Binarized Convolutional Neural Networks with Separable Filters for Efficient Hardware Acceleration

Jeng-Hau Lin[1], Tianwei Xing[2], Ritchie Zhao[3], Zhiru Zhang[3], Mani Srivastava[2], Zhuowen Tu[1,4] and Rajesh K. Gupta[1]
[1]Department of Computer Science and Engineering, UC San Diego
[2]Department of Electrical Engineering, UC Los Angeles
[3]Department of Electrical and Computer Engineering, Cornell University
[4]Department of Cognition Science, UC San Diego
[1]{jel252,ztu,rgupta}@ucsd.edu, [2]{twxing,mbs}@ucla.edu,
[3]{rz252,zhiruz}@cornell.edu

Abstract

State-of-the-art convolutional neural networks are enormously costly in both compute and memory, demanding massively parallel GPUs for execution. Such networks strain the computational capabilities and energy available to embedded and mobile processing platforms, restricting their use in many important applications. In this paper, we push the boundaries of hardware-effective CNN design by proposing BCNN with Separable Filters (BCNNw/SF), which applies Singular Value Decomposition (SVD) on BCNN kernels to further reduce computational and storage complexity. To enable its implementation, we provide a closed form of the gradient over SVD to calculate the exact gradient with respect to every binarized weight in backward propagation. We verify BCNNw/SF on the MNIST, CIFAR-10, and SVHN datasets, and implement an accelerator for CIFAR-10 on FPGA hardware. Our BCNNw/SF accelerator realizes memory savings of 17% *and execution time reduction of* 31.3% *compared to BCNN with only minor accuracy sacrifices.*

1. Introduction

Albeit the community of neural networks has been prospering for decades, state-of-the-art CNNs still demand significant computing resources (i.e., high-performance GPUs), and are eminently unsuited for resource and power-limited embedded hardware or Internet-of-Things (IoT) platforms [13]. Reasons for high resource needs include the complexity of connections among layers, the sheer number of fixed-point multiplication and accumulation (MAC) operations, and the storage requirements for weights and biases. Even if network training is done off-line, only a few high-end IoT devices can realistically carry out the forward propagation of even a simple CNN for image classification.

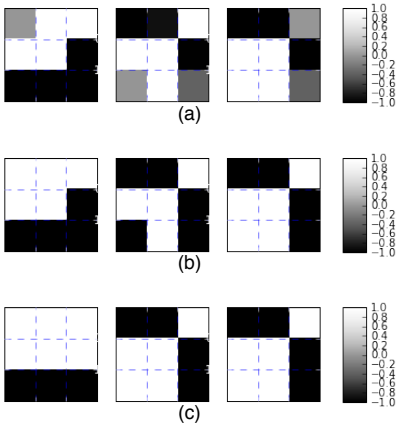

Figure 1. Comparison of filters: (a) original floating point filter; (b) same filter binarized; (c) approximated separable binary filter.

Binarized convolutional neural networks (BCNNs) [6, 3, 18, 9, 13] have been proposed as a more hardware-friendly model with extremely degenerated precision of weights and activations. BCNN replaces floating or fixed-point multiplies with XNOR operations (which can be implemented extremely efficiently on ASIC or FPGA devices) and achieved near state-of-the art accuracy on a number of real-world image datasets at time of publication. Unfortunately, this hardware efficiency is offset by the fact that a BCNN model is typically tens or hundreds times the size of a CNN model of equal accuracy. To make BCNNs practical, an effective way to reduce the model size is required.

In this paper, we introduce Separable Filters (SF) on binarized filters, as shown in Fig. 1(c), to further reduce the

hardware complexity in two aspects:

- SF reduces the number of possible unique d-by-d filters from 2^{d^2} to just 2^{2d-1}, enabling the use of a small look-up table during forward propagation. This directly results in the $\frac{(d-1)^2}{d^2}$ reduction of memory footprint.
- SF replaces each d-by-d 2D convolution with two d-length 1D convolutions, which reduces the number of MAC operations by $d/2$. This translates to either speedup or the same throughput with fewer resources.

In addition, we propose two methods to train BCNNw/SF:

Method 1 - Extended Straight-through Estimator (eSTE): take the rank-1 approximation for SFs as a process adding noise into the model and rely on batch normalization to regularize the noise. During backward propagation, we extend the straight-through estimator (STE) to propagate gradient across the decomposition.

Method 2 - Gradient over SVD: go through the analytic closed form of the gradient over SVD to push the chain rule in backward propagation to the binarized filters, which is the filter before SVD.

The rest of the paper is organized as follows: Sec. 2 provides a brief survey of previous works, Sec. 3 presents the design of BCNNw/SF and some implementation details, Sec. 4 presents two methods for the training of BCNNw/SF., Sec. 5 shows experimental results, Sec. 6 describes the implementation of BCNNw/SF on an FPGA platform, and Sec. 7 concludes the paper.

