoph.meinel}@hpi.de`
`{zhongk19,nxf16}@mails.tsinghua.edu, yu-wang@tsinghua.edu.cn`

Abstract

Recent works on Binary Neural Networks (BNNs) have made promising progress in narrowing the accuracy gap of BNNs to their 32-bit counterparts. However, the accuracy gains are often based on specialized model designs using additional 32-bit components. Furthermore, almost all previous BNNs use 32-bit for feature maps and the shortcuts enclosing the corresponding binary convolution blocks, which helps to effectively maintain the accuracy, but is not friendly to hardware accelerators with limited memory, energy, and computing resources. Thus, we raise the following question: *"How can accuracy and energy consumption be balanced in a BNN network design?"* We extensively study this fundamental problem in this work and propose a novel BNN architecture without most commonly used 32-bit components: *BoolNet*. Experimental results on ImageNet demonstrate that BoolNet can achieve 4.6× energy reduction coupled with 1.2% higher accuracy than the commonly used BNN architecture Bi-RealNet [30]. Code and trained models are available at: `https://github.com/hpi-xnor/BoolNet`.

Interesting point of View.

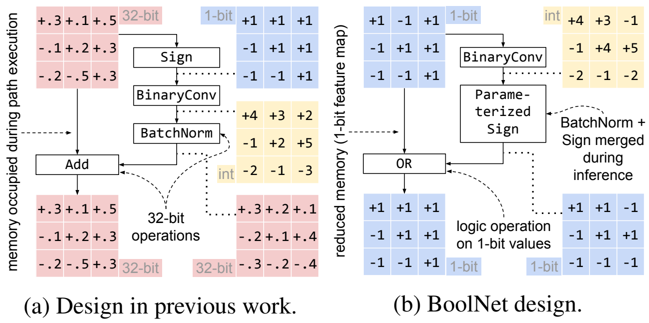
1 Introduction

The recent success of *Deep Neural Networks* (DNNs) is like the jewel in the crown of modern AI waves. However, the large size and the high number of operations cause the current DNNs to heavily rely on high-performance computing hardware, such as GPU and TPU. Training sophisticated DNN models also results in excessive energy consumption and CO_2 emission, e.g., training the OpenAI's GPT-3 [5] causes as much CO_2 emissions as 43 cars during their lifetime [38]. Moreover, their computational expensiveness strongly limits their applicability on resource-constrained devices such as mobile phones, IoT devices, and embedded devices. Various works aim to solve this challenge by reducing memory footprints and accelerating inference. We can roughly categorize these works into the following directions: network pruning [16, 17], knowledge distillation [12, 39], compact networks [22, 21, 42, 32, 43], and low-bit quantization [10, 41, 47, 23]. From the latter, there is an extreme case, Binary Neural Networks (BNNs) (first introduced by [11]) that uses only 1 bit for weight and activation.

As shown in the literature [41], BNNs can achieve 32× memory compression and up to 58× speedup on CPU, since the conventional arithmetic operations can be replaced by bit-wise `xnor` and `bitcount` operations. However, BNNs suffer from accuracy degradation compared to their 32-bit counterparts. For instance, XNOR-Net leaves an 18% accuracy gap to ResNet-18 on ImageNet classification [41]. Therefore, recent efforts (analyzed in more detail in Section 2) mainly focus on narrowing the

*Equal contribution.

Preprint. Under review.



(a) Design in previous work.

(b) BoolNet design.

Method	Bitwidth (W/A/F)	Energy (mJ)	Top-1 Acc.	OPs $(\cdot 10^8)$
Bi-Real-Net [30]	1/1/32	3.90	56.4%	1.63
BoolNet (ours)	1/1/4	0.84	57.6%	1.64
BaseNet (ours)	1/1/1	0.61	48.9%	1.51

(c) BoolNet reduces energy consumption by 4.6× compared to Bi-RealNet.

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

accuracy gap, including specific architecture design [30, 4, 3, 29], real-valued weight and activation approximation [27, 48], specific training recipes [34], a dedicated optimizer [20], leveraging neural architecture search [6, 46] and dynamic networks [7]. In the existing work, efficiency analysis usually only considers the theoretical instruction counts. However, memory usage, inference efficiency and energy consumption, which are essential to practical applications, have received little attention. Furthermore, [14] points out that the theoretical complexity is often inconsistent with the actual performance in practice and measurable performance gains on existing BNN models are hard to achieve as the 32-bit components in BNNs (such as BatchNorm, scaling, and 32-bit branches) become bottlenecks. Using 32-bit information flow (e.g., 32-bit identity connections, 32-bit downsampling layers are equipped by almost all latest BNNs, see Figure 1a), and multiplication/division operations (in BatchNorm, scaling, average pooling etc.) significantly increase the memory usage and power consumption of BNNs and are thus unfriendly to hardware accelerators. For these reasons, even if BNNs have achieved MobileNet-level accuracy with a similar theoretical number of OPs [3, 34], they still cannot be used as conveniently as compact networks [22, 21, 42].

但是 访存能量消
耗更大吗?

