FracBNN: Accurate and FPGA-Efficient Binary Neural Networks with Fractional Activations

Yichi Zhang[1], Junhao Pan[2], Xinheng Liu[2], Hongzheng Chen[3,1], Deming Chen[2], Zhiru Zhang[1]
[1]Cornell University [2]University of Illinois Urbana-Champaign [3]Sun Yat-sen University
{yz2499,zhiruz}@cornell.edu

ABSTRACT

Binary neural networks (BNNs) have 1-bit weights and activations. Such networks are well suited for FPGAs, as their dominant computations are bitwise arithmetic and the memory requirement is also significantly reduced. However, compared to start-of-the-art compact convolutional neural network (CNN) models, BNNs tend to produce a much lower accuracy on realistic datasets such as ImageNet. In addition, the input layer of BNNs has gradually become a major compute bottleneck, because it is conventionally excluded from binarization to avoid a large accuracy loss.

This work proposes FracBNN, which exploits fractional activations to substantially improve the accuracy of BNNs. Specifically, our approach employs a dual-precision activation scheme to compute features with up to two bits, using an additional sparse binary convolution. We further binarize the input layer using a novel thermometer encoding. Overall, FracBNN preserves the key benefits of conventional BNNs, where all convolutional layers are computed in pure binary MAC operations (BMACs). We design an efficient FPGA-based accelerator for our novel BNN model that supports the fractional activations. To evaluate the performance of FracBNN under a resource-constrained scenario, we implement the entire optimized network architecture on an embedded FPGA (Xilinx Ultra96 v2). Our experiments on ImageNet show that FracBNN achieves an accuracy comparable to MobileNetV2, surpassing the best-known BNN design on FPGAs with an increase of 28.9% in top-1 accuracy and a 2.5× reduction in model size. FracBNN also outperforms a recently introduced BNN model with an increase of 2.4% in top-1 accuracy while using the same model size. On the embedded FPGA device, FracBNN demonstrates the ability of real-time image classification.[1]

KEYWORDS

Deep Learning; Binary Neural Networks; FPGA Acceleration; HLS

ACM Reference Format:
Yichi Zhang, Junhao Pan, Xinheng Liu, Hongzheng CHen, Deming Chen, Zhiru Zhang . 2021. FracBNN: Accurate and FPGA-Efficient Binary Neural Networks with Fractional Activations. In *Proceedings of the 2021 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '21), February 28-March 2, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3431920.3439296

[1]Code is available at: https://doi.org/10.5281/zenodo.4420520

FPGA '21, February 28-March 2, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8218-2/21/02…$15.00
https://doi.org/10.1145/3431920.3439296

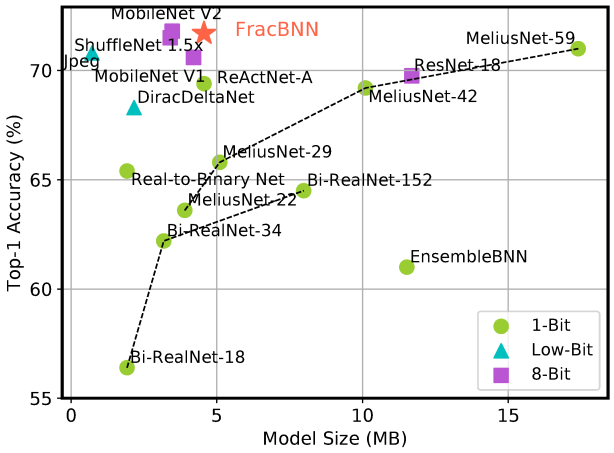


Figure 1: Comparison of ImageNet top-1 accuracy and model size of various compact CNN models.

1 INTRODUCTION

Binary neural network (BNN) is a promising approach to improving the efficiency of deep learning execution, especially for CNNs [21, 31, 37, 40]. With binarized weights and activations, the dominant computations of a BNN model are binary multiply-accumulate (BMAC) operations, which can be implemented in a highly hardware-efficient way using XNORs and population counts (popcnt). The extreme quantization can also reduce the memory requirement for storing the model.

FPGAs are a great match for implementing the BNN models, as the BMAC operations can be mapped and executed on the LUT-based logic fabric in a massively parallel fashion. The reduced memory footprint is also attractive since FPGAs tend to have limited on-chip SRAM capacity. For these reasons, extensive studies have been devoted to the FPGA acceleration of BNNs. Umuroglu et al. [45] and Zhao et al. [57] are among the first to implement an off-the-shelf binarized VGG network on CIFAR-10 [21] using high-level synthesis (HLS). Later, several hardware-friendly BNN models are proposed to make inference more efficient [11, 15, 28, 37, 50]. For example, [15] improves the latency of the BNN CIFAR-10 accelerator to 1.9ms per inference on a Xilinx Zynq device; FP-BNN [28] implements a binarized AlexNet [26] for the ImageNet dataset and delivers a latency of 1.16ms on an Intel Stratix V FPGA.

While BNNs provide obvious benefits in hardware implementation, it is facing two major challenges that are detailed as follows.

BNNs produce a low accuracy on realistic datasets – Hubara et al. [21] pioneered the recent advances in BNNs. This work shows competitive accuracy on small datasets such as MNIST and CIFAR-10. Unfortunately, a binarized AlexNet on ImageNet only achieves a top-1 accuracy of 36.1%, which is more than 20% lower (in absolute difference) than the original full-precision model. The state-of-the-art ImageNet BNN accelerator on FPGAs is based on the same model [28]. Not surprisingly, its accuracy remains very low at 42.9%. Most recently, ReActNet [30] modifies the MobileNet V1 architecture [20] and dramatically increases the accuracy to 69.4% through activation shifting and reshaping. However, this model has as many as 29.3 million parameters (29.3 million bits). In contrast, compact CNN models such as MobileNet V2 [41] can achieve an accuracy of 72% with 3.4 million parameters (27.2 million bits).

The first convolutional layer is not binarized – Existing BNN models commonly use floating-point weights and activations in the input layer to avoid a large degradation in accuracy [2, 30, 31, 40, 51]. The first layer copes with three input channels, thus involving fewer floating-point MAC operations compared to other layers in a conventional CNN. On embedded FPGA devices, however, it is difficult to exploit high parallelism to compute the floating-point input layer due to limited DSP resources. Moreover, a dedicated floating-point convolution engine must be instantiated to execute the input layer, which is not resource-efficient since this engine cannot be reused by other layers. Some prior efforts have attempted to quantize the input layer using fixed-point types [21, 57]. Unfortunately, these techniques typically incur a nontrivial accuracy loss, especially on realistic datasets such as ImageNet.

To overcome the aforementioned challenges, we propose *FracBNN*, an efficient and accurate binary neural network with fractional activations. All convolutional layers in FracBNN are computed in pure BMACs (input layer included). We first construct a baseline BNN model motivated by ReActNet [30]. To improve the accuracy, we compute an extra sparse binary convolutional layer to update a fraction of the features using two bits, thus exploiting fractional activations. As shown in Figure 1, FracBNN outperforms state-of-the-art BNNs and low-bitwidth networks by a large margin. In particular, it achieves MobileNetV2-level accuracy with a competitive model size. We further design an efficient FPGA-based BNN accelerator that supports fractional activations. We implement the entire FracBNN accelerator using HLS, and accelerate the inference on an embedded FPGA (Xilinx Ultra96 v2). FracBNN demonstrates the ability of real-time image classification by achieving a frame rate of 48.1 fps.

Our main technical contributions are as follows:

- We propose FracBNN, an accurate and efficient BNN architecture with fractional activations, where all convolutional layers are computed in BMACs. On ImageNet, FracBNN outperforms the best-known FPGA-targeted BNN by 28.9% and the state-of-the-art ReActNet model by 2.4% in top-1 accuracy. For the first time, we show a CNN with pure BMACs can achieve the same level of accuracy with MobileNet V2.
- In an end-to-end trainable BNN, we propose to use thermometer encoding to preprocess the images and binarize the input layer. We show that thermometer encoding helps with preserving the feature similarity, thus incurring minimal accuracy degradation.
- We design a novel FPGA-based BNN accelerator that supports fractional activations. We implement our design in HLS and demonstrate real-time performance for inference on an embedded FPGA. In terms of frame rate, our FPGA implementation outperforms the most accurate BNN accelerator for CIFAR-10 [57] and a state-of-the-art 4-bit CNN accelerator for ImageNet [51].