2. Related Works

We leverage the lightweight method for training a BCNN as proposed by Hubara et al. [6, 3], which achieved state-of-the-art results on datasets such as CIFAR-10 and SVHN. Two important ideas contributed to the effectiveness of their BCNN:

Batch normalization with scaling and shifting [7]: A BN layer regularizes the training process by shifting the mean to zero, making binarization more discriminative. It also introduces two extra degrees of freedom in every neuron to further compensate for additive noises.

Larger Model: As with the well-known XOR problem [15], using a larger network increases the power of the model by increasing the number of dimensions for projection and making the decision boundary more complex.

Rastegari et al. proposed XNOR-Net [13], an alternative BCNN formulation which relies on a multiplicative scaling layer instead batch normalization to regularize the additive noise introduced by binarization. The scaling factors are calculated to minimize the 1-norm error between real-valued and binary filters. While Hubara's BCNN did not perform well with a larger dataset such as ImageNet [4], obtaining a top-1 error rate of 72.1%, XNOR-Net improves this error rate to 55.8%.

Rigamonti et al. [14] proposed a rank-1 approximate method to replace the 2-D convolution in a CNN with two successive 1-D convolutions. Every filter was approximated by the outer product of a column vector and a row vector which were obtained through Singular Value Decomposition (SVD). The authors proposed two schemes of the learning of separable filters: (1) retain only the largest singular value and corresponding vectors to reconstruct a filter; (2) linearly combine the outer products to lower the error rate. However, the first scheme sacrificed too much performance because the the other singular values can be comparable with the largest one in terms of magnitude. The second scheme was designed to compensate for loss of performance, but more singular values used to recover a filter means a lesser benefit from the approximation. Although learning with separable filters was computationally expensive, the low rank approximation is an important idea to alleviate hardware complexity.

Inspired by Rigamonti's work, more research projects has been conducted to explore a more economic model, *i.e*. networks with smaller memory requirements for the kernels. Jaderberg *et al*. [8] proposed a filter compression method that analyzed the redundancy in a pre-trained model, decomposed the filters into single-channel separable filters, and then linearly combined separable filters to recover original filters. The decomposition was optimized to minimize the L2 reconstruction error of original filters. Alvarez *et al*. [1] presented DecomposeMe that further reduced the redundancy by sharing the separated filters in the same layer. To alleviate the computational congestion of GoogLeNet [22], Szegedy *et al*. [21, 20] proposed a multi-channel asymmetric convolutional structure, which has the same architecture as the second scheme in the work of Jaderberg *et al*. [8] but in different purposes: Szegedy used the asymmetric convolutional structure to avoid the expensive 2D convolutions and train the filter directly, while Jaderberg decomposed pre-trained filters to exploit both input and output redundancies. However, both Jaderberg's and Alvarez's methods required a pre-trained model, and both Jaderberg's and Szegedy's multi-channel asymmetric convolution brought additional channels requiring a larger memory footprint.

Our proposed method differs from the three methods above because we maintain the network structure during training phase, train rank-1 separable filters directly, and then decompose the rank-1 filters into pairs of vector filters for hardware implementation. Last but not least, to the best of our knowledge no existing work provides an analytic closed form of the gradient of filter-decomposition process for backward propagation.

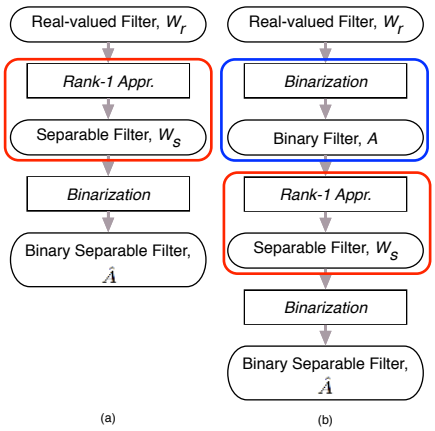

Figure 2. Comparison of the two SVD flows; (a) Flow 1: binarize the result of SVD on the floating-point filter; (b) Flow 2: directly decompose the binarized filters.

3. Binarized CNN with Separable Filters

Here we describe the theory of BCNN with Separable filter in detail. Our main idea is to apply SVD on binarized filters to further reduce the memory requirement and computation complexity for hardware implementation. We present the details of forward propagation in this section and two methods of backward propagation in the next section.

3.1. The Subject of Decomposition

For BCNN, there are two approaches to binary filter decomposition. Fig. 2 depicts the two choices. If we adopt flow 1 and apply the rank-1 approximation (the red box) directly on the real-valued filters, we cannot avoid real-time decomposition during training because the input filter has an infinite number of possible combination of pixel strengths. Therefore, we introduce an extra binarization (the blue box) on the real-valued filters and apply the rank-1 approximation on the binarized filters. Then, the number of possible input filters of rank-1 approximation are limited to 2^{d^2}, where d is the width or height of a filter. With flow 2, we can build a look-up table beforehand and avoid real-time SVD during training.

Naturally, the rank-1 approximation and the extra binarization will limit the size of the basis to recover the original filters and equivalently introduce more noise into the model, as shown in Fig. 1 from (b) to (c). Instead of introducing an additional linear-combination layer to improve the accuracy, we leave the task to the two aforementioned reasons that make BNN work.

