Combining Learnable Low-dimensional Binary Filter Bases for Compressing Convolutional Neural Networks

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

17-11-2021 / 19-11-2021

CITATION

Lan, Weichao; Cheung, Yiu-ming; Lan, Liang (2021): Combining Learnable Low-dimensional Binary Filter Bases for Compressing Convolutional Neural Networks. TechRxiv. Preprint.
<https://doi.org/10.36227/techrxiv.17031917.v1>

DOI

[10.36227/techrxiv.17031917.v1](https://doi.org/10.36227/techrxiv.17031917.v1)

Combining Learnable Low-dimensional Binary Filter Bases for Compressing Convolutional Neural Networks

Weichao Lan, Yiu-Ming Cheung, Liang Lan

Department of Computer Science, Hong Kong Baptist University
Hong Kong SAR, China

cswclan@comp.hkbu.edu.hk, ymc@comp.hkbu.edu.hk, lanliang@hkbu.edu.hk

Abstract

Existing convolutional neural networks (CNNs) have achieved significant performance on various real-life tasks, but a large number of parameters in convolutional layers requires huge storage and computation resources which makes it difficult to deploy CNNs on memory-constraint embedded devices. In this paper, we propose a novel compression method that generates the convolution filters in each layer by combining a set of learnable low-dimensional binary filter bases. The proposed method designs more compact convolution filters by stacking the linear combinations of these filter bases. Because of binary filters, the compact filters can be represented using less number of bits so that the network can be highly compressed. Furthermore, we explore the sparsity of coefficient through L1-ball projection when conducting linear combination to avoid overfitting. In addition, we analyze the compression performance of the proposed method in detail. Evaluations on four benchmark datasets under VGG-16 and ResNet-18 structures show that the proposed method can achieve a higher compression ratio with comparable accuracy compared with the existing state-of-the-art filter decomposition and network quantization methods.

1. Introduction

Currently, deep convolutional neural networks (CNNs) have become one of the most popular tools in various fields like computer vision [11, 28], natural language process [35] and speech recognition [1] because of their effectiveness and remarkable performance. However, a CNN model with good performance consists of a huge number of parameters, which requires considerable storage space and time-consuming computation [26]. A typical example is the ResNet-50 with 50 layers which requires more than 95 MB of memory to store the model and 3.8 billion floating-point operations (FLOPs) for a single prediction. Although pow-

erful GPUs and CPUs can help deal with computationally complex tasks when training the network, the huge resource consumption still limits the applications of deep CNNs on resource-constraint devices such as mobile phones, smart watches and other edge devices at the stage of inference. Therefore, in order to make deep CNNs deployable on resource-constraint devices, it is desired that the number of parameters in CNNs can be reduced significantly through compression.

In recent years, different model compression methods have been proposed. These studies have shown that the CNN models can still work as usual but the memory and computing time will be saved significantly after discarding redundancy. These methods can be roughly divided into five categories: (1) network pruning [8, 32]: removing redundant components such as channels and filters; (2) network quantization [14]: using low-bit representation of parameters; (3) filter decomposition [18, 36]: approximating the original filter with lightweight kernels; (4) knowledge distillation [16, 29]: training shallower models based on large deeper models. (5) compact model design [12, 13]: directly designing a more compact network with special operations such as spatial convolution and shift convolution. Usually, the compact networks can be further pruned or optimized based on actual requirements. Recently, the re-designed lightweight models have attracted more and more attention due to their impressive success in different tasks. These methods aim at establishing deep neural networks with fewer parameters and lower storage consumption to achieve high efficiency, which also provides more choices for automatic compression like neural architecture search (NAS) [33].

Among the various compression methods, filter decomposition simplifies the network by abandoning unimportant kernel components and provides an optimal solution of a given weight matrix through minimizing the approximation error. Early low-rank approximation methods utilize mathematical operations such as SVD [7], tucker decomposition [18] and CP-decomposition [22] that decompose the

original filters into two to four components. [15] decomposes the 2D $w \times h$ kernel into the multiplication of two small 1D vectors with size of $w \times 1$ and $1 \times h$. [18] considers the low rank property both of input channel and output channel so that the 2D kernel can be decomposed into two 1×1 kernels and one $w \times h$ kernel through tucker decomposition. Despite the impressive acceleration effect of these filter decomposition methods, there is still a limitation on the actual compression in terms of the number of parameters. Further, the decomposition usually relies on rank selection and contains multiple or complex decompositions that is time-consuming.