2 BNN PRELIMINARIES

In this section, we first describe conventional BNN models used for FPGA implementation. We then introduce a recently proposed BNN model that has achieved a dramatic improvement in accuracy.

2.1 Conventional BNN Models

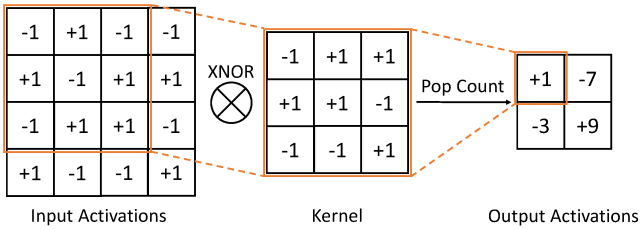


Figure 2: Convolution in a BNN model.

In BNNs, we define a binary convolutional layer as

$$\mathbf{O} = \mathbf{W}^b * \mathbf{X}^b$$

$\mathbf{O}$ is the convolution results. $\mathbf{W}^b$ is the kernel weight tensor, and $\mathbf{X}^b$ is the activation tensor. Both $\mathbf{W}^b$ and $\mathbf{X}^b$ are binarized to either -1 or $+1$ using the sign function. Specifically,

$$w^b = \text{sign}(w^r) = \begin{cases} +1 & w^r \geq 0 \\ -1 & w^r < 0 \end{cases}, \quad x^b = \text{sign}(x^r) = \begin{cases} +1 & x^r \geq 0 \\ -1 & x^r < 0 \end{cases}$$

The superscripts b and r denote binary and real values, respectively.

As shown in Figure 2, due to the binarization of weights and activations, a multiplication and addition (MAC) can be implemented as a bitwise XNOR followed by a popcnt. We can therefore rewrite the binary convolutional layer as

$$\mathbf{O} = \text{popcnt}(\text{XNOR}(\mathbf{W}^b, \mathbf{X}^b))$$

Since XNOR and popcnt operations can easily be mapped and parallelized on the LUT fabric, it is highly efficient to perform the BNN inference on FPGAs.

Note that the BNN models for realistic datasets such as ImageNet implemented by existing FPGA accelerators [28, 57] are typically binarized AlexNet [26] or VGG [42]. The stacked building block consists of a sign function, a binary convolutional layer, and a normalization layer in sequence. Although the binarized layers are efficient on FPGAs, the accuracy of the aforementioned binarized models is low. Moreover, the entire model size is usually more than 10 MB, which exceeds the typical on-chip SRAM capacity of modern embedded FPGAs.

2.2 An Improved BNN Model

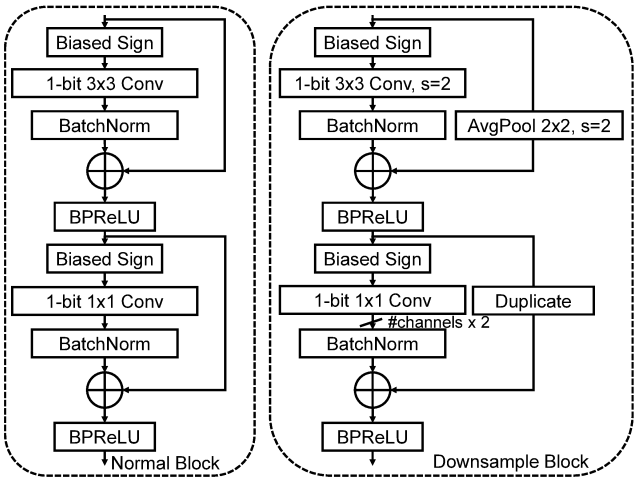


Figure 3: ReActNet building blocks.

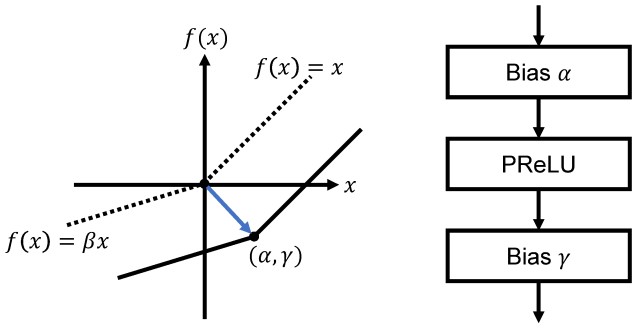


Figure 4: Biased parametric ReLU (BPReLU).

Most recently, a more accurate BNN model named ReActNet [30], is proposed to mitigate the accuracy gap between the binarized model and its full-precision counterpart. The building blocks of ReActNet is shown in Figure 3. ReActNet is based on MobileNet [20] architecture. It achieves a top-1 accuracy of 69.4% on ImageNet dataset using 4.82G BMACs with a model size of 4.6 MB.

The key feature of ReActNet is a biased PReLU (BPReLU) activation function that shifts and reshapes the feature maps between two convolutional layers [18]. This substantially improves the model accuracy. As shown in Figure 4, BPReLU translates the PReLU function to a new origin point (α, γ). It is implemented as a PReLU function sandwiched by two learnable channelwise biases α and γ. Based on the same idea, ReActNet introduces a learnable bias to the sign function to learn the binarization threshold through optimization. Similar to Bi-RealNet [31], ReActNet also adds a full-precision shortcut connection to each convolutional layer in the model. In the downsample layer, the average pooling layer and the channel duplication ensure the shortcut matches the spatial and channel dimensions of the residual. ReActNet uses full instead of depthwise convolutional layers since they increase the capacity of the binarized model. The ReLU activation functions in the original MobileNet are all removed. The limitation of ReActNet is that the input layer is floating-point. Moreover, its accuracy remains low compared to compact networks such as MobileNetV2 which has 72% top-1 accuracy and 3.4 million parameters.

3 THE FRACTIONAL BNN MODEL

In this section, we will present our fractional BNN model. We first describe how we improve the building block of ReActNet to achieve a higher accuracy. We then propose a novel method of binarizing the input layer with minimal accuracy degradation. Finally, we introduce the fractional convolutional layer to further improve model accuracy. FracBNN preserves the key hardware benefits of conventional BNNs. Meanwhile, it achieves a top-1 accuracy of 71.8% on ImageNet, which rivals that of 8-bit MobileNetV2-level with a slightly larger model size.

3.1 New Building Blocks

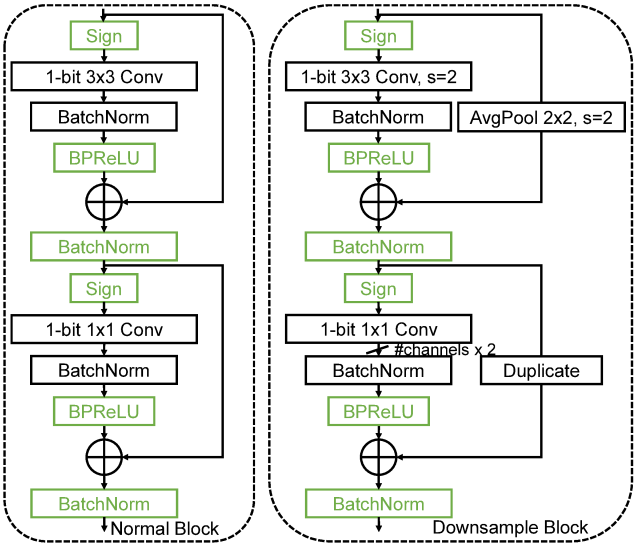


Figure 5: Our model building blocks – Differences from ReActNet are highlighted.

The building block used in our baseline BNN model is illustrated in Figure 5. Different from ReActNet, we move the BPReLUs before the shortcut connections, given that previous works have pointed out that activation functions residing before the shortcuts tend to perform better [6, 33]. We also add a BatchNorm layer [22] after each shortcut connection. The affine transformation of the BatchNorm layer serves as learning a new distribution for both branches in the next block such that the number of positive and negative values in the activations are more balanced. This shift in distribution is shown to be crucial for feature learning in binary convolutional layers in ReActNet. We generalize it to the shortcut connections as well. Since BPReLU and BatchNorm layers only contain channelwise parameters, their impact on the total number of model parameters is negligible. We further find that a learnable threshold is unnecessary for the sign function after adding the BatchNorm layer since it is already included in the bias term.