3.2. Binarized Separable Filters

Here we provide the detailed steps from binarized filters to binarized separable filters. The result of SVD on a matrix A includes three matrices as shown in Eq. 1.

$$A = UDV^T \tag{1}$$

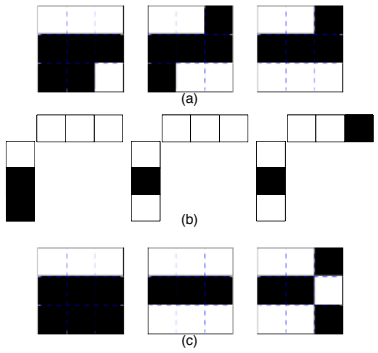

Figure 3. (a) a kernel before approximation; (b) pairs of vectors (u, v^T) in SVD; (c) a kernel after rank-1 approximation in which every filter is an outer product of u, and v. The white and black colors stand for $+1$ and -1, respectively.

Similar to real value rank-1 approximation for separable filter, the binarized separable filters are obtained with an extra binarization process on the dominate singular vectors as shown in Eq. 2.

$$\hat{A} = b(U[:,1])b(V[:,1]^T), \tag{2}$$

where $U[:,1]$ and $V[:,1]$ stand for the left and right singular vector corresponding to the largest singular value, respectively, and the function $b(.)$ denotes the binarization and can be implemented in either a deterministic function or a stochastic process [6]. Please note the largest singular value is dropped because all singular values are always positive and have no effect on binarization.

Fig. 3(a) and (c) illustrates a kernel with three filters before and after binarized rank-1 approximation. As with [6] we keep a copy of the real-valued filters during each training iteration and accumulate the weight gradients on them since SGD and other convex optimization methods presume a continuous hypothesis space. This also allows us to train the kernels as if the model is real-valued without the need for penalty rules [14] during the backward propagation. For the test phase, all filters are binarized and rank-1 approximated to be binarized separable.

In our FPGA implementation, we use the pairs of vectors in Fig. 3(b) to replace 2-D filters and perform separable convolution, which involves a row-wise 1D convolution followed by a column-wise convolution in back-to-back fashion before accumulating across different channels. More details on the FPGA implementation are presented in Sec. 6.

3.3. Details of the Implementation

As mentioned in sec. 3.1, the benefit of flow 2 is to leverage a finite-sized look-up table (LUT) to replace the costly

SVD computation during the forward propagation of training phase. Although the training takes place on a highly-optimized parallel computing machine, the LUT access is still a potential bottleneck if searching for an entry in the mapping is not efficient enough.

We build two tables to avoid real-time SVD. The first table is composed by all binarized separable filters. The number of entries in the first table can be calculated with Eq. 3.

$$K = 2^{2d-1}, \tag{3}$$

where d is the width or height of a filter.

The second table is the mapping relationship between all possible binary filters to their corresponding binarized separable filters. We design an estimation function to make the tables content-addressable. The key to index the first table can be obtained with Eq. 4.

$$key = \Lambda \cdot A, \tag{4}$$

where Λ is a vector or a matrix in the same size of A, and all elements in Λ are the weightings to convert a matrix A into a number. The simplest choice of Λ is the binary-to-integer conversion method. We take the first element in A as the least significant bit (LSB), so the Λ is designed as Eq. 5

$$\Lambda = \begin{bmatrix} 2^0 & 2^1 & 2^2 & \dots & 2^N \end{bmatrix}, \tag{5}$$

where N is the amount of elements of A, and $N = d^2$. With this simple hash function and the efficient broadcasting technique in Theano [23], we are able to efficiently obtain the keys for all filters in a convolutional layer.

4. Backward Propagation of Separable Filters

Besides the extra degrees of freedom introduced to BCNNw/SF's forward propagation, there are two more important techniques making binarized separable filters work. In this section, we present two methods utilizing the two techniques for the training of BCNNw/SF.

4.1. Method 1: Extended STE

As shown in Fig. 2(b), during the forward propagation, all filters must be degraded thrice. Since binarization can be considered as noise addition into the model and be regularized with batch normalization, the rank-1 approximation, which is just another process adding extra noise, can be regularized as well. In details, we extend the straight-through estimator across the three degradation processes in Fig. 2 to update the real-valued filters with the rank-1 approximated filters. Eq. 6 shows the backward propagation of the gradient of rank-1 approximated filter,g_{bs}, to the gradient of real-valued filter, g_r.

$$g_r = g_{\hat{A}} 1_{|r|\leq 1} \tag{6}$$

This simple method totally relies on batch normalization to regularize the noise introduced by two binarization and one rank-1 approximation.