In this paper, we extensively study the trade-off between BNN's accuracy and hardware efficiency. We propose a novel BNN architecture: BoolNet, which replaces most commonly used 32-bit components (see Section 3). First, ① BoolNet uses binary feature maps in the network - by shifting the starting point of the shortcuts from the BatchNorm (BN) layer to the Sign function (see Figure 1b) - and uses Boolean functions instead of 32-bit additions to accumulate features. Second, during inference, we fuse the BN layer into the Sign function through a lossless transformation, ② thereby effectively removing the MAdds brought by BN. Other changes include removing components that require additional 32-bit multiplication/division operations: (1) PReLU, (2) average pooling, and (3) binary downsampling convolutions. We further propose ③ a *Multi-slice strategy* to help alleviate the loss of representational capacity incurred by binarizing the feature maps and shortcut connections. We show the effectiveness of our proposed methods and the increased energy efficiency of BoolNet with experiments on the ImageNet dataset [13]. The results show the key benefit of BoolNet: a reasonable accuracy coupled with a higher energy efficiency over state-of-the-art BNNs (see Figure 1c for a brief summary and Section 4 for more details). The energy data is obtained through hardware accelerator simulation (see Section 4.4 for details). We summarize our main contributions as follows:

- The first work studying the effects of 32-bit layers often used in previous works on BNNs.
- A novel BNN architecture BoolNet with minimal 32-bit components for higher efficiency.
- A Multi-slice strategy to alleviate the accuracy loss incurred by using 1-bit feature maps.
- State-of-the-art performance on the trade-off between accuracy and energy consumption with a 4.6× lower power consumption than Bi-RealNet [30] and 1.2% higher accuracy.

known X

2 Related Work

In recent years, *Efficient Deep Learning* has become a research field that has attracted much attention. Technical directions, such as, compact network design [22, 21, 42, 45, 32], knowledge distillation [12, 39], network pruning [16, 17, 26, 19], and low-bit quantization [10, 41, 30, 29, 3] are proposed for model compression and acceleration. The efficient models have evolved from the earliest handcrafted designs to the current use of neural architecture search to search for the best basic block and overall network structure [43, 21, 44, 40]. The criterion of efficiency evaluation has also changed from instruction and parameter counts to more precise measurements of actual memory and operating efficiency on the target hardware [8, 9].

Binary Neural Networks were first introduced by Courbariaux et al. [11] and their initial attempt only evaluated on small datasets such as MNIST [25], CIFAR10 [24] and SVHN [36]. The follow-up XNOR-Net [41] proposes channel-wise scaling factors for approximating the real-valued parameters, which achieves 51.2% top-1 accuracy on ImageNet. However, there is an 18% gap compared with its 32-bit counterpart, ResNet-18. Therefore, recent efforts mainly focused on narrowing the accuracy gap. WRPN [35] shows that expanding the channel width of binary convolutions can obtain a better performance. In ABC-Net [27] and GroupNet [48], instead of using a single binary convolution, they use a set of k binary convolutions (referred to as binary bases) to approximate a 32-bit convolution. This sort of method achieves higher accuracy but increases the required memory and number of operations of each convolution by the factor k. Bi-RealNet [30] proposes using real-valued (32-bit) shortcuts to maintain a 32-bit information flow, which effectively improves the accuracy. This design strategy became a standard for later work e.g., [4, 3, 29]. Martinez et al. [34] propose using a real-valued attention mechanism and well-tuned training recipes to boost the accuracy further. Thanks to the special architecture design, the recent MeliusNet [3] and ReActNet [29] achieve MobileNet-level accuracy with similar number of theoretical operations. Other attempts, such as leveraging neural architecture search [6, 46] and dynamic networks [7], show that those successful methods on regular real-valued networks are also effective for BNN. Often, with improved accuracy, 32-bit components are used more frequently as well, such as PReLU and BatchNorm after each binary convolution [29], real-valued attention module [34] and scaling factors, etc. On the contrary, efficiency analysis in the literature often only considers the theoretical operation number. However, the memory usage and the actual energy consumption has received very little attention so far.

3 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 3.1). 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 3.2). 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 (in Section 3.3). The focus on boolean operations and binary feature maps leads to the name of our network: **BoolNet**.

good ✓
&
sharp

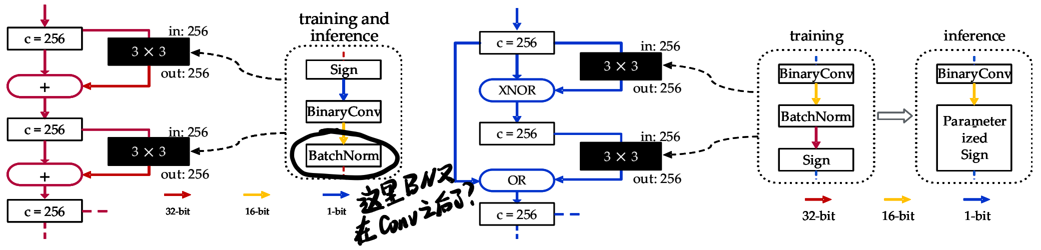
3.1 Improving Accuracy with Additional 32-bit Components

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 in [14] and further discussed in Section 4.4). The following list summarizes the 32-bit components commonly used in the latest BNNs:

- The **channel-wise scaling factor** was first proposed by XNOR-Net [41] for approximating the 32-bit parameters. It increases the value range of activation and weight.
- Bi-RealNet [30] 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 (cf. Figure 2a).

→ why 1x1?