Thus, to overcome the compression limitation while alleviating these issues, we propose a new method to generate compact convolution filters by stacking sparse linear combinations of a set of low-dimensional binary bases. Both the sparse linear coefficients and binary bases can be directly learned and optimized. Through binary values, we can then explore the redundancy in representation by reducing the bit number of parameters. Due to the high sparsity of the linear combination coefficients and low-memory cost of binary bases, the proposed method can achieve a higher compression ratio with comparable accuracy in comparison with the state-of-the-art decomposition and quantization methods. The main contributions of this paper are summarized as follows:

- (1) We propose a novel method that can generate compact convolution filters by stacking the sparse linear combinations of a set of learnable low-dimensional binary filter bases.
- (2) We develop an effective algorithm to learn the sparse coefficient and low-dimensional binary bases.
- (3) The proposed method is flexible and can be generalized easily to meet up different compression requirements by adjusting the number and dimensions of filter bases.
- (4) We conduct the experiments on popular VGG and ResNet structures with four benchmark datasets to demonstrate the effectiveness of the proposed method. The experimental results show that the proposed method can make a good tradeoff between accuracy and compression ratio.

2. Related Work

2.1. Binary Neural Network

The initial intention of binary neural networks is that binary parameters can greatly reduce the memory cost since they can be stored in low-bit format. At the same time, complex computations such as multiplication can be substituted

with cheap operations like addition and subtraction. BinaryConnect (BC) [3] first considers binary weights which limits the values in weight tensors as only 1 or -1. Binarized Neural Network (BNN) [4] extends the idea of BC by introducing binarized activations. Further work in [27] proposes another two binary networks called Binary Weight Network (BWN) and XNOR-Net, where full-precision scaling factors are introduced to narrow the gap between binarized values and real values based on BC and BNN, respectively. In the existing works, parameters with real values are reserved for updating during training process according to STE strategy [2]. In spite of the fact that binary neural networks have been proved to be beneficial for network acceleration and compression, the accuracy loss compared to full-precision network is still large, which makes it a challenge to enhance the performance of binary networks in many tasks.

2.2. Filter Decomposition

In a general neural network, tensor operation occupies most of the computation cost, so decomposition or approximation of tensors is a favorable method for compression and acceleration [6], which uses several low rank matrices to approximate the original full rank weights. [36] assumes a low rank property of output feature to estimate the original weight with the production of two low rank matrices. PENNI [23] conducts SVD on the weight to obtain a small number of bases with lower dimensions. Binary weighted decomposition (BWD) [17] expands each filter in a convolutional layer into the production of several binary filters and full-precision scaling factors. In addition to decomposed binary filters, ABC-Net [24] further approximates full-precision activations with the linear combination of some preset binary activation bases. Compared with XNOR-Net, ABC-Net gets better results because it applies multiple binary weights and activations, but the complexity also increases at the same time that contradicts with the initial purpose of binary neural networks [37].

3. Preliminaries

Convolution Filter. In general, a convolutional layer in a CNN transforms a three-dimensional tensor with width, height and depth to another tensor with the same dimensions. For a given layer, we represent the input and output tensor as $\mathbf{X}_{in} \in \mathbb{R}^{w_{in} \times h_{in} \times c_{in}}$ and $\mathbf{X}_{out} \in \mathbb{R}^{w_{out} \times h_{out} \times c_{out}}$, respectively. Then the weight tensor can be written as a four-dimensional tensor $\mathbf{W} \in \mathbb{R}^{w \times h \times c_{in} \times c_{out}}$, where the four dimensions correspond to width, height of the convolution kernel and the number of input and output channels. Usually the kernel is in a square shape so that the width w and height h is set as the same value d . Let us use "Conv" to denote the convolution operation, then the output of a convolutional layer can be calculated as:

Figure 1. The proposed method to approximate convolution filters. The convolution filters are generated from the linear combination of a set of low-dimensional binary filters with sparse coefficients.

$$\mathbf{X}_{out} = a(\text{Conv}(\mathbf{X}_{in}, \mathbf{W}) + \mathbf{b}), \quad (1)$$

where $\mathbf{b}$ is the bias and $a(\cdot)$ represents the activation function usually chosen to be Rectified Linear Units (ReLU) function in CNN. Then we can obtain the final output $\hat{\mathbf{Y}}$ through pooling and fully-connected layers.