4.2. Method 2: Gradient over SVD

Whereas binarization is not a continuous function, Hubara *et al*. [6] resorted to the STE to update the real-valued weights with the gradient of loss w.r.t binarized weights. Howbeit, owing to the continuity of singular value decomposition, we are allowed to calculate the gradient w.r.t. the resultant of the first binarization, W_b. More specifically, the rank-1 approximation is differentiable because all of the three resultant matrices, *i.e.* U,D, and V, of SVD in Eq. 1 are differentiable w.r.t. every element of the original input matrix, A. From the approximation we adopt for separable filters as shown in Eq. 2, one can easily obtain the derivative of $\hat{A}$ w.r.t. the elements of the original matrix before the approximation as Eq. 7, if the STE for binarization is applied.

$$\frac{\partial \hat{A}}{\partial a_{ij}} = \frac{\partial U[:,1]}{\partial a_{ij}} b(V[:,1]^T) + b(U[:1]) \frac{\partial V[:,1]^T}{\partial a_{ij}} \tag{7}$$

Papadopoulo *et al*. [11] provided the mathematical closed form of the gradient of the three resultant matrices, as shown in Eq. 8, and 9.

$$\frac{\partial U}{\partial a_{ij}} = U \Omega_U^{ij} \tag{8}$$

$$\frac{\partial V}{\partial a_{ij}} = -V \Omega_V^{ij}, \tag{9}$$

where $\Omega_U^i j$ and $\Omega_V^i j$ are anti-symmetric matrices with zeros on their diagonals, and all off-diagonal elements can be obtained by Eq. 10 and 11.

$$\Omega_{U_{kl}}^{ij} = \frac{d_l u_{ik} v_{jl} + d_k u_{il} v_{jk}}{d_l^2 - d_k^2} \tag{10}$$

$$\Omega_{V_{kl}}^{ij} = \frac{d_k u_{ik} v_{jl} + d_l u_{il} v_{jk}}{d_k^2 - d_l^2} \tag{11}$$

Eq. 12 shows the general form of the differential equation.

$$\frac{\partial \hat{A}}{\partial a_{ij}} = \begin{bmatrix} \frac{\partial \hat{a}_{11}}{\partial a_{ij}} & \frac{\partial \hat{a}_{12}}{\partial a_{ij}} & \dots & \frac{\partial \hat{a}_{1N}}{\partial a_{ij}} \\ \frac{\partial \hat{a}_{21}}{\partial a_{ij}} & \frac{\partial \hat{a}_{22}}{\partial a_{ij}} & \dots & \frac{\partial \hat{a}_{2N}}{\partial a_{ij}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \hat{a}_{M1}}{\partial a_{ij}} & \frac{\partial \hat{a}_{M2}}{\partial a_{ij}} & \dots & \frac{\partial \hat{A}_{MN}}{\partial a_{ij}} \end{bmatrix} \tag{12}$$

$$\frac{\partial \hat{a}_{kl}}{\partial a_{ij}} = b(U_{k1}) \sum_{n=2}^{N} V_{kn} \Omega_{V_{1n}}^{ij} - b(V_{l1}) \sum_{n=2}^{N} U_{ln} \Omega_{U_{1n}}^{ij} \tag{13}$$

From Papadopoulo's equations 8 to 11, we can derive every element in Eq. 12 as shown in Eq. 13 and see there exist

cross-terms between elements. The gradient of a SVD resultant matrix w.r.t. one element in the original input matrix is also a matrix of the same dimension, $M$by$N$, *i.e.* a single element's change in the input matrix can affect all other elements in the resultant of SVD. The intuition behind is that the rank-1 approximation is a matrix-wise filter-level mapping relationship rather than an element-wise operation, and multiple elements contribute to the mapping result of a filter.

To recap Eq. 12 with the chain rule calculation of backward propagation, we follow the similar fashion how higher layer neurons collect errors from the lower layer. Eq. 14 shows the inner product for collecting error from lower layer and propagate the error to every element in binarized filters A. For method 2, we also build a table of the derivatives together with the binarized rank-1 approximation to avoid real-time calculation of Eq. 12.

$$\frac{\partial loss}{\partial a_{ij}} \equiv \frac{dloss}{d\hat{A}} \cdot \frac{\partial \hat{A}}{\partial a_{ij}} \tag{14}$$

5. Experiments

We conduct experiments on the Theano [23] based on the Courbariaux's framework [2], using 2 GPUs: NVIDIA GeForce GTX Titan X and GTX 970 to finish the training/testing process. In most of the experiments, we obtain near state-of-the-art results using BCNNw/SF.

In this section, we describe the network structures we use, and list the classification result on 3 datasets. We compare our result with relevant works, and then make analysis on different perspectives, including binarized separable filter and learning ripples.

5.1. Datasets and Models

We evaluate our methods on three benchmark image classification datasets: MNIST, CIFAR-10 and SVHN. MNIST is a dataset for 28x28 gray-scale handwritten digits, which has a training set of 60K examples, and a testing set of 10K examples. SVHN is a real-world image dataset for street view house numbers, cropped to 32x32 color images, with 604K digits for training, 26K digits for testing. Both of these datasets classify digits ranging from 0 to 9. CIFAR-10 dataset consists of 60K 32x32 color images in 10 mutually exclusive classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck), with $6,000$ images per class. There are 50K training images and 10K test images.