Table 1: Comparison with SOTA ResNet-20 BNN on CIFAR-10 dataset – The first and last layers are in floating-point.

Model	Method	Precision (W/A)	Acc. %
ResNet-20	DoReFa [58]	1/1	79.3
	DSQ [13]	1/1	84.1
	IR-Net [38]	1/1	86.5
	ReActNet [30]	1/1	85.8
	Ours	1/1	**87.2**

On the CIFAR-10 dataset, we observe that the ResNet-20 BNN model with our proposed building block outperforms other state-of-the-art binarized ResNet-20 variants. We modify the popular ResNet-20 model using the top half of the proposed block in Figure 5 that contains a 3×3 convolutional layer. Table 1 shows the comparison against other methods. DoReFa [58], DSQ [13], and IR-Net [38] explore different backward approximations for the Sign function. As the result shows, the binarized ResNet-20 model constructed with our proposed building blocks achieves the highest accuracy (87.2%). The nontrivial accuracy improvement over the ReActNet structure is gained by moving the activation function right after the convolutional layer and shifting the distribution of both the convolutional branch and the shortcut.

Table 2: Accuracy on ImageNet of the corresponding real-valued models before binarization.

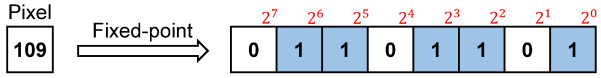
Model	Params ($\times 10^6$)	MACs ($\times 10^9$)	Acc. %
Full Conv MobileNet [20]	29.3	4.8	71.7
Real-Valued ReActNet [30]	29.3	4.8	72.4
Ours	29.3	4.8	**75.6**

For ImageNet dataset, we replace the depthwise separable convolutional layer in the MobileNet architecture with our proposed building block shown in Figure 5.

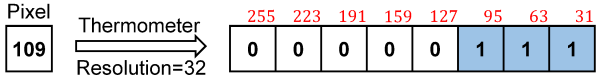
To estimate the potential of the binarized model, we first synchronize the forward and backward behaviors of the sign function to be the straight-through estimator, and train the corresponding real-valued model on ImageNet. The results are in Table 2. All three models have the same number of parameters and MACs. We observe that our approach is 3.2% more accurate than the real-valued ReActNet. Compared with the base full convolution MobileNet, the accuracy increment is 3.9%, even higher than that with ReActNet. The dramatic improvement of our model on ImageNet indicates the efficacy of the pre-shortcut BPReLU activation function and the balance of both branches in a convolutional block.

3.2 Binary Input Layer

A binary input layer can reduce the resource consumption of an FPGA accelerator, since a separate floating- or fixed-point convolution engine is no longer required. The challenge of binarizing both weights and activations in the input layer is the lack of input channels. Anderson et al. [1] show that high-dimensional binary vectors can approximately preserve the dot products in the continuous space. Given the 3×3 kernels, the dimensionality of dot products in the input layer depends on the number of input channels. Three RGB channels are insufficient for binarization. It is therefore necessary to split the images into more channels.



(a) Fixed-point encoding – Representing a fixed-point number as an 8-dimensional binary vector.



(b) Thermometer encoding – The ratio of height of '1's in an encoded binary vector reflects the magnitude of a pixel.

Figure 6: Encoding input images.

Directly using the fixed-point representation of a pixel incurs a large loss. As shown in Figure 6a, a natural way of enriching the channels is to treat each pixel as an 8-dimensional binary vector since pixels are 8-bit fixed-point numbers. Nevertheless, each binary digit has its own associated weight, labeled at the top right corner of each bit. By the time a pixel is converted to a binary vector, the weight information is lost. The magnitude of each bit becomes the same. One may argue that neural network can learn the weights of the binary digits to make an equivalent transformation. However, this is a very challenging task for the BNN training.

In this work, we propose to use thermometer encoding to transform a pixel to a thermometer vector. Previous work has used thermometer encoding to resist adversarial attacks to neural networks [5]. There is also a study [15] that binarizes the input images but the dimension of the encoded vector must be a power of two. Here we use thermometer encoding to binarize the input layer in an end-to-end trainable BNN, and our method supports a flexible vector length. Given a pixel intensity p, $i \in \{1, \ldots, L\}$ is the index of its thermometer vector $TV \in \{0, 1\}^L$, then TV is defined as

$$TV_i = \begin{cases} 0 & 1 \le i \le L - p \\ 1 & L - p < i \le L \end{cases}$$

Namely, the number of 1s in TV is exactly equal to p. The integer L is the dimensionality of TV. In this case $L = 255$. An input image with RGB channels is now converted to 765 (255*3) binary channels. The dimensionality of the dot products hence increases, and there is no associated weights on each bit.

To provide a flexibility on the dimensionality of TV, we further introduce a hyperparameter, resolution R. As depicted in Figure 6b where $R = 32$, each '1' in the encoded thermometer vector represents an intensity of 32. An intensity less than 16 (i.e., $0.5 * R$) will be rounded to '0'. Therefore, $p = 109$ is converted to a binary vector with three '1's. Formally, the new thermometer vector $\tilde{TV}$ is defined as:

$$\tilde{TV}_i = \begin{cases} 0 & 1 \le i \le \lceil \frac{255}{R} \rceil - \lfloor \frac{p}{R} \rceil \\ 1 & \lceil \frac{255}{R} \rceil - \lfloor \frac{p}{R} \rceil < i \le \lceil \frac{255}{R} \rceil \end{cases}$$

where $\lfloor \rceil$ is the round operation, and $L = \lceil \frac{255}{R} \rceil$. Throughout the experiments in this work, we select $R = 8$. Hence, each input channel is expanded to 32 binary channels.

Finally, we transform the thermometer encoded input images to the {-1, +1} bipolar representation to keep consistent with other

activations in the network. Namely, we replace all 0s with −1s in the thermometer vectors. The weights in the input layer are binarized using the regular sign function.

Table 3: Results of binarizing the input layer using thermometer encoding on CIFAR-10 – ResNet-20 BNN has 0.27 million parameters and 40.9 million BMACs.

Model	Method	Acc. (%)	Δ (%)
ResNet-20 BNN	Base	87.2	0.0
	DBID [10]	78.9	-8.3
	BIL (K=256) [10]	83.7	-3.5
	Thermometer (R=8)	87.2	**0.0**

The evaluation of binarizing the input layer is in Table 3. Prior works [10] attempted direct unpacking of the 8-bit fixed-point input data, dubbed as *DBID*, and adding an additional binary pointwise convolutional layer between the unpacked input data and the first layer to increase the number of channels, dubbed as *BIL*. We implement and compare our proposed method against these techniques on the ResNet-20 BNN introduced in Section 3.1.

Our binarized ResNet-20 model has an accuracy of 87.2%. On such a lightweight model, directly unpacking the 8-bit fixed-point data and binarizing the input layer leads to an 8.3% accuracy degradation. This shows that the hidden associated weight information is critical to feature learning. On the basis of that, adding an extra pointwise convolutional layer expands the number of channels from 24 to 256 (K=256), thus increasing the model capacity. This does not address the fundamental problem of losing the associated weights, and still results in a 3.5% degradation. Our proposed method of using thermometer encoding works very well in this case. It preserves the model accuracy after binarizing the first layer. In the meantime, it has 2.7× less BMACs in the input layer compared to BIL since our approach only requires 96 channels. Hence a binarized input layer with thermometer encoding can enjoy a reduced latency and FPGA resources without sacrificing the accuracy.

To understand how an increment in input channels helps with the binarization, we analyze the correlation of the dot products before and after binarization in the input layer. In Figure 7, for both CIFAR-10 and ImageNet models, we plot the 2D histogram of the dot products of the activations and binarized weights (vertical axis) and the dot products of the activations and floating-point weights (horizontal axis) in the input layer. As shown in the first column, the correlation is weak if the inputs are RGB images and have three channels. This is consistent with the observation by Anderson et al. [1]. While using a 96-channel thermometer encoding in the second column, we see that the pre- and post-binarization dot products are highly correlated. This means that thermometer encoding preserves the feature similarity after binarizing the input layer, thus achieving minimal accuracy degradation.