- XNOR-Net [41] uses **32-bit 1×1 downsampling** convolutions, which is also used by most subsequent methods [30, 34, 3]. [4] shows that this simple strategy can achieve about 3.6% Top-1 accuracy gains on ImageNet based on a binary ResNet-18 model.
- [34, 6, 7] show that **PReLU activation** effectively improves accuracy of BNNs. ReActNet [29] constructs the RPReLU activation function and uses it before every sign function.



这里BN又在Conv之后了?

(a) Typical binary basic block with 32-bit shortcuts and Batch Normalization layer.

(b) Our binary block design with logic shortcuts without 32-bit operations. c indicates the number of channels.

Figure 2: Comparison between a conventional binary convolution block with 32-bit shortcuts (a) and our proposed BoolNet convolution block with 1-bit logic shortcuts (b).

- Real-to-Binary Net [34] reuses the 32-bit activation after BN through 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.

3.2 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 replace them with boolean operations. We further propose a new basic convolution block without 32-bit operations, as shown in Figure 2b, where we rearranged the order of convolution basic block as {BinaryConv-BatchNorm-Sign}, so that all feature maps are binary. These general changes constitute our BoolNet baseline, in short *BaseNet*.

3.2.1 Integrating BatchNorm into Sign Function

Most studies on binary neural architecture design have kept the 32-bit BatchNorm (BN) layer in both the training and testing stages [23, 41, 30, 29, 3]. However, using a 32-bit BN right after the 1-bit convolution layer decreases the computational efficiency on 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.

During the training phase, the batch normalization layer normalizes feature maps with an 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:

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

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. $||\ldots||$ 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:

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

★ 这个fuse有点意思.

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

3.2.2 1-bit Logic Shortcuts

The residual shortcut is usually a 32-bit branch which branches off after the BatchNorm (BN) and pointwise addition in previous work [30, 29, 3]. We modify the residual shortcut in two aspects: (i) We shift the starting point of the shortcut connection from the output of BN to the output of the Sign function. (ii) We utilize the logic operators XNOR and OR for merging the binary features to the consecutive block (instead of 32-bit addition). Based on this novel shortcut design, called **Logic Shortcuts**, the feature maps in each stage of the network is completely binary without 32-bit operations. It reduces the memory consumption of the intermediate feature maps by 32× and is the first binary residual structure proposed for BNNs to the best of our knowledge.

Although boolean operators can fulfill the needs of fusing binary information branches, they are not inherently differentiable. To allow our network with boolean operators to be trained using back-propagation, we replace XNOR and OR in the training stage with the following differentiable terms:

$$\text{XNOR}\,(x',\,y') = x \cdot y \qquad \text{OR}\,(x',\,y') = 2 \cdot \text{Min}\left(1, \frac{x+y}{2}+1\right) - 1 \tag{5}$$

where $x,\,y \in \{-1,+1\}$ denote the binary variables during training (and $x',\,y' \in \{0,1\}$ during inference). This allows us to convert them back to logic operators during inference loss-free.

In summary, our proposed basic block (see Figure 2b) maximizes efficiency by using only 1-bit operations during inference and uses two different logic shortcuts based on XNOR and OR. This is contrary to conventional BNN blocks [30, 29], which use 1-bit only for convolution layers, whereas other components are 32-bit or 16-bit (cf. Figure 2a),

3.2.3 Further Reducing 32-bit Operations

We rarely use the PReLU activation function, which is commonly used in most literature [30, 34] 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 factor as suggested by [30, 4]. Furthermore, we binarize the 1×1 downsampling convolution, which is usually kept full-precision in previous methods [30, 34] without the severe accuracy loss described in previous work [41, 30]. This further reduces the number of 32-bit operations and 32-bit parameters in BoolNet, but due to space limitations, we discuss the details on, alternatives to, and results of these changes in the supplementary material. There are two components using 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 [41], thus we leave the investigation of alternatives for these special cases for future work.

3.3 BoolNet: Enhancing Binary Information Flow

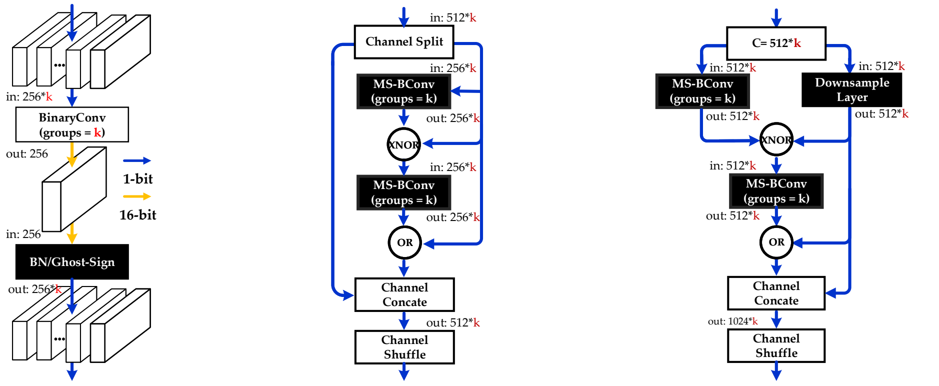
The network design changes explained in the previous section, constitute our BoolNet baseline, called *BaseNet*. Although it uses a completely binary information flow which minimizes the energy and memory consumption, the representative capacity of BaseNet is drastically degraded compared to its 32-bit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