The convolutional neural networks we use has almost the same architecture as Hubara *et al.* [6]'s except for some small modification. This architecture is inspired from the VGG [17] network. It contains 3 fully-connected layers and 6 convolutional layers, in which the kernels for convolutional layers is 3 x 3. For detailed network structure parameters, see Table 1.

Name	MNIST(CNN)	CIFAR-10	SVHN
Input	1x28	3x32x32	3x32x32
Conv-1	64x3x3	128x3x3	64x3x3
Conv-2	64x3x3	128x3x3	64x3x3
Pooling	2 x 2 Max Pooling		
Conv-3	128x3x3	256x3x3	128x3x3
Conv-4	128x3x3	256x3x3	128x3x3
Pooling	2 x 2 Max Pooling		
Conv-5	256x3x3	512x3x3	256x3x3
Conv-6	256x3x3	512x3x3	256x3x3
Pooling	2 x 2 Max Pooling		
FC-1	1024	1024	1024
FC-2	1024	1024	1024
FC-3	10	10	10

Table 1. Network architecture for different datasets. The dimension of a convolutional layer's kernel stands for number of kernels on the concerned layer M, number of channels C, the width of a filter W, and the height of a filter H; the dimension of a fully-connected layer's weights means the number of preceding layer's neurons and the number of the concerned layers' neurons.

In each experiment, we split the dataset into 3 parts: 90% of the training set is used for training the network, the remaining 10% is used as validation set. During the training, we use both the training loss on training set and inference error-rate on the validation set as performance measurements. To evaluate the different trained models, we use the classification accuracy on the testing set as the evaluation protocol.

In order for all these benchmark to remain challenging, We didn't use any pre-processing, data-augmentation or unsupervised learning. We use binarized hard tangent [6] function as activation function. The ADAM adaptive learning rate method [10] is used while minimizing the square hinge loss with an exponentially decayed learning rate. We also apply batch normalization to our networks, with a mini-batch of size 100, 50 and 50 (separately for MNIST, CIFAR-10 and SVHN), to speed up the learning, and we scale the learning rate for each convolutional layer with a factor from Glorot's batch normalization [7]. We train our networks for 300 epochs on MNIST and CIFAR-10 datasets, and 200 epochs on SVHN datasets. The results are given in Sec. 5.2.

5.2. Benchmark Result

Fig. 4 depicts the learning curves on CIFAR-10 dataset. There exists certain accuracy degradation if we compare BCNN with our methods due to a more aggressive noise. By the end of the training phase, our method 1 yields an accuracy less than that of BCNN by roughly 2.72%, and the method 2 reaches a even more inferior accuracy. For the sake of CIFAR-10's higher difficulty, the loss of accuracy meets our expectation. We will discuss in detail the bene-

Dataset	MNIST(CNN)	CIFAR-10	SVHN
No binarization (standard results)			
Maxout Networks [5]	0.94%	11.68%	2.47%
Binarized Network			
BCNN(BinaryNet) [6]	0.47%	11.40%	2.80%
Binarized Network with Separable Filters			
BCNNw/SF Method 1 (this work)	0.48%	14.12%	4.60%
BCNNw/SF Method 2 (this work)	0.56%	15.46%	4.18%

Table 2. Error Rate Comparison on Different Datasets. BCNNw/SF1 stands for our training method 1; BCNNw/SF2 denotes for our training method 2.

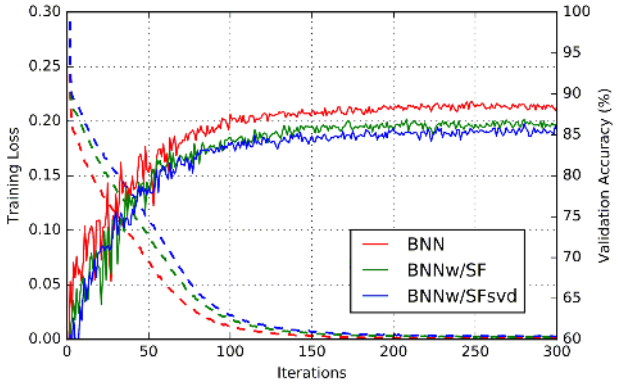

Figure 4. Learning Curves of ConvNets for BNN(red), BNNw/SF1(green) and BNNw/SF2(blue) on CIFAR-10 dataset. The dotted lines represent the training costs(square hinge losses) and the continuous lines the corresponding validation accuracy.

fit of using exact gradient over the rank-1 approximation in next sub-section.

Tab. 2 summarizes the experimental results in terms of error rate. Compared with BNN [6], for the gray-scale manuscript number classification, both of our two training methods achieve a accuracy close to that of the binarized convolutional neural networks. The difference is within 0.09%. It is noteworthy that our method 2 outperforms method 1 on SVHN by 0.42% error rate. For CIFAR-10 and SVHN, our methods are inferior to BCNN by a difference less than 2.72% because we limit choices of filters from a number of 512 to 32, where the filter size is 3x3. Since the performance degradation on CIFAR-10 is the largest, we implement a hardware accelerator in FPGA to inspect at what extent of hardware complexity can be improved with the sacrifice of the 2.72% accuracy loss. Sec. 6 provides the details and a comparison with a BCNN accelerator to demonstrate the benefits of BCNNw/SF.