3.3 Fractional Convolution

Low-precision quantized networks usually suffer from accuracy degradation compared to their full-precision counterparts [9]. Prior work proposes an end-to-end trainable technique *precision gating* (PG) [55] that dynamically updates important features to high precision to improve the model accuracy. Binarization is an extreme case

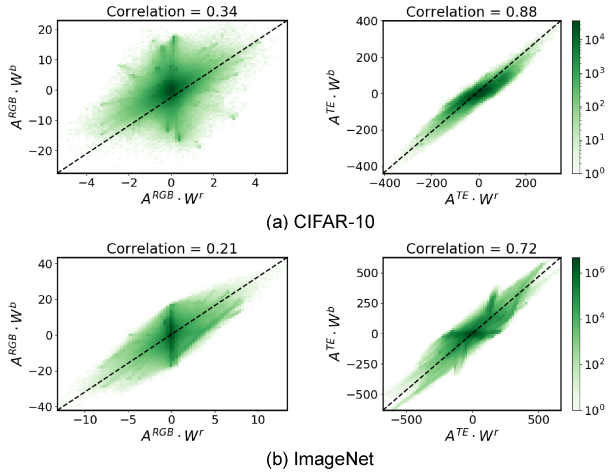


Figure 7: Pearson correlation between dot products before and after binarizing the weights – Dot products in the left column use the RGB images A^{RGB}. Dot products in the right column use the thermometer encoded binary inputs A^{TE}.

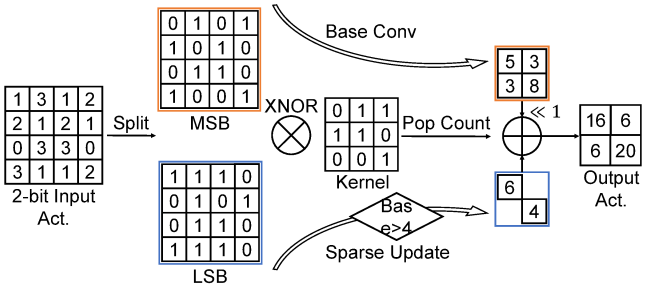


Figure 8: Improving BNN by computing an additional sparse binary convolutional layer.

of quantization. Hence, our fractional convolutional layer (FracConv) adopts PG in the binarized network, except for the input layer and the classifier, to further increase the accuracy. During inference, we dynamically identify important features and compute an additional BMAC to update their popcnt results.

The specific execution flow of FracConv is shown in Figure 8. Instead of binarizing the activations, we store the 2-bit activations in memory. The weight kernels are 1-bit as usual. Each convolutional layer now consists of a base phase and an update phase. The base phase is a regular binary convolutional layer where the inputs are the 1-bit weights and the most-significant bit (MSB) of the activations. During the computation, we maintain a flag bit for each output feature that indicates whether its popcnt result is larger than a learnable threshold Δ. If the flag bit is 1, the output feature is considered to contribute more to the model predictions. It will then be updated by an additional BMAC in the update phase that takes the least-significant bit (LSB) of the activations and the same 1-bit kernels as inputs. Formally, we write the outputs of a FracConv as:

$$\mathbf{O} = \begin{cases} \mathbf{O}_{MSB} << 1 & \mathbf{O}_{MSB} \leq \Delta \\ (\mathbf{O}_{MSB} << 1) + \mathbf{O}_{LSB} & \mathbf{O}_{MSB} > \Delta \end{cases}$$

where

$$\mathbf{O}_{MSB} = \text{popcnt}(\text{XNOR}(\mathbf{W}^b, \mathbf{X}^b_{MSB}))$$

$$\mathbf{O}_{LSB} = \text{popcnt}(\text{XNOR}(\mathbf{W}^b, \mathbf{X}^b_{LSB}))$$

and << is the left shift operation. The learnable threshold Δ is channelwise, and can be viewed as scoring the importance of an output feature based on its popcnt result in the base phase.

Although FracConv incurs extra computation, it still enjoys the benefit of the efficient kernels in BNNs. All of the MACs in the convolutional layers in our model are binary. Moreover, the update phase in FracConv is sparse, thus additional compute effort is not significant. While the activations are quantized to two bits, the memory footprint of FracBNN is similar to a regular BNN since the weights remain binary. Also note that the popcnt accumulations produce multi-bit integers regardless. Hence the increased size of the activation/feature buffer in the update phase is small compared to the weight storage.

We integrate the binary input layer and fractional convolution into our base BNN model introduced in Section 3.1 and construct FracBNN. We then evaluate FracBNN on the ImageNet dataset. The results are in Table 4. We measure that the sparsity in the update binary convolutional layer is 56%. Hence, the equivalent precision of activations in FracBNN is calculated by $1b + (1 - 56\%)b \approx 1.4b$. The baselines include state-of-the-art BNNs, low-precision networks from FPGA accelerator designs, as well as some popular full-precision compact models. To estimate the size of the full-precision models, we assume that they use 8-bit weights and activations since prior studies have shown that 8-bit quantization usually incurs a small accuracy loss [9, 29, 56].

From the ImageNet results we have several key observations:

The accuracy of FracBNN significantly outperforms prior BNN and low-precision FPGA accelerators. FracBNN is 28.9% higher in top-1 accuracy than FP-BNN [28], an FPGA BNN accelerator which implements the binarized AlexNet. Among low-precision networks, DiracDeltaNet constructed in Synetgy [51] is based on ShuffleNetV2 [32]. It replaces the compute-intensive 3×3 convolutional layer by a shift and a pointwise convolutional layer. This results in a higher efficiency for FPGA implementation, but at the expense of reduced model capacity. The JPEGCompress [34] uses a model that is very similar to Synetgy, but is deeper. On the ImageNet dataset, the top-1 accuracy of FracBNN is 3.5% and 1% higher than Synetgy and JPEGCompress, respectively.

The accuracy of FracBNN surpasses SoTA BNNs by a large margin. At a similar model size, FracBNN outperforms MeliusNet-29 [2] and ReActNet-A [30] by 6% and 2.4% in top-1 accuracy, respectively. Even with a 2.2× smaller model size, FracBNN is 2.6% more accurate than MeliusNet-42. Compared to BNN Ensemble [59] that aggregates six binarized ResNet-18 models, FracBNN is 2.5× smaller in size but 10.8% higher in accuracy.

FracBNN achieves MobileNetV2 level accuracy. We compare FracBNN with popular full-precision compact network architectures, and observe that FracBNN achieves the same accuracy level as MobileNetV2. While its convolutional layers are computed in pure BMACs, FracBNN can still reach and even surpass the accuracy of compact CNNs such as MobileNet [20] and ShuffleNet 1.5× [54]. It is also worth noting that FracBNN is more accurate than ResNet-18 (+2%) with a 2.6× smaller model size.

FracBNN has the lowest number of floating-point MACs. With the help of the binary input layer and the fractional convolutional layers, the dominant arithmetic operations in FracBNN are BMACs. Only the classifier is computed in integer MACs, and there are no floating-point MACs. Other models in the baselines have floating-point or 8-bit input layers. Though FracBNN has a considerable number of BMACs, they can be massively parallelized on FPGAs.

We also show the CIFAR-10 results of FracBNN in Table 5. Compared to previous FPGA BNN accelerators [28, 57], FracBNN achieves the highest accuracy, meanwhile with 50× reduction in model size. This enables fully unrolling the network on an embedded FPGA. With the same model size, FracBNN is also 2.6% more accurate than IR-Net [38], the state-of-the-art ResNet-20 BNN variant.

4 FRACBNN ACCELERATOR DESIGN

Our FracBNN consists of the following types of operations:

- 3×3 fractional convolution (stride 1 and 2)
- 1×1 fractional convolution
- 3×3 binary convolution (the input layer)
- Average pooling
- Linear classifier (matrix multiplication)
- Batch normalization and BPReLU activation function
- Residual connection and concatenation

We have designed and implemented an efficient accelerator that supports these operations on an embedded FPGA. In the following, we will describe the hardware engines in detail.

4.1 3×3 and 1×1 Fractional Convolution Engine

One of the key contributions in our network architecture is the fractional convolutional layer introduced in Section 3.3. The fractional convolution scheme improves the accuracy from a single binary convolution with a small resource overhead.