Multi-slices Binary Convolution. Instead of using a single 1-bit value for each 32-bit value in a regular BNN, our multi-slice strategy proposes of using a set of k 1-bit values. The key intention is to reduce the information loss caused by the sign function. We consider the typical binarization process Sign(x_i, zero-point) as a special case of single-slice numerical projection. Thus, we propose a multi-slice projection strategy for binary convolution to retain more relative magnitude information. Specifically, we redesign sign function as follows:

$$x_i^b = \text{Sign}(x_i, b_n), \tag{6}$$

where b_n indicates a set of constant bias:

$$b_n = \frac{\pm 2n}{k}, \text{where } n = 0, 1, ..., k/2 \tag{7}$$



(a) Multi-Slices binary convolution

(b) BoolNet basic block

(c) BoolNet downsample block

Figure 3: Detail Architecture of BoolNet. To enhance the information flow, we modify the baseline architecture from two aspects: a) Reducing information loss through multi-slices binary convolution. b) Strengthening information propagation by features reusing.

We adopt b_n to conveniently expand the channel dimension to enhance the capacity of the binary feature map. If $n = 0, k = 1$, Equation (7) degenerates to the ordinary sign function. In Equation (6), x_i^b denotes the binary projection output with the dimension of $[N, C * k, H, W]$, which will be fed into the subsequent binary convolution layer. The constant k also denotes the group number of the convolution. That is, by setting the number of groups to k in each convolution, the overall amount of parameters and operations of each convolution is unchanged. Motivated by FReLU [31], we enhance the first multi-slices projecting module, after the input convolution of network, with a **Local Adaptive Shifting** module. This module consists of a depth-wise 3×3 convolution and a batch normalization layer and is able to adaptively change the zero points of each pixel, in a light-weight manner. For simplicity, the multi-slices binary convolution is referred to as **MS-BConv**, subsequently. Figure 3b shows the detailed block design of MS-BConv.

Strengthening The Information Propagation in BoolNet. The layer-by-layer feature extraction and accumulation mechanism are key reasons deep neural networks have strong representative capacity. Unlike typical residual shortcuts, which accumulates information from shallow to deep based on addition operation, logic shortcuts using boolean operators such as XNOR and OR can only represent True and False states, making them difficult to accumulate and propagate information. To alleviate this bottleneck, we strengthen feature propagation by reusing features. In ShuffleNet-V2 [33], the input tensor is divided into two equal parts, the first half is used for feature extraction, and the other half is directly copied and concatenated with the extracted features. Inspired by its characteristics of information fusion and retention, we use a similar method to enhance the information retention capability of the BoolNet block. As demonstrated in Figure 3b, the feature extraction branch consists of two MS-BConv modules with logic shortcuts, and the other branch remains identity. Two branches are concatenated and followed by channel shuffle, ensuring that the features from different layers are uniformly distributed. Figure 3c shows the downsampling block design of BoolNet, where no channel splitting is required, and it doubles the number of channels in the output. Changing this information accumulation mechanism constitutes our proposed **BoolNet** over the *BaseNet* (as referred to in Section 4).

3.4 Training with Progressive Weight Binarization

Though we intend to build highly efficient BNNs with fully binary information flow, this strategy make the network more sensitive to weight initialization during training. Traditional methods have tried alleviate similar problem through two-stage training [34, 29], which makes training more complicated. In this paper, we adopt a progressive binarization technique based on the traditional Hardtanh-STE method [11]. This can be viewed as a smooth version of previous multi-stage training. Specifically, in the training phase, a differentiable function $F(x)$ is used to replace sign function.

Table 1: Our ablation study on ImageNet [13] regarding accuracy, number of 32-bit operations (FLOPs), 1-bit operations (BOPs), and model size. We **highlighted** the positive effects of *Logic Shortcuts*, *Local Adaptive Shifting*, and *Multi-slice Convolution* (k denotes the number of slices).

Network Configuration (k=1)	BaseNet						BoolNet					
	Top 1 Acc.	Top 5 Acc.	FLOPs ($\cdot 10^8$)	BOPs ($\cdot 10^9$)	OPs ($\cdot 10^8$)	Model Size	Top 1 Acc.	Top 5 Acc.	FLOPs ($\cdot 10^8$)	BOPs ($\cdot 10^9$)	OPs ($\cdot 10^8$)	Model Size
Baseline (no shortcuts)	46.26%	70.84%	1.23	1.76	1.51	3.49 MB	-	-	-	-	-	-
+ Logic Shortcuts (XNOR/OR)	**48.60%**	**72.79%**	1.23	1.76	1.51	3.49 MB	49.92%	74.17%	1.23	2.01	1.55	3.71 MB
+ Local Adaptive Shifting	48.83%	73.19%	1.26	1.76	1.53	3.49 MB	**51.51%**	**75.41%**	1.26	2.01	1.57	3.71 MB

k	BaseNet (with Logic Shortcuts)						BoolNet (with Logic Shortcut and Local Adaptive Binarization)					
	Top 1 Acc.	Top 5 Acc.	FLOPs ($\cdot 10^8$)	BOPs ($\cdot 10^9$)	OPs ($\cdot 10^8$)	Model Size	Top 1 Acc.	Top 5 Acc.	FLOPs ($\cdot 10^8$)	BOPs ($\cdot 10^9$)	OPs ($\cdot 10^8$)	Model Size
1	48.60%	72.79%	1.23	1.76	1.51	3.49 MB	51.51%	75.41%	1.26	2.01	1.57	3.71 MB
4	**52.15%**	**75.89%**	1.23	2.01	1.55	3.56 MB	**54.45%**	**77.83%**	1.26	2.50	1.65	3.84 MB
8	52.51%	76.34%	1.23	2.35	1.60	3.65 MB	-	-	-	-	-	-