5.3. Scalability

We also explore different sizes of networks to improve the accuracy and exam the scalability of BCNNw/SF. Tab. 3 lists two additional larger models and an AlexNet-like model for CIFAR-10. The wider one stands for a model with all numbers of kernels doubled, and the deeper one is a network including two extra convolutional layers. Different from the models above, the AlexNet-like model includes three sizes of filters: 5-by-5, 3-by-3, and 1-by-1. Applying our rank-1 approximation on 5-by-5 filter, we can get 64% memory reduction. We train the three bigger networks

Name	Deeper	Wider	AlexNet-like
Input	3x32x32	3x32x32	3x32x32
Conv-1	128x3x3	256x3x3	96x5x5
Conv-2	128x3x3	256x3x3	256x5x5
Pooling	2 x 2 Max Pooling		
Conv-3	256x3x3	512x3x3	512x3x3
Conv-4	256x3x3	512x3x3	512x3x3
Pooling	2 x 2 Max Pooling		
Conv-5	512x3x3	1024x3x3	256x3x3
Conv-6	512x3x3	1024x3x3	512x1x1
Pooling	2 x 2 Max Pooling		
Conv-7	512x3x3	-	-
Conv-8	512x3x3	-	-
Pooling	2x2 Max Pooling	-	-
FC-1	1024	1024	1024
FC-2	1024	1024	128
FC-3	10	10	10

Table 3. The 1st column shows a deeper model with two extra convolutional layers, and the 2nd column shows a widened network with all numbers of kernels doubled. The 3rd column is inspired by AlexNet to include 3 sizes of filters.

with our method 1, and Fig. 5 shows the learning curves of the two enlarged models for CIFAR-10. Since the number of trainable parameters has been increased, it requires more epochs to travel in the hypothesis space and reach a local minimum. Therefore, we train these two bigger networks with 500 epochs, and compare with BCNN(BinaryNet). As shown in Fig. 5 the wider one (blue) starts with largest ripple yet catch up the same performance as BCNN(black) does around the 175th epoch.

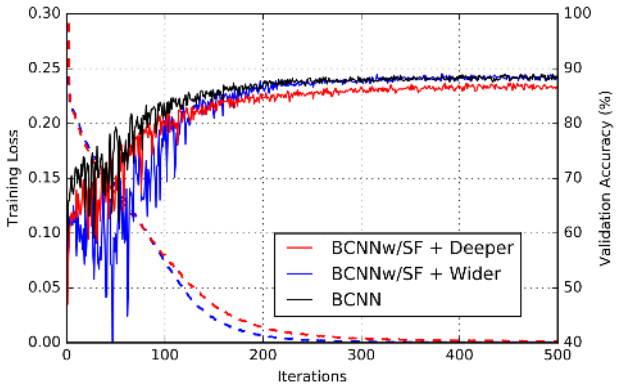

Figure 5. Learning Curves of ConvNets for BCNNw/SF with deeper Network(red), BCNNw/SF with wider Network(blue) and original BCNN on CIFAR-10 dataset. The dotted lines represent the training costs(square hinge losses) and the continuous lines the corresponding validation error rates.

Dataset	CIFAR-10
BCNN(BinaryNet) [6]	11.40%
Binarized Network with Separable Filters (this work)	
BCNNw/SF Method 1	14.12%
BCNNw/SF Method 1 depper	14.11%
BCNNw/SF Method 1 wider	**11.68%**
BCNNw/SF Method 1 AlexNet-like	15.1%

Table 4. Classification Accuracy (Error Rate) of the three larger models.

Tab. 4 lists the results on CIFAR-10 of the three bigger models as well as the CIFAR-10 results in Tab. 2. The performance improvement of deeper network is very scarce since the feature maps experience the extra destructive max pooling layer as shown in Tab. 3, which reduces the size of the first fully-connected layer, FC-1, and hence suppresses the improvement. The wider network achieves 11.68%, which is very close to the performance of BCNN(BinaryNet). The AlexNet-like model demonstrates that a model with 5-by-5 filters sacrifices more accuracy to provide higher memory reduction. In summary, the accuracy degradation of BCNNw/SF can be compensated by enlarging the size of network.

5.4. Discussion

In this section, we use the experimental results on CIFAR-10 as an example of detailed analysis. We unpack the trained rank-1 filters and learning curves to gain a better understanding of the mechanism of BCNNw/SF.