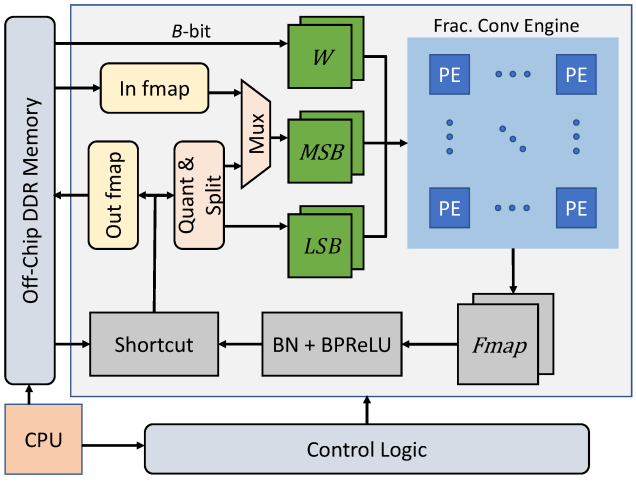


Figure 9: FracBNN accelerator architecture.

Figure 9 illustrates the overall architecture of the accelerator. The entire life cycle of a fractional convolution starts with fetching

Table 4: Comparison of FracBNN with other efficient models on ImageNet – The effective bitwidth of activations in FracBNN is 1.44 bits since the average sparsity in fractional convolution is 56%. IMAC denotes integer MACs. FPMAC denotes floating-point MACs.

Network	Precision (W/A)	Model Size (MB)	IMAC ($\times 10^8$)	BMAC ($\times 10^9$)	FPMAC ($\times 10^8$)	Top-1 (%)	Top-5 (%)
Bi-RealNet-34 [31]	1/1	3.18	0	3.53	1.39	62.2	83.9
MeliusNet-29 [2]	1/1	5.10	0	5.47	1.29	65.8	86.2
MeliusNet-42 [2]	1/1	10.1	0	9.69	1.74	69.2	88.3
Real-to-Binary Net [33]	1/1	1.92	0	1.68	1.56	65.4	86.2
ReActNet-A [30]	1/1	4.56	0	4.82	0.12	69.4	-
BNN Ensemble [59]	1/1	11.52	0	10.10	8.34	61.0	-
FP-BNN [28]	1/1	11.4	1.10	1.03	0	42.9	66.8
JPEGCompress [34]	1/8	0.72	6.21	0	0	70.8	90.1
Synetgy [51]	4/4	2.16	3.30	0	0.09	68.3	88.1
ResNet-18 [17]	8/8	11.69	0	0	18.2	69.8	89.1
MobileNet [20]	8/8	4.20	0	0	5.69	70.6	-
MobileNet V2 [41]	8/8	3.47	0	0	3.00	71.8	91.0
ShuffleNet 1.5× [54]	8/8	3.40	0	0	2.92	71.5	90.2
FracBNN	1/1.4	4.56	0.01	7.30	0	**71.8**	90.1

Table 5: Comparison of FracBNN with other BNNs on CIFAR-10 – The effective precision of activations in FracBNN is 1.4 bits since the average sparsity in fractional convolution is 60%.

Model	Model Size (MB)	BMAC ($\times 10^6$)	IMAC ($\times 10^6$)	Top-1 (%)
IR-Net [38]	0.03	40.5	0.4	86.5
FP-BNN [28]	1.67	58.1	3.6	86.3
BNN [57]	1.67	58.2	3.5	88.9
FracBNN	**0.03**	71.5	0	**89.1**

the 2-bit feature maps from the on-chip Block RAM and the 1-bit weights from the off-chip DDR memory. We manage to store the low-precision feature maps on the Block RAM for faster loading.

After obtaining the feature maps, we first split each 2-bit feature into MSB and LSB. We then pack bits along the channel dimension into B-bit arbitrary precision integers for concurrent access. As we acquire weights from the DDR, we pack them into B-bit vectors to align the precision with the feature maps. In order to balance between parallelism and resource utilization, we select $B = 64$ for the CIFAR-10 design and $B = 32$ for the ImageNet design. The packed weights and feature maps are fed to the convolution engine. Meanwhile, auxiliary parameters, including thresholds, and weights in the BatchNorm and activation functions are fetched from the DDR. We also load the fixed-point shortcuts from the DDR to prepare for the residual connection summation.

The pseudo code of the fractional convolution engine is described in Algorithm 1. As we introduced in Section 3.3, the fractional convolution consists of a base phase and an update phase.

In the base phase (line 5 to 8), the MSB feature maps are convolved with the weights. The computation is the same as a binary convolution. We first compute bitwise XNOR on the B-bit features and weights, and then perform a popcnt on the outcome as the final result. We find that the straight-forward implementation for popcnt of directly adding each bit is sufficiently resource efficient. The combined XNOR and popcnt operations are able to be finished in one cycle. As completing the base path, the output feature maps of the MSB convolution are stored in an on-chip buffer, and we proceed to the update phase.

In the update phase (line 9 to 18), a conditional binary convolution is executed based on the per channel gating thresholds. As shown in the algorithm, an LSB output feature will be computed only when its corresponding MSB output feature exceeds the threshold. The fractional convolution promotes to a binary convolution if the sparsity is 100%; on the other hand, it degenerates to a 2-bit convolution if the sparsity is 0%. Otherwise it is in between. If the sparsity is low enough, it then becomes more straightforward to directly parallelize the MSB and LSB convolutions to a 2-bit convolution at the cost of more hardware resources.

The binary convolutions are pipelined on the spatial dimension and parallelized on the channel dimension. Within the resource constraint, we fully unroll the computation in a 3×3 window to maximize the parallelism. The point-wise fractional convolution is essentially the same as the 3×3 convolution other than the size of the window. Upon finishing the convolution operations, we perform BatchNorm and BPReLU activation on the results and produces the output of a block. Finally, we organize the outputs into a buffer and transfer them to DDR.

4.2 3×3 Binary Convolution Engine

The pure binary convolutional layer takes place only at the thermometer encoded inputs. We reuse the 3×3 convolution engine in the fractional convolutional layer to handle this occasion. Since the input layer is excluded from fractional activations, we only execute the base phase. For ImageNet, we pack 32 bits together along the input channel and feed into the convolution engine. It outputs 32 channels as well. Since the input has 96 channels after the thermometer encoding, we iterate three times for each spatial position and accumulate the results for the output of the layer.

4.3 Average Pooling, Classifier, and Other Operations

In addition to the accelerators for convolution operations, which account for the majority of the computations in the network, we also

Algorithm 1 Fractional Convolution

```
function FracConv(fmap, weights, threshold, output)
    W ← Width
    H ← Height
    fmap_msb ← Split(fmap, msb)
    fmap_lsb ← Split(fmap, lsb)
    for ch_out in 0, 1, ... C_out − 1 do
        for ch_in in 0, 1, ... C_in − 1 do
            w ← LoadWeights(ch_out, ch_in)
            output[ch_out] + = BinaryConv(fmap_msb[ch_in], w)
        end for
        for ch_in in 0, 1, ... C_in − 1 do
            w ← LoadWeights(ch_out, ch_in)
            for pixel in 0, 1, ..., H ∗ W − 1 do
                if output[ch_out][pixel] > threshold[ch_out] then
                    act ← LoadWindow_lsb(fmap_lsb, ch_in, H, W)
                    output[ch_out][pixel] +=
popcnt(XNOR(act, w))
                end if
            end for
        end for
    end for
end function
```

implement hardware accelerators for less frequent operations such as average pooling and matrix multiplications. An average pooling layer sums up features in the spatial dimension, and is divided by the number of features summed together. The challenge of designing this kernel is determining the amount of parallelism and the trade-off between resource and latency. We use an adder tree to serve as a sum engine to compute the sum in one dimension in a pooling tile in one cycle and then pipeline the addition operation for the other dimension to achieve the optimal concurrency. Even though it is possible to maximize the parallelism by accessing all the elements in a tile at once and compute the sum with the fewest number of cycles, it requires partitioning the array in both dimensions. The cost of resource utilization may not be worth the performance gain. Our implementation achieves comparable performance with economic resource consumption. The matrix multiplication unit serves as the linear classifier at the final state of the network. It computes the class dimension in parallel. For further optimization, it can be assigned to CPU and executed in parallel with the accelerator. This compact accelerator design is the balanced choice between performance and resource usage. Other miscellaneous operations such as BatchNorm, BPReLU activation, and channel concatenation are fused seamlessly with adjacent operations.