During the forward, the slope of this function is adjusted by a single scalar λ. As the slope shrinks, the weight gradually changes from 32-bit to 1-bit. During backward propagation, we approximate $F(x/\lambda)$ with $F(x/1)$, which escapes BoolNet from the gradient vanishing as λ decreases. In the testing phase, we use traditional sign function for inference. The whole process can be formulated as:

$$F(x, \lambda) = \lim_{\lambda \to 0} \text{Hardtanh}\left(\frac{x}{\lambda}\right) \simeq \text{Sign}(x). \tag{8}$$

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.

4 Experiments

We use the task of image classification on the ImageNet [13] dataset as our main means of evaluation. In the following section, we first present the training details for our experiments. Afterwards, we study the effects of our proposed network design changes, (in Section 4.2) and the *Multi-slice* convolution (in Section 4.3) and analyze the energy consumption of BoolNet and other recent work on BNNs (in Section 4.4) and compare our model accuracy to state-of-the-art BNN models (in Section 4.5).

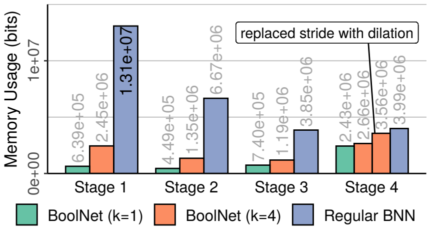
4.1 Training Details

Our general training strategy and hyperparameters are mostly based on [3], the exact hyperparameters, training details and training code are available in the supplementary material.

As an alternative to the two-stage training approach, as described in [29, 34], we proposed progressive weight binarization (see Section 3.4, Equation 8). In the following experiments, we used $\sigma = 0.965$ and thus $\lambda = 0.965^t$, with t being the number of iterations divided by 1000 (i.e. λ is multiplied by 0.965 every 256000 samples). Note, that the progressive weight binarization is replaced by a regular sign function during the validation pass. The two stage training strategy aims to provide a good initialization for a BNN training, by first training a model with 1-bit activations/32-bit weights and weight decay of 10^{-5}, and use it to initialize the training of a 1-bit activations/1-bit weights model. We tested the effect of both strategies with a plain ResNet-like model with binary feature maps and our proposed Logic Shortcuts on ImageNet. The two-stage training (trained 60 epochs in each stage - a total of 120 epochs) achieved 49.60% accuracy. Our progressive weight binarization achieves 48.39% when training for **60** epochs, but achieves 50.19% when training for **120** epochs. Thus we deduce that our training strategy effectively removes the need for a two stage training (based on a similar total training time) and leads to a similar or better result.

Operation	Power (mw)	Area (um^2)	Operation	Power (mw)	Area (um^2)
BConv	108.8	131737	Int8 Conv(1/8)	504	836269
1-bit Agg	1.4	2150	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 and circuit area of different operations (see Section 4.4 for a detailed explanation).



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

Figure 4: 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 $k = 4$.

4.2 Ablation on Network Design

For brevity, we refer to our BoolNet baseline (consisting of our changes described in Section 3.2) as *BaseNet*. When we apply our changes regarding the information propagation described in Section 3.3, we refer to it as *BoolNet*. In the following section, we study the effects of all of our proposed network design changes, in particular of the *Logic Shortcut* (see Section 3.2.2) and the *Local Adaptive Shifting* module (see Section 3.3) on the ImageNet dataset. Our results (see upper half of Table 1) show, that adding *Logic Shortcuts* to a plain BaseNet (without shortcuts) to accumulate 1-bit features with XNOR and OR increases accuracy by 2.4% with minimal extra cost. We infer that such shortcuts can be a suitable replacement for the addition that were used to accumulate 32-bit features in previous BNNs and use them in all our network designs. However, the *Local Adaptive Shifting* module is only effective for our proposed BoolNet (providing an accuracy increase of 1.59%) and does not provide a benefit for a BaseNet-style network (accuracy is increased by only 0.23%) compared to the extra cost.

4.3 Ablation on the Multi-slice Convolution

We also evaluated whether using Multi-slice Convolutions (see Section 3.3) can reduce the accuracy loss caused by using 1-bit feature maps (k denotes the number of slices). Our results on ImageNet (see lower half of Table 1) show, that using $k = 4$ increases accuracy significantly for our BaseNet (3.55%) and BoolNet (2.94%) architectures. Although the convolutions used throughout the network use a number of groups equal to k to keep the required parameters and operations constant, operations and parameters are still slightly increased in the 1×1 convolution in the downsampling branch which uses all channels. Overall $k = 4$ leads to a slight increase of operations (compared to $k = 1$), however is still significantly lower than compared to previous work [34, 30]. However, further increasing k to $k = 8$ only slightly improves accuracy (by 0.36%), but again increases operations and parameters in the downsampling branch. Therefore, $k = 4$ provides the best trade-off, which is further proven in the following section. Furthermore, using the Multi-slice strategy allows us to use a downsampling branch *without 32-bit components without accuracy degradation* (using 1-bit 1×1 convolutions) and use this design in our comparison to state-of-the-art. (Due to space limitations, the details on our downsampling branch design are in the supplementary material.)