Given a set of data pairs $\{(\mathbf{x}, \mathbf{y})\}$, the training of a standard CNN can be regarded as a minimization problem,

$$\begin{aligned} \arg \min_{\mathbf{W}, \mathbf{b}} \mathcal{L}(\hat{\mathbf{Y}}, \mathbf{y}) &= \arg \min_{\mathbf{W}, \mathbf{b}} \mathcal{L}(f((\mathbf{W}, \mathbf{b}); \mathbf{x}), \mathbf{y}) \\ &= \arg \min_{\mathbf{W}, \mathbf{b}} \mathcal{L}(f(\mathbf{X}_{out}), \mathbf{y}) \end{aligned} \quad (2)$$

where $\mathbf{y}$ is the given image label or ground truth and $\hat{\mathbf{Y}}$ is the predicted value of network. $\mathcal{L}(\cdot)$ denotes a predefined loss function such as least-square loss and cross-entropy loss for classification, while $f(\cdot)$ represents the operations in pooling and fully-connected layers. This objective function can be solved by any popular optimizer such as stochastic gradient descent (SGD) to learn the parameters of weight $\mathbf{W}$ and bias $\mathbf{b}$. During training, the output of each convolutional layer is first computed in forward propagation, then the gradients are calculated and the parameters are updated in backward propagation. In this way, a CNN can be trained for better expression.

4. The Proposed Method

4.1. Convolution Filters Approximation

In the commonly used filter decomposition methods for compressing CNNs, the original large weight matrix is commonly represented to be the multiplication of several low-rank matrices for saving memory and computation [36]. However, these methods still use full-precision bases and

fixed rank that limit the compression performance. To overcome these limitations, we design a new kind of compact convolution filters containing multiple low-dimensional binary bases and sparse coefficient to approximate the original convolution filters. Instead of decomposing the original weight tensor that is complex and laborious, the filter bases and coefficient in our method are generated randomly at the beginning, then they are directly learned from data samples during training and updated in backward propagation. Therefore, the proposed method is more flexible than prior work on filter decomposition.

Figure 1 illustrates the overview of the proposed method to approximate convolution filters for a given layer. The idea is that the original weight tensor $\mathbf{W} \in \mathbb{R}^{d \times d \times c_{in} \times c_{out}}$ for a given layer is reconstructed from a set of m low-dimensional binary bases $\{\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_m\}$ where $\mathbf{B}_i \in \{1, -1\}^{d \times d \times n}$ with $n \ll c_{in}$ and a sparse full-precision coefficient $G \in \mathbb{R}^{m \times \frac{c_{in}}{n} \times c_{out}}$ in Figure 1a. Specifically, the input is first split into $q = \frac{c_{in}}{n}$ parts, then the t -th convolution filter $\mathbf{W}^t \in \mathbb{R}^{d \times d \times c_{in}}$ is approximated by stacking q low-dimensional convolution filter blocks $\mathbf{W}_i^t \in \mathbb{R}^{d \times d \times n}$, that is, $\mathbf{W}^t = [\mathbf{W}_1^t, \mathbf{W}_2^t, \dots, \mathbf{W}_q^t]$. As shown in Figure 1b, each building block $\mathbf{W}_i^t$ is a linear combination of the binary bases $\{\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_m\}$,

$$\mathbf{W}_i^t = G_{1i}^t \mathbf{B}_1 + G_{2i}^t \mathbf{B}_2 + \dots + G_{mi}^t \mathbf{B}_m = \sum_{j=1}^m G_{ji}^t \mathbf{B}_j. \quad (3)$$

where $G^t \in \mathbb{R}^{m \times \frac{c_{in}}{n}}$ is a coefficient matrix and each scaling factor G_{ji}^t is an element in this matrix. Then, the t -th convolution filter will be approximated as:

$$\begin{aligned} \mathbf{W}^t &= [\mathbf{W}_1^t, \mathbf{W}_2^t, \dots, \mathbf{W}_q^t] \\ &= [\sum_{j=1}^m G_{j1}^t \mathbf{B}_j, \sum_{j=1}^m G_{j2}^t \mathbf{B}_j, \dots, \sum_{j=1}^m G_{jq}^t \mathbf{B}_j]. \end{aligned} \quad (4)$$