5 EVALUATION

In this section, we first describe the experiment setup, and then present our results on FPGAs.

5.1 Modeling Training Setup

We evaluate FracNN on both CIFAR-10 [25] and ILSVRC12 ImageNet [23] classification datasets. We augment the input images using random horizontal flip and random crop. Color jitter is used only for ImageNet. We follow the two-step training strategy as described in Real-to-Binary Net [33]. In the first step, the activations are binarized but the weights are floating-point. The weight decay is 1e-5. In the second step, weights are binarized and initialized from the first step. Activations are still binary. The weight decay is zero. To train the FracBNN, the first two steps are the same except that the activations are quantized to 2 bits. We add a third step that initializes the weights from the second step, and applies the fractional convolutional layer. The first and the last layer are excluded. We use cross-entropy loss during the training on CIFAR-10. For ImageNet, we calculate the KL divergence between the softmax output of a teacher model and that of the trained model as the loss function, same as ReActNet [30]. In our experiments the teacher model is a pretrained ResNet-50.

For CIFAR-10, we train the model for 300 epochs in each training step with a batch size of 128. The initial learning rate is 1e-3, and decays linearly to 0 in each epoch. For ImageNet, we train the model for 120 epochs in each step. The batch size is 256. The initial learning rate is 5e-4 and also decays linearly to 0 in each epoch. We use PyTorch [36] to specify models and training scripts. All training experiments are completed on NVIDIA RTX 2080Ti GPUs.

5.2 FPGA Implementation

We evaluate the performance of the FracBNN accelerator architecture on a Xilinx Ultra96 v2 FPGA board. This board uses the Zynq UltraScale+ MPSoC device (ZU3EG), which contains an embedded ARM CPU. The programmable logic fabric has 71k LUTs, 360 DSPs, and 7.6 Mb BRAMs. Ultra96 v2 is supported by PYNQ, which can import and invoke the accelerator as an overlay in a Python environment. Programmers can feed data and control signals from software via AXI/DMA interfaces to the accelerator. We have implemented accelerators for both CIFAR-10 and ImageNet. The CIFAR-10 accelerator is fully unrolled on the board. Both designs run at a clock frequency of 250MHz. The post-implementation resource utilization is summarized in Table 8.

Table 6 compares our ImageNet accelerators against other previous works. Notably, our ImageNet model provides a significant advantage in the model accuracy. The top-1 accuracy is the best amongst the other comparable network models that are implemented on FPGAs. To map the entire model on the FPGA, it takes 72% LUTs and 93% BRAM usage. Our DSP utilization is allocated mainly for the index calculation, normalization, and activation functions. Due to the limited available resource, we store the intermediate feature maps on the DDR memory upon completing a combination of convolutional, BatchNorm, and BPReLU layers, and fetch them when computing the residual connection. We are able to achieve 48.1 frames per second (FPS) as we test our design in the PYNQ environment. Our hardware logic runs at 16 ms. We allocate double buffers to overlap part of the communication overhead and compute. Our design can perform real-time image classifications, and the attainable frame rate surpasses current state-of-the-art embedded hardware accelerators on the same task. In Table 6, the designs that have significantly higher frame rates either target a server-class FPGA such as Intel Stratix V or have a lower accuracy than ours. It is worth noting that Synetgy [51] take a different

Table 6: Hardware performance of FracBNN on ImageNet at batch size of 1.

	ReBNet [11]	AlexNet [28]	FINN-R [3]	T-DLA [7]	MobileNetV2 [49]	Synetgy [51]	JPEGComp [34]	FracBNN
Device	Virtex VCU108	Stratix-V	Zynq ZU3EG	Zynq 7Z020	Zynq ZU2EG	Zynq ZU3EG	VirtexUS+ XCVU9P	Zynq ZU3EG
FPS	170	862.1	200.0	20.48	205.3	41.1	3321.2	48.1
Top-1 (%)	41.43	42.9	50.3	65.6	68.1	68.3	70.8	**71.8**
Top-5 (%)	-	66.8	-	-	-	88.1	90.1	**90.1**
Bits (W/A)	1/1	1/1	1/2	2/2	8/8	4/4	1/8	1/1.4
F_{max} (MHz)	200	150	220	250	430	250	300	250
Power (W)	-	26.2	10.2	2.58	-	5.5	75	6.1
LUT	537600	230918	36249	37921	31198	51776	274795	50656
BRAM	3456	2210	432	97	145	159	2746	201
DSP	768	384	-	202	212	360	2370	224

Table 7: FracBNN performance on CIFAR-10 (batch size 1).

	ReBNet [11]	BNN [57]	FBNA [15]	FracBNN
Device	Zynq ZC702	Zynq 7Z020	Zynq ZC702	Zynq ZU3EG
FPS	2000	168.4	520.8	**2806.9**
Top-1 (%)	86.98	88.8	88.6	**89.1**
Bits (W/A)	1/1	1/1	1/1	1/1.4
F_{max} (MHz)	200	143	-	250
Power (W)	-	4.7	3.3	4.1

Table 8: Resource utilization of the FracBNN accelerator.

	DSPs	BRAM	LUTs
CIFAR-10	126 (35%)	212 (98.1%)	51444 (72.9%)
ImageNet	224 (62.2%)	201 (93.0%)	50656 (71.8%)

approach in their FPGA implementation — the system consists of independent accelerators, which the CPU invokes them as needed.

Table 7 compares our results against other works that target CIFAR-10. FracBNN once again achieves the best model accuracy. Compared to the BNN accelerator in [57] and FBNA [15], our design achieves the highest frame rate with a better accuracy on a comparable embedded FPGA platform. Since our CIFAR-10 network model is very compact, we are able to unroll the entire network on the FPGA logic to eliminate unnecessary transactions between the logic blocks and the DDR memory. The only data transmissions are the input image and prediction results. Under this setting, we achieve an FPS of 2806.9 with 72.9% of LUT utilization. The frame rate is 1.4× higher than ReBNet [11], which has the highest frame rate among the baselines but with a lower accuracy (by 2.1%).

5.3 Ablation Study

We further validate the efficacy of the binary input layer and the fractional convolutional layer.

We observe that our binary input layer runs significantly faster than a conventional fixed-point input layer with trivial additional resource consumption. Table 9 shows the comparison in resource utilization and latency. As described in Section 4.2, our binary input layer reuses the convolution engine in the fractional convolutional layer. The only resource consumption occurs in loading the image and the weights, therefore remains low. Moreover, the latency of our binary input layer is very low since the convolution engine is highly parallelized when designed for the fractional convolution. In contrast, the conventional input layer with 8-bit inputs and weights requires DSP to compute the the results. The number of DSPs, however, is limited on the target FPGA device. This resource constraint makes it difficult for the 8-bit design to achieve the same parallelism as binary computations, thus resulting in a much higher latency.

Table 9: Comparison between the implementation of our binary input layer and an 8-bit conventional input layer.

Conv.	DSPs	BRAM	LUTs	Latency(ms)
1-bit	3 (0.8%)	2.5 (1.2%)	3603 (5.1%)	2.0
8-bit	287 (79.7%)	34 (15.6%)	22509 (31.9%)	65.9

To evaluate the efficiency of the fractional convolution, we implement and compare the logic part of three networks — each with 1-bit convolutional layers, fractional convolutional layers, and 2-bit convolutional layers, respectively. In the previous sections, we discuss the expectation that a neural network exploiting the fractional convolution should perform slightly worse than a pure binary (1-bit weights and 1-bit activation) model, but have significant improvements over a conventional 2-bit activations and 1-bit weights model. Table 10 shows a side-by-side comparison among the three models. Clearly, the implementation results confirm our expectations on the fractional network. We expend some extra resource to ensure the same concurrency as the 1-bit model to match the performance. The model with fractional convolutional layers has a slightly worse latency compared to the 1-bit network, but it is more than 3x better than a conventional 2-bit convolutional network. The only difference in the models is the precision of the convolution accelerator modules. The reason behind such difference is that the conventional 2-bit convolution accelerator requires more resources than either 1-bit or fractional ones. If we hope to ensure the same concurrency, there is no room to fit a much more expensive 2-bit convolution accelerator. Eventually, we have to tune down the concurrency to fit the network on the hardware, resulting in a higher latency but slightly less resource utilization than the 1-bit and fractional convolution models.