Fig. 6 lists all the 32 rank-1 filters and their frequency on CIFAR-10. Although certain filters are rarely used, there is no filter forsaken. In Fig. 6 we can learn that the all-positive and all-negative filters are trained most frequently,

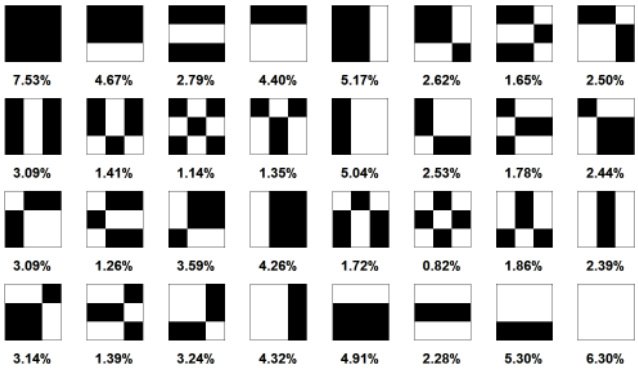

Figure 6. Separable Filters & Frequencies used in CIFAR Model

Statistics	mean	std	max
BCNN(BinaryNet) [6]	0.052	1.213	5.09
BCNNw/SF Method 1	0.055	1.059	4.465
BCNNw/SF Method 2	0.035	0.723	3.622

Table 5. The statistics of the ripples in terms of percent of error rate.

and these two filters render the convolution to running-sum calculation with a sliding window. As mentioned in Sec. 3.1,through the summation of separated convolution from a preceding layer, we can achieve the tangled linear combinations, which are essential to BCNNw/SF.

Unknowing the spectrum of the ripple, we apply Savitsky-Golay filter [16] to obtain the baseline of validation accuracy and, thereby, subtract the original accuracy with the baseline to get the ripple. The window width of the Savitsky-Golay filter is 51, and we use quadratic equation to fit the original learning curve. All ripples are quantized into 100 categories for the statistic analysis. Tab. 5 compares our method 1 and methods 2 with BCNN. All three statistic values of the method 2 are reduced. The analytic gradient over the rank-1 approximation stabilizes the descending trajectory with more accurate gradient calculation. Both BCNN and our method 1 rely on the gradient w.r.t. binarized filters to update all parameters due to the lack of analytic gradient w.r.t. real-valued filters. However, it is also the rigorous gradient that limits the possibility to escape a local minimum on the error surface. As we can see in Tab. 2, the results of our method 1 are closer to that of BCNN. We use the trained binarized separable filter from our method 1 to implement a FPGA accelerator for CIFAR-10 in the following section.

6. FPGA Accelerator

6.1. Platform and Implementation

To quantify the benefits that BCNNw/SF can achieve for hardware BCNN accelerators, we created an FPGA accelerator for the six convolutional layers of the Courbariaux's

CIFAR-10 network. Our accelerator is built from the open-source FPGA implementation in [24]. The dense layers were excluded as they are not affected by our technique. As BCNNw/SF is ideal for small, low-power platforms, we targeted a Zedboard with a Xilinx XC7Z020 FPGA and an embedded ARM processor. This is a much smaller FPGA device compared to existing CNN FPGA accelerators [12, 19]. We write our design in C++ and use Xilinx's SDSoC tool to generate Verilog through high-level synthesis. We implement both BCNN and BCNNw/SF and examine the performance and resource usage of the accelerator with and without separable filters.

Our accelerator is designed to be small and resource-efficient; it classifies a single image at a time, and executes each layer sequentially. The accelerator contains two primary compute complexes: `Conv1` computes the first (non-binary) convolutional layer, and `Conv2-5` is configurable to compute any of the binary convolutional layers. Other elements include hardware to perform pooling and batch normalization, as well as on-chip RAMs to store the feature maps and weights. Computation with the accelerator proceeds as follows. Initially all input images and layer weights are stored in off-chip memory accessible from both CPU and FPGA. The FPGA loads an image into local RAM, then for each layer it loads the layer's weights and performs computation. Larger layers require several accelerator calls due to limited on-chip weight storage. Intermediate feature maps are fully stored on-chip. After completing the convolutional layers we write the feature maps back to main memory and the CPU computes the dense layers.

We kept the BCNN and BCNNw/SF implementations as similar as possible, with the main difference being the convolution logic and storage of the weights. For BCNN, each output pixel requires $3 \times 3 = 9$ MAC operations to compute. For BCNNw/SF we can apply a 3x1 vertical followed by a 1x3 horizontal convolution, a total of 6 MACs. As the MACs are implemented by XORs and an adder tree, BCNNw/SF can potentially save resource.

In terms of storage, BCNN requires the 9 bits to store each filter. Naively, BCNNw/SF requires 6 bits, as each filter is represented as two 3-bit vectors. However, recall we only use rank-1 filters — Eq. 3 shows that the number of unique 3×3 is 32, meaning we can encode them losslessly with only 5 bits. A small decoder in the design is used to map the 5-bit encodings into 6-bit filters.

6.2. Results and Discussion

Table 6 compares the execution time and resource usage of the two FPGA implementations. Resource numbers are reported post place and route, and runtime is wall clock measured on a real Zedboard. We exclude the time taken to transfer the final feature maps from FPGA to main memory, as it is equal between the two networks; transfer time for the initial image and weights are included.