组卷积会在downsample处增加op数.

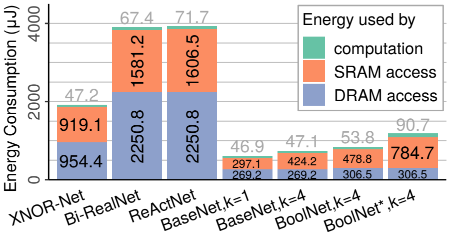
4.4 Energy Consumption Evaluation

This section evaluates the energy consumption of BoolNet and several classic BNN architectures through hardware simulation. We design five accelerators for five BNNs in RTL language, and the power and area of computing circuits are given by Design Compiler (DC) simulation with TSMC 65nm process and 1GHz clock frequency. We further evaluate the energy consumption of on-chip SRAM access and off-chip DRAM access by using CACTI 6.5 [1], and the power calculator of DDR provided by Micron [2]. The above components sum the overall energy consumed by a single inference pass.

Memory access and computation are the primary factors that affect energy consumption of a hardware accelerator. However, in the existing BNNs, efficiency analysis only considers the theoretical instruction counts [30, 34, 3, 29] while the impact of memory access has been neglected. A theoretical

Methods	Bitwidth (W/A/F)	Energy Consumption	Top-1 Acc.	OPs $(\cdot 10^8)$
ReActNet (Bi-Real) [29]	1/1/32	3.93mJ	65.9%	1.63
Bi-RealNet [30]	1/1/32	3.90mJ	56.4%	1.63
XNOR-Net [41]	1/1/32	1.92mJ	51.2%	1.59
BoolNet*, k=4 (ours)	1/1/4	1.18mJ	59.6%	1.76
BoolNet, k=4 (ours)	1/1/4	0.84mJ	57.6%	1.64
BaseNet, k=4 (ours)	1/1/4	0.74mJ	55.1%	1.54
BaseNet, k=1 (ours)	1/1/1	**0.61mJ**	48.9%	1.51

(a) The advantage of BoolNet is reduced energy consumption.



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

Figure 5: Comparison between BoolNet and state-of-the-art BNNs. The energy consumption is calculated through hardware simulations. BoolNet* uses dilation instead of stride in the last stage.

analysis (see Figure 4b) of the required memory shows that the total memory by BoolNet is much lower than previous BNNs, especially during the earlier stages of the network. This analysis also shows that using dilation in the last stage of BoolNet still uses less memory for convolution blocks than in previous BNNs. Our energy evaluation results (see Figure 5b) show that the energy consumption of computing units accounts for a small proportion in the whole calculation. Our design achieves higher energy efficiency due to a lower memory access. In other BNNs, preserving and reading 32-bit feature maps drastically increase energy consumption. Since the overall memory usage of BoolNet is minimal, it requires much less DRAM access than the others. Generally, DRAM has much higher power consumption than SRAM.

Furthermore, the energy consumption of some commonly used components is shown in Figure 4a. For instance, the energy consumption of Int8 downsampling convolution is 37× larger than binary downsampling[2]. The *Logic Shortcut* aggregation is 31× more energy efficient than additive aggregation. Surprisingly, 32-bit PReLU 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 previous BNNs. More implementation and evaluation details can be found in supplementary materials.

4.5 Comparison to State-of-the-Art BNNs

For our comparison to state-of-the-art BNNs, we replaced the Cross-Entropy loss with a knowledge distillation approach, based on the implementation of [29] with a 32-bit ResNet-34 [18] as the teacher model and train the models for 80 or 90 epochs instead of 60 epochs. (Due to limited hardware resources, we were not able to choose a longer training time, but suspect increasing the training time, e.g. to 120 epochs, could improve the results.)

Removing 32-bit elements from previous BNNs (e.g. ReActNet [29]) leads to an energy reduction by up to 6× (BaseNet with k=1), but incurs an accuracy drop of 17% (see Table 5a). Using the proposed Multi-slice strategy (k=4) reduces the accuracy drop by 6.2% and still achieves 5.3× energy reduction. Our BoolNet design further increases the accuracy by 2.5%, but requires 12% more energy (for a 4.5× reduction). Compared to the result of Bi-RealNet [30], which has been the basis for other works [34] BoolNet with k=4 provides an accuracy improvement of 1.2% (and a 4.5× energy reduction). The accuracy of our BoolNet can be further increased with common techniques, such as replacing stride with dilation (denoted with a star*) during the last stage of the network, which increases accuracy by 2% (and yields a 3.3× reduction of energy). Overall our results show that our proposed BaseNet and BoolNet can achieve significant energy reduction with little accuracy loss compared to recent state-of-the-art models.

5 Conclusion

In this paper, we studied how to balance energy consumption and accuracy of binary neural networks. We proposed several simple yet useful strategies to remove or replace 32-bit components from BNNs.

[2] 37=504×8/108.8, where Int8 Conv has only 1/8 of the parallel capability of BConv.

Our novel BoolNet with fully binary information flow is constructed and still maintains reasonable accuracy. Experiments on ImageNet and the hardware simulations show that (1) theoretical number of operations does not fully reveal the actual efficiency and (2) BoolNet is more energy-efficient with less computing requirements, lower memory usage and lower energy consumption. We believe this is orthogonal to the goals of previous works and a meaningful first step towards achieving extremely efficient BNNs.