6 RELATED WORK

Binary Neural Networks. The pioneering works on BNN [8, 21] establish the end-to-end training flow for the discrete networks.

Table 10: Comparisons among the 1-bit, fractional, and 2-bit Networks on ImageNet.

Network Bitwidth	DSPs	BRAM	LUTs	Latency (ms)
1-bit	87 (23.7%)	137 (63.4%)	59488 (84.3%)	15.0
2-bit	85 (23.8%)	169 (78.5%)	53419 (75.7%)	61.3
Frac.	224 (62.2%)	201 (93.0%)	50656 (71.8%)	16.3

Courbariaux et al. [21] binarize the weights and activations using the sign function. This incurs nearly no loss in accuracy on small datasets such as MNIST [27], SVHN [35], and CIFAR-10 [25]. While its preliminary result of binarized AlexNet [26] only achieves 36.1% top-1 accuracy on the ImageNet dataset, this work demonstrates the feasibility of BNNs. There have been extensive follow-up efforts to improve the accuracy of BNNs. Most of the attempts are along the line of modifying BNN network architectures [2, 31, 40]. There are also works that explore different training strategies of BNNs [33, 44, 60]. Recently, PReLU is found to be a better activation function for BNNs [6]. With an additional shift on the basis of PReLU, ReActNet [30] binarizes MobileNet [20] and obtains ResNet-18 level accuracy. In addition, [59] explores using an ensemble of multiple BNN models to improve the accuracy, albeit at the cost of higher compute complexity. There is also a study that tailors BNNs for FPGAs by constructing LUTNet, an area-efficient LUT-based neural network [46]. Unlike the aforementioned research, the proposed approach exploits fractional activations to achieve efficient and accurate quantization. Specifically, we leverage our recent work on precision gating (PG) [55], a dynamic dual-precision scheme that updates important features to a high precision at the inference time based on a learnable threshold. We adopt PG in our baseline BNN model motivated by ReActNet, and update a small portion of features to two bits to improve the model accuracy.

Quantization of Input Layer. One visible drawback about current BNNs in terms of hardware design is that the first layer remains full-precision. Hirtzlin et al. [19] propose to use stochastic computing for the binarization of the input images. This method expands the 3 input color channels from images in CIFAR-10 to more than 1500 binary channels. Consequently, it increases the number of parameters and MACs in the input layer by nearly 16×. Dürichen et al. [10] discuss two other options. The first one is using the 8-bit fixed-point representation of a pixel, named DBID. However, the associated weight of each binary digit is lost when converted to a binary vector. In the second option, a pointwise convolutional layer is added between the images and the input layer while using DBID. It does not address the fundamental problem in DBID. Unfortunately, when applied to the VGG-8 model on CIFAR-10, these two methods degrade the accuracy by at least 4.6%. Our method is very different from these techniques as we use thermometer encoding to split the pixels into a binary vector. It incurs minimal or even no accuracy degradation. FBNA [15] expands each pixel to a sum of a binary vector for a pretrained model; but the dimension of that vector must be a power of two. Our proposed method is different as it can encode a pixel to an arbitrary dimension between 1 and 255, and BNNs with it are still end-to-end trainable. In MeliusNet [2], a grouped stem architecture is proposed to reduce the MACs in the input layer by 40%. The technique replaces the 7×7 convolutional layer with three 3×3 group convolutional layers. Though there is MACs reduction, the input layer is still floating-point. Our proposed method is different as we use a binary input layer.

FPGA-Based CNN Accelerators. There have been extensive studies on accelerating low-precision neural networks [37]. Qiu et al. [39] first show convolutional layers are computation-bound while fully-connected layers are memory-bound, and propose a dynamic-precision data quantization method to accelerate CNN. Different hardware architectures are then proposed to address the bottleneck in computation and memory bandwidth [34, 48, 53], and high-level programming frameworks are proposed to help efficient quantization [3, 7, 12, 43, 52]. However, most of these works target AlexNet and VGG, which are much less efficient than more recent CNN models such as ResNet and MobileNet. As another approach, algorithm-hardware co-design method is leveraged to develop hardware-efficient networks with fewer bits in weights and activations [14, 16, 24, 47, 49, 51]. Our work is very different as all of the convolutional layers in the model are computed in pure BMACs.

There is also an active body of research on accelerating BNN models on FPGAs. Zhao et al. [57] make the first attempt implementing a BNN accelerator on FPGA, which introduces a BitSel module and variable-width length buffers to make the network inference efficient. FINN [4, 45] provides a framework for fast BNN inference. ReBNet [11] leverages multi-level residual binarization to improve the accuracy. FBNA [15] binarizes all the network layers but only targets the CIFAR-10 dataset. Liang et al. [28] implement the binarized AlexNet on ImageNet. The top-1 accuracy is 42.9%, which is 13% lower than its full-precision model. Our work uses fractional activations to create an accurate model.

7 CONCLUSIONS

This work proposes FracBNN, which exploits fractional activations to substantially improve the accuracy of BNNs. FracBNN employs a dual-precision activation scheme. Features are computed with up to two bits, using an additional sparse binary convolution. The input layer is also binarized using a novel thermometer encoding. FracBNN preserves the key hardware benefits of conventional BNNs. We design an efficient FPGA-based accelerator for our novel fractional convolution kernel. We also implement the entire optimized network on an embedd FPGA (Xilinx Ultra96 v2). Experiments show that FracBNN achieves a top-1 accuracy comparable to MobileNetV2, surpassing that of the best-known BNN FPGA accelerator and a recently introduced BNN by a large margin. On the embedded FPGA, FracBNN demonstrates the ability of real-time image classification.

ACKNOWLEDGEMENTS

This work was supported in part by the Semiconductor Research Corporation (SRC) and DARPA, NSF Award #2007832, the Xilinx Center of Excellence and Xilinx Adaptive Compute Clusters (XACC) program at the University of Illinois Urbana-Champaign. One of the Titan Xp GPUs used for this research was donated by the NVIDIA Corporation. We thank Yuwei Hu, Yuan Zhou, Yi-Hsiang Lai, Hanchen Jin, and Ecenur Ustun of the Zhang Research Group at Cornell for their helpful discussions.

REFERENCES

[1] Alexander G. Anderson and Cory P. Berg. The High-Dimensional Geometry of Binary Neural Networks. *Int'l Conf. on Learning Representations (ICLR)*, 2018.

[2] Joseph Bethge, Christian Bartz, Haojin Yang, Ying Chen, and Christoph Meinel. MeliusNet: Can Binary Neural Networks Achieve MobileNet-level Accuracy? *arXiv preprint arXiv:2001.05936*, 2020.

[3] Michaela Blott, Thomas B. Preußer, Nicholas J. Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks. *ACM Trans. Reconfigurable Technol. Syst.*, Dec 2018.

[4] Michaela Blott, Thomas B. Preußer, Nicholas J. Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks. *ACM Trans. Reconfigurable Technol. Syst.*, Dec 2018.

[5] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer Encoding: One Hot Way To Resist Adversarial Examples. *Int'l Conf. on Learning Representations (ICLR)*, 2018.

[6] Adrian Bulat, Georgios Tzimiropoulos, Jean Kossaifi, and Maja Pantic. Improved training of binary networks for human pose estimation and image recognition. *arXiv preprint arXiv:1904.05868*, 2019.

[7] Y. Chen, K. Zhang, C. Gong, C. Hao, X. Zhang, T. Li, and D. Chen. T-DLA: An Open-source Deep Learning Accelerator for Ternarized DNN Models on Embedded FPGA. *IEEE Computer Society Annual Symp. on VLSI (ISVLSI)*, 2019.

[8] Zhiyong Cheng, Daniel Soudry, Zexi Mao, and Zhenzhong Lan. Training Binary Multilayer Neural Networks for Image Classification using Expectation Backpropagation. *arXiv preprint arXiv:1503.03562*, 2015.

[9] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. PACT: Parameterized Clipping Activation for Quantized Neural Networks. *arXiv preprint arXiv:1805.06085*, 2018.