	BCNN	BCNNw/SF (this work)	δ
Conv layer runtime (ms)	0.949	0.652	-31.3%
LUT	35255	36384	+3.2%
FF	41418	41054	-1.0%
Block RAM	94	78	-17.0%
DSP	8	8	0.0%

Table 6. Comparison of performance and resource usage between BCNN and BCNNw/SF FPGA implementations. Runtime is for a single image, averaged over 10000 samples.

Our experimental results show that BCNNw/SF achieves runtime reduction of 31% over BCNN, which equates to a 1.46X speedup. This is due mostly to the reduction of memory transfer time of the compressed weight filters. For similar reasons BCNNw/SF is able to save 17% of the total block RAM (RAMs are used for both features and weights). Look-up table (LUT) counts have increased slightly, due most likely to the additional logic needed to map the 5-bit encodings to actual filters. Overall, BCNNw/SF realizes significant improvements to performance and memory requirement with minimal logic overhead.

7. Conclusion and Future Work

In this paper, we proposed binarized convolutional neural network with Separable Filters (BCNNw/SF) to make BCNN more hardware-friendly. Through binarized rank-1 approximation, 2D filters are separated into two vectors, which reduce memory footprint and the number of logic operations. We have implemented two methods to train BCNNw/SF with Theano and verified our methods with various CNN architectures on a suite of realistic image datasets. The first method relies on batch normalization to regularize noise, making it simpler and faster to train, while the second method uses gradient over SVD to make the learning curve more smooth and potentially achieves better accuracy. We also implement an accelerator for the inference of a CIFAR-10 network on an FPGA platform. With separable filters, the total memory footprint is reduced by 17.0% and the performance of the convolution layers is improved by 1.46X compared to baseline BCNN.

Integrating probabilistic methods [18] to reduce the training time and exploring more elegant structures of networks [20] will be a promising direction for future works.

References

[1] J. Alvarez and L. Petersson. DecomposeMe: Simplifying ConvNets for End-to-End Learning. *arXiv e-print*, arXiv:1606.05426, Jun 2016. 2

[2] M. Courbariaux. BinaryNet. https://github.com/MatthieuCourbariaux/BinaryNet/, 2016. 5

[3] M. Courbariaux, Y. Bengio, and J.-P. David. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. *Advances in Neural Information Processing Systems (NIPS)*, pages 3123–3131, 2015. 1, 2

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. pages 248–255, Jun 2009. 2

[5] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. C. Courville, and Y. Bengio. Maxout Networks. *Int'l Conf. on Machine Learning (ICML)*, pages 1319–1327, Feb 2013. 6

[6] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized Neural Networks. *Advances in Neural Information Processing Systems (NIPS)*, 2016. 1, 2, 3, 4, 5, 6, 7

[7] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv e-print*, arXiv:1502.03167, Mar 2015. 2, 5

[8] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up Convolutional Neural Networks with Low Rank Expansions. *arXiv e-print*, arXiv:1405.3866, May 2014. 2

[9] M. Kim and P. Smaragdis. Bitwise Neural Networks. *arXiv e-print*, arXiv:1601.06071, Jan 2016. 1

[10] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv e-print*, arXiv:1412.6980, Dec 2014. 5

[11] T. Papadopoulo and M. I. Lourakis. Estimating the jacobian of the singular value decomposition: Theory and applications. *European Conference on Computer Vision (ECCV)*, pages 554–570, 2000. 4

[12] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, et al. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, pages 26–35, Feb 2016. 8

[13] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. *European Conference on Computer Vision (ECCV)*, Oct 2016. arXiv:1603.05279. 1, 2

[14] R. Rigamonti, A. Sironi, V. Lepetit, and P. Fua. Learning Separable Filters. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2754–2761, 2013. 2, 3

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. *CALIFORNIA UNIV SAN DIEGO LA JOLLA INST FOR COGNITIVE SCIENCE*, 1985. 2

[16] A. Savitzky and M. J. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, pages 1627–1639, Jul 1964. 7

[17] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv e-print*, arXiv:1409.1556, Sep 2014. 5

[18] D. Soudry, I. Hubara, and R. Meir. Expectation backpropagation: parameter-free training of multilayer neural networks with continuous or discrete weights. *Advances in Neural Information Processing Systems (NIPS)*, pages 963–971, 2014. 1, 8

[19] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, and Y. Cao. Throughput-Optimal OpenCL-based FPGA Accelerator for Large-Scale Convolutional Neural Networks. *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, pages 16–25, Feb 2016. 8

[20] C. Szegedy, S. Ioffe, and V. Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *arXiv e-print*, arXiv:1602.07261, Feb 2016. 2, 8

[21] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *arXiv e-print*, arXiv:1512.00567, Dec 2015. 2

[22] C. Szegedy, Y. J. Wei Liu, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015. 2

[23] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-print*, arXiv:1605.02688, May 2016. 4, 5

[24] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang. Accelerating Binarized Convolutional Neural Networks with Software-Programmable FPGAs. *Int'l Symp. on Field-Programmable Gate Arrays (FPGA)*, Feb 2017. 8