References

[1] CACTI. `http://www.hpl.hp.com/research/cacti/`. Accessed: 2021-05-28.

[2] Micron. `https://media-www.micron.com/-/media/client/global/documents/products/data-sheet/modules/parity_rdimm/asf9c512x72pz.pdf?rev=32d87a7b4a2b4d05ae8d2a047361700d`. Accessed: 2021-05-28.

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

[4] Joseph Bethge, Haojin Yang, Marvin Bornstein, and Christoph Meinel. Binarydensenet: developing an architecture for binary neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[6] Adrian Bulat, Brais Martinez, and Georgios Tzimiropoulos. Bats: Binary architecture search. In *European Conference on Computer Vision*, 2020.

[7] Adrian Bulat, Brais Martinez, and Georgios Tzimiropoulos. High-capacity expert binary networks. In *International Conference on Learning Representations*, 2021.

[8] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

[9] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[10] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[11] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[12] Elliot J Crowley, Gavin Gray, and Amos J Storkey. Moonshine: Distilling with cheap convolutions. In *Advances in Neural Information Processing Systems*, pages 2888–2898, 2018.

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[14] Joshua Fromm, Meghan Cowan, Matthai Philipose, Luis Ceze, and Shwetak Patel. Riptide: Fast end-to-end binarized neural networks. *Proceedings of Machine Learning and Systems*, 2:379–389, 2020.

[15] Jian Guo, He He, Tong He, Leonard Lausen, Mu Li, Haibin Lin, Xingjian Shi, Chenguang Wang, Junyuan Xie, Sheng Zha, Aston Zhang, Hang Zhang, Zhi Zhang, Zhongyue Zhang, and Shuai Zheng. GluonCV and GluonNLP: Deep Learning in Computer Vision and Natural Language Processing. *arXiv preprint arXiv:1907.04433*, 2019.

[16] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. 2015. cite arxiv:1510.00149Comment: Published as a conference paper at ICLR 2016 (oral).

[17] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[19] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

[20] Koen Helwegen, James Widdicombe, Lukas Geiger, Zechun Liu, Kwang-Ting Cheng, and Roeland Nusselder. Latent weights do not exist: Rethinking binarized neural network optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[21] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the*

IEEE International Conference on Computer Vision, pages 1314–1324, 2019.

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

[23] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 4114–4122, 2016.

[24] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[25] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[26] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 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, 2017.

[27] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[28] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the Variance of the Adaptive Learning Rate and Beyond. *arXiv preprint arXiv:1908.03265*, 2019.

[29] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. Reactnet: Towards precise binary neural network with generalized activation functions. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XIV*, volume 12359 of *Lecture Notes in Computer Science*, pages 143–159. Springer, 2020.

[30] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. 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)*, pages 722–737, 2018.

[31] Ningning Ma, Xiangyu Zhang, and Jian Sun. Funnel activation for visual recognition. *arXiv preprint arXiv:2007.11824*, 2020.

[32] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.

[33] N. Ma, X. Zhang, H. T. Zheng, and J. Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *European Conference on Computer Vision*, 2018.

[34] Brais Martinez, Jing Yang, Adrian Bulat, and Georgios Tzimiropoulos. Training binary neural networks with real-to-binary convolutions. In *International Conference on Learning Representations*, 2020.

[35] Asit Mishra, Eriko Nurvitadhi, Jeffrey J Cook, and Debbie Marr. Wrpn: Wide reduced-precision networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[36] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[37] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

[38] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. *arXiv preprint arXiv:2104.10350*, 2021.

[39] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. In *ICLR (Poster)*. OpenReview.net, 2018.

[40] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10428–10436, 2020.

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

[42] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[43] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[44] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[45] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.

[46] Tianchen Zhao, Xuefei Ning, Xiangsheng Shi, Songyi Yang, Shuang Liang, Peng Lei, Jianfei Chen, Huazhong Yang, and Yu Wang. Bars: Joint search of cell topology and layout for accurate and efficient binary architectures. *arXiv preprint arXiv:2011.10804*, 2020.

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

[48] Bohan Zhuang, Chunhua Shen, Mingkui Tan, Peng Chen, Lingqiao Liu, and Ian Reid. Structured binary neural networks for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, page 413–422, 2019.

A 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 four experiments in Section 4.5 of our paper. In total, all four experiments (BaseNet k=1, BaseNet k=4, BoolNet k=4, BoolNet* k=4) were trained on 4 GPUs and thus required 276, 252, 204, and 156 GPU hours respectively, in total: 888 GPU hours.

For our ablation studies and our intermediate, initial, or discarded experiments, which were not presented in the paper, we can only provide an estimation of the amount of GPU hours, since we did not have exact measurements in place at the start of this work. We have recorded more than 4300 GPU hours for these experimental results, but estimate that a further 1500-2000 hours were needed in the initial experiments, before we started measuring the runtime.

A.1 Training Details and Further Experimental Results

The training strategy is mostly based on [3]. More specifically, we use the RAdam optimizer [28] with a learning rate of 0.002 without weight decay, use the *cosine learning rate decay* [15], 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. During validation we resize the images to 256×256, and then crop the center with a size of 224×224. Our implementation is based on PyTorch [37], and the code can be online[3]. The implementations of many previous works can not be sped up with `XNOR` and `popcount` (also observed by [14]), 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.