[10] Robert Dürichen, Thomas Rocznik, Oliver Renz, and Christian Peters. Binary Input Layer: Training of CNN models with binary input data. *arXiv preprint arXiv:1812.03410*, 2018.

[11] M. Ghasemzadeh, M. Samragh, and F. Koushanfar. ReBNet: Residual Binarized Neural Network. *IEEE Symp. Field Programmable Custom Computing Machines (FCCM)*, 2018.

[12] C. Gong, Y. Chen, Y. Lu, T. Li, C. Hao, and D. Chen. VecQ: Minimal Loss DNN Model Compression With Vectorized Weight Quantization. *IEEE Trans. on Computers*, 2020.

[13] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable Soft Quantization: Bridging Full-Precision and Low-Bit Neural Networks. *Int'l Conf. on Computer Vision (ICCV)*, Oct 2019.

[14] K. Guo, S. Han, S. Yao, Y. Wang, Y. Xie, and H. Yang. Software-Hardware Codesign for Efficient Neural Network Acceleration. *IEEE Micro*, 2017.

[15] P. Guo, H. Ma, R. Chen, P. Li, S. Xie, and D. Wang. FBNA: A Fully Binarized Neural Network Accelerator. *Int'l Conf. on Field Programmable Logic and Applications (FPL)*, 2018.

[16] Cong Hao, Xiaofan Zhang, Yuhong Li, Sitao Huang, Jinjun Xiong, Kyle Rupnow, Wen-mei Hwu, and Deming Chen. FPGA/DNN Co-Design: An Efficient Design Methodology for IoT Intelligence on the Edge. *Design Automation Conf. (DAC)*, 2019.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *Int'l Conf. on Computer Vision (ICCV)*, 2015.

[19] T. Hirtzlin, B. Penkovsky, M. Bocquet, J. Klein, J. Portal, and D. Querlioz. Stochastic Computing for Hardware Implementation of Binarized Neural Networks. *IEEE Access*, 2019.

[20] 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.

[21] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks. *Conf. on Neural Information Processing Systems (NeurIPS)*, 2016.

[22] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Int'l Conf. on Machine Learning (ICML)*, 2015.

[23] Deng J., Dong W., Socher R., Li L., Li Kai, and Fei-Fei Li. ImageNet: A large-scale hierarchical image database. *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[24] L. Jiao, C. Luo, W. Cao, X. Zhou, and L. Wang. Accelerating low bit-width convolutional neural networks with embedded FPGA. *Int'l Conf. on Field Programmable Logic and Applications (FPL)*, 2017.

[25] Alex Krizhevsky and Geoffrey Hinton. Learning Multiple Layers of Features from Tiny Images. *Tech report*, 2009.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Conf. on Neural Information Processing Systems (NeurIPS)*, 2012.

[27] Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2010.

[28] Shuang Liang, Shouyi Yin, Leibo Liu, Wayne Luk, and Shaojun Wei. FP-BNN: Binarized neural network on FPGA. *Neurocomputing*, 2018.

[29] Darryl D. Lin, Sachin S. Talathi, and V. Sreekanth Annapureddy. Fixed Point Quantization of Deep Convolutional Networks. *Int'l Conf. on Machine Learning (ICML)*, 2016.

[30] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. ReActNet: Towards Precise Binary Neural Network with Generalized Activation Functions. *European Conf. on Computer Vision (ECCV)*, 2020.

[31] 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. *European Conf. on Computer Vision (ECCV)*, Sep 2018.

[32] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. *European Conf. on Computer Vision (ECCV)*, Sep 2018.

[33] Brais Martinez, Jing Yang, Adrian Bulat, and Georgios Tzimiropoulos. Training binary neural networks with real-to-binary convolutions. *Int'l Conf. on Learning Representations (ICLR)*, 2020.

[34] H. Nakahara, Z. Que, and W. Luk. High-Throughput Convolutional Neural Network on an FPGA by Customized JPEG Compression. *IEEE Symp. Field Programmable Custom Computing Machines (FCCM)*, 2020.

[35] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading Digits in Natural Images with Unsupervised Feature Learning. *Conf. on Neural Information Processing Systems (NeurIPS)*, 2011.

[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Conf. on Neural Information Processing Systems (NeurIPS)*, 2019.

[37] Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe. Binary Neural Networks: A Survey. *arXiv preprint arXiv:2004.03333*, 2020.

[38] Haotong Qin, Ruihao Gong, Xianglong Liu, Mingzhu Shen, Ziran Wei, Fengwei Yu, and Jingkuan Song. Forward and Backward Information Retention for Accurate Binary Neural Networks. *arXiv preprint arXiv:1909.10788*, 2019.

[39] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, Yu Wang, and Huazhong Yang. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2016.

[40] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. *European Conf. on Computer Vision (ECCV)*, 2016.

[41] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun 2018.

[42] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *Int'l Conf. on Learning Representations (ICLR)*, 2015.

[43] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. Throughput-Optimized OpenCL-Based FPGA Accelerator for Large-Scale Convolutional Neural Networks. *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2016.

[44] Wei Tang, Gang Hua, and Liang Wang. How to Train a Compact Binary Neural Network with High Accuracy? *AAAI Conf. on Artificial Intelligence (AAAI)*, 2017.

[45] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2017.

[46] E. Wang, J. J. Davis, P. Y. K. Cheung, and G. A. Constantinides. LUTNet: Rethinking Inference in FPGA Soft Logic. *IEEE Symp. Field Programmable Custom Computing Machines (FCCM)*, 2019.

[47] Junsong Wang, Qiuwen Lou, Xiaofan Zhang, Chao Zhu, Yonghua Lin, and Deming Chen. Design Flow of Accelerating Hybrid Extremely Low Bit-Width Neural Network in Embedded FPGA. *Int'l Conf. on Field Programmable Logic and Applications (FPL)*, 2018.

[48] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and J. Cong. Automated systolic array architecture synthesis for high throughput cnn inference on fpgas. *Design Automation Conf. (DAC)*, 2017.

[49] D. Wu, Y. Zhang, X. Jia, L. Tian, T. Li, L. Sui, D. Xie, and Y. Shan. A High-Performance CNN Processor Based on FPGA for MobileNets. *Int'l Conf. on Field Programmable Logic and Applications (FPL)*, 2019.

[50] Li Yang, Zhezhi He, and Deliang Fan. A Fully Onchip Binarized Convolutional Neural Network FPGA Impelmentation with Accurate Inference. *Int'l Symp. on*

Low Power Electronics and Design, 2018.

[51] Yifan Yang, Qijing Huang, Bichen Wu, Tianjun Zhang, Liang Ma, Giulio Gambardella, Michaela Blott, Luciano Lavagno, Kees Vissers, John Wawrzynek, and Kurt Keutzer. Synetgy: Algorithm-Hardware Co-Design for ConvNet Accelerators on Embedded FPGAs. *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2019.

[52] C. Zhang, Zhenman Fang, Peipei Zhou, Peichen Pan, and Jason Cong. Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks. *Int'l Conf. on Computer-Aided Design (ICCAD)*, 2016.

[53] Jialiang Zhang and Jing Li. Improving the Performance of OpenCL-Based FPGA Accelerator for Convolutional Neural Network. *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, 2017.

[54] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun 2018.

[55] Yichi Zhang, Ritchie Zhao, Weizhe Hua, Nayun Xu, G. Edward Suh, and Zhiru Zhang. Precision Gating: Improving Neural Network Efficiency with Dynamic Dual-Precision Activations. *Int'l Conf. on Learning Representations (ICLR)*, 2020.

[56] Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang. Improving Neural Network Quantization without Retraining using Outlier Channel Splitting. *Int'l Conf. on Machine Learning (ICML)*, Jun 2019.

[57] Ritchie Zhao, Weinan Song, Wentao Zhang, Tianwei Xing, Jeng-Hau Lin, Mani Srivastava, Rajesh Gupta, and Zhiru Zhang. Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs. *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, Feb 2017.

[58] Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *CoRR*, 2016.

[59] Shilin Zhu, Xin Dong, and Hao Su. Binary Ensemble Neural Network: More Bits per Network or More Networks per Bit? *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun 2019.

[60] Bohan Zhuang, Chunhua Shen, Mingkui Tan, Lingqiao Liu, and Ian Reid. Towards Effective Low-Bitwidth Convolutional Neural Networks. *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun 2018.