Algorithm 1 $\epsilon - L_1$ ball projection. [30]

Input: vector $\mathbf{v} \in \mathbb{R}^n$, L_1 ball radius r , tolerance t

Output: new sparse vector $\mathbf{v}$

```
1: If  $\|\mathbf{v}\| \leq r(1+t)$  return  $\mathbf{v}$ 
2:  $low = 0; up = \|\mathbf{v}\|_\infty; b = \|\mathbf{v}\|_1$ 
3: while  $b > r(1+t)$  or  $b < r$  do
4:    $\theta = (up + low)/2$ 
5:    $b = \sum_{i=1}^n \max(0, |v_i| - \theta)$ 
6:   If  $b \leq r$  then  $up = \theta$  else  $low = \theta$ 
7: end while
8: for  $i = 1$  to  $n$  do
9:    $\mathbf{v}_i = \text{sign}(\mathbf{v}_i) * \max(0, |v_i| - \theta)$ 
10: end for
```

The proposed idea concatenates q low-dimensional filters with depth n to form the high dimensional filter with depth c_{in} . And each low-dimensional filter block is a linear combination of m binary bases. These m binary bases are shared across all convolution filters for a given layer. Therefore, in a given convolution layer, we only need to store binary bases $\{\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_m\}$ and the sparse coefficient tensor G . Since each binary value can be stored in 1-bit, the memory cost of m binary bases is $d \times d \times n \times m$ bits. To store the sparse coefficient tensor G , we can use the popular coordinate (COO) list which utilizes arrays to store non-zero values and their corresponding coordinates. The memory cost for storing G using COO is $nnz(G) \times 4 \times 32$ bits where $nnz(G)$ denotes the number of non-zeros values in G . Compared with the original weight tensor $\mathbf{W} \in \mathbb{R}^{d \times d \times c_{in} \times c_{out}}$ whose memory cost is $d \times d \times c_{in} \times c_{out} \times 32$ bits, the compression ratio of proposed method can be computed as:

$$\frac{d \times d \times c_{in} \times c_{out} \times 32}{nnz(G) \times 4 \times 32 + d \times d \times n \times m}. \quad (5)$$

Use s to denote the sparsity rate of the coefficient tensor G , then the $nnz(G) = m \times q \times c_{out} \times (1 - s)$. Thus, (5) can be rewritten as

$$\frac{1}{\frac{m \times q \times (1-s) \times 4}{c_{in} \times d \times d} + \frac{m}{c_{out} \times q \times 32}}. \quad (6)$$

4.2. Sparsity of Coefficient

As shown in (6), we seek sparse solution for G to achieve high compression ratio. For the coefficient $G \in \mathbb{R}^{m \times q \times c_{out}}$, each column $\mathbf{g} \in \mathbb{R}^m$ represents the scaling factor to generate each filter block $\mathbf{W}_i^t$ as in Equation (3).

Many works have studied the sparsity property and proved that it can not only save the storage and computational consumption of the networks but also prevent over-fitting. Achieving sparsity through regularization is one of

Algorithm 2 Training Compressed CNN with linear combination of stacked small binary filters.

Input: training data $\mathbf{X}_{train}$, corresponding label $\mathbf{y}_{train}$, hyperparameter for L_1 projection r (radius) and t (tolerance), parameter for binary filter q and m

Output: compressed network

```
1: Random Generate proxy variables  $\{\mathbf{A}_1, \dots, \mathbf{A}_m\}$  and coefficient  $G$  based on CNN configuration  $q$  and  $m$ 
2: for iter = 1 to maxIter do
3:   Get a minibatch of training data  $\{\mathbf{X}, \mathbf{y}\}$ 
4:   for  $l = 1$  to  $L$  do
5:     Get  $m$  binary filter bases  $\{\mathbf{B}_1, \dots, \mathbf{B}_m\}$  according to (9)
6:     Apply Algorithm 1 on each column  $\mathbf{g}$  of coefficient  $G$  to achieve sparsity
7:   end for
8:   Conduct forward propagation
9:   Compute the loss in (7)
10:  Conduct backward propagation based on (10)
11:  Clip the binarized weights in the range of  $\{-1, 1\}$ 
12:  Update parameters using any popular optimizer (e.g., SGD with momentum or ADAM)
13: end for
```

the popular methods such as L_0 [25] and L_1 [9] regularization. In the proposed method, the sparse coefficient is obtained by $\epsilon - L_1$ ball projection [30]. As described in Algorithm 1, the radius of L_1 -ball is constrained between λ and $(1 + \epsilon)\lambda$, then the original dense vector is projected on the L_1 -ball after finding a proper threshold θ based on bisection. The element will be changed to zero if its absolute value is smaller than the threshold θ so that sparsity can be achieved. Compared to a random set threshold, $\epsilon - L_1$ ball projection can calculate a more accurate threshold based on the distribution of data.

4.3. Training Process

Note that the input $\mathbf{X}_{in} \in \mathbb{R}^{w_{in} \times h_{in} \times c_{in}}$ for a given convolutional layer has already been split and the original filter is divided into $q = \frac{c_{in}}{n}$ parts. When training, the parameters that need to be learned and updated in our methods are the set of binary bases $\{\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_m\}$ and coefficient matrix G . Based on the optimization problem in Equation 2, the objective function of the proposed method can be set as:

$$\min \sum_{t=1}^{c_{out}} \mathcal{L}(f(a(\sum_{i=1}^q \text{Conv}(\mathbf{X}_{in(i)}, \sum_{j=1}^m G_{ij}^t \mathbf{B}_j))), \mathbf{y}) \quad (7)$$
$$s.t. \mathbf{B}_j \in \{-1, 1\}.$$

where $\mathcal{L}(\cdot)$ is the loss function. In a given layer, the convolution operation in this objective actually contains two

stages: the linear combination of low-dimensional binary bases with coefficients first constructs the whole filter and then the new compact filter is convoluted with the input feature maps. The drawback is that repeated convolution operation may occur if the same bases are selected at the same position regardless of the coefficient. Therefore, we design a more efficient strategy to implement the convolution:

$$\sum_{j=1}^m G_{ij}^t \text{Conv}(\mathbf{X}_{in(i)}, \mathbf{B}_j). \quad (8)$$

This transform means that all the bases can be firstly convoluted with the input and get a set of the feature maps, then we can compute the results as the linear combination of these feature maps. This property is easy to be proved based on the law of commutation. In this way, the repeated time-consuming convolution operations can be eliminated so as to reduce computational cost.

During forward propagation, in order to get the binary bases and make it easier to back propagate the gradients, we introduce a set of full-precision intermediate bases $\{\mathbf{A}_1, \mathbf{A}_2, \dots, \mathbf{A}_m\} \in \mathbb{R}^{d \times d \times n}$. Then the binary bases $\{\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_m\} \in \mathbb{R}^{d \times d \times n}$ are obtained through sign function, that is:

$$b = \text{sign}(a) = \begin{cases} 1 & \text{if } a \geq 0, \\ -1 & \text{otherwise.} \end{cases} \quad (9)$$

where b and a are the elements in $\mathbf{B}$ and $\mathbf{A}$. Besides, the $\epsilon - L_1$ ball projection in Algorithm 1 is applied on each column of G so that we can obtain the sparse coefficient.

In backward propagation, the strategy of STE [2] is applied since the gradient of the $\text{sign}()$ function in Equation (9) is almost 0 everywhere that is obviously not conducive to backward propagation. The core idea of STE is mapping the original continuous float parameters to $\{-1, 1\}$ to calculate the output of the network in forward propagation, and then directly updating the original float parameters instead of the binary ones in backward propagation. Let $\mathcal{L}(\cdot)$ still denotes the loss function, the process of calculating the gradient under STE can be formulated as:

$$\frac{\partial \mathcal{L}(a)}{\partial a} = \frac{\partial \mathcal{L}(b)}{\partial b} \frac{\partial b}{\partial a} = \frac{\partial \mathcal{L}(b)}{\partial b} 1_{|a| \leq 1}. \quad (10)$$

For example, the gradient of the set of binary bases is:

$$\frac{\partial \mathcal{L}(\mathbf{A}_i)}{\partial \mathbf{A}_i} = \frac{\partial \mathcal{L}(\mathbf{B