A.1.1 Progressive Weight Binarization vs. Two-Stage Training

We have introduced the progressive weight binarization strategy in Section 3.4, Equation 8 and discussed the results briefly in Section 4.1. 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 6). We chose σ, so the final λ values are equal, i.e. if t_{max} represents the final epoch, then $\lambda_{t_{\text{max}}} = \lambda'_{t_{\text{max}}} = 10^{-6}$. 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.

A.1.2 Code for Reproducibility

We uploaded our training code and all details needed to reproduce each of our experiments depicted in Section 4.5 to `https://github.com/hpi-xnor/BoolNet`.

A.2 Ablation Study on the Downsample Structure

As described in Section 3.2.3, we modify the 1×1 convolution in the downsampling branch in contrast to many previous works [41, 30, 29, 34]. 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,

[3] `https://github.com/hpi-xnor/BoolNet`

			AvgPool Acc. (%)		MaxPool Acc. (%)		Stride=2 Acc. (%)	
k	Bits	Groups	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
1	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
8	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.

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 32-bit most 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. wow !

A.3 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 has 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

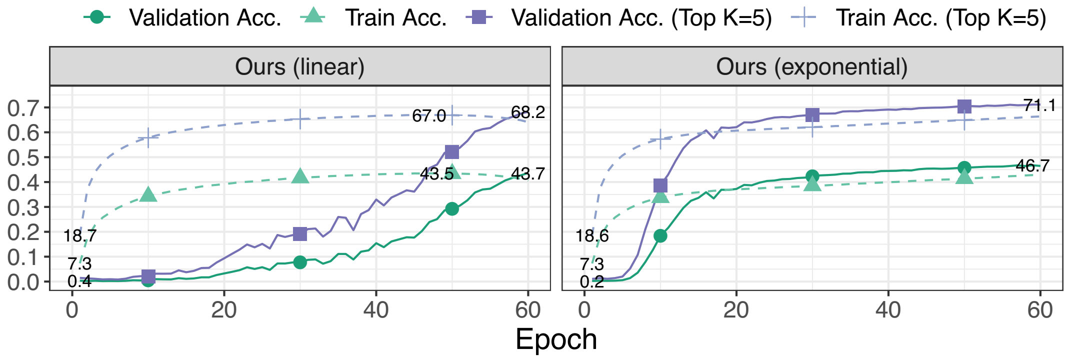
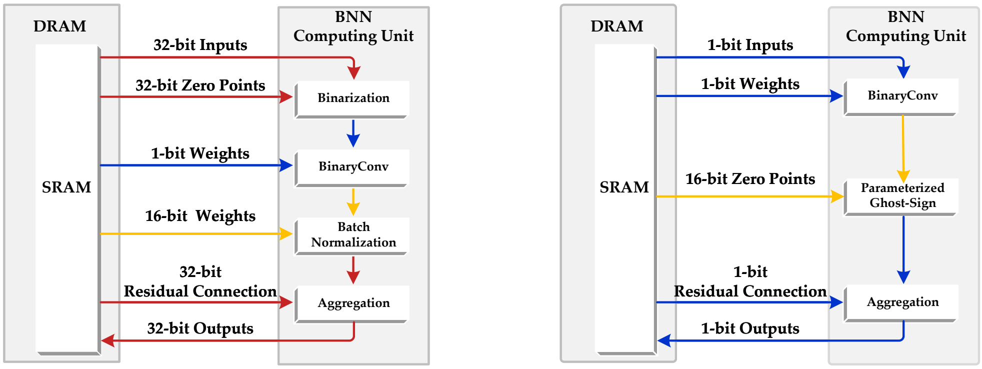


Figure 6: The training and validation accuracy curves of our proposed *Progressive Weight Binarization*. An exponential increase of the slope leads to much better results, than a linear increase.

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. BoolNet* uses dilation instead of stride before the last stage, and thus needs more memory to store the features. However BoolNet* still requires less memory than a regular BNN in the fourth stage.

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)	BoolNet*(k=4)	Regular BNN
Weights	589,824	589,824	589,824	2,359,296	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	100,352·4 = 401,408	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	2·100,352·4 = 802,816	2·25,088·32 = 1,605,632
Total	**740,352**	**1,191,936**	**3,851,264**	**2,434,560**	**2,660,352**	**3,563,520**	**3,990,016**



(a) BiReal Net Data Flow on Hardware

(b) BoolNet Data Flow on Hardware

Figure 7: Hardware data flow comparison between BiReal Net and BoolNet.

access on energy consumption. The details of simulation and energy estimation are introduced as follow.

Overall 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, 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. We write RTL code to realize the above design, use Design Compiler software to synthesize with a TSMC 65nm process, and simulate at 1GHz clock frequency. The software can provide the hierarchical circuit area and power of computing units, including static power (Ps) and dynamic power (Pd). For each layer of the network, we know the calculation amount (A) of each operation. According to the circuit parallelism (Pa), we can calculate the required number of cycles (Cn = A / Pa), and then calculate the energy consumption according to the frequency and power (Ec = Cn × (Ps + Pd) / 10^-9). For the operations with less calculation cycles, the energy consumption waiting for other units is estimated by static power (Es = (Cnmax - Cn) × Ps / 10^-9).

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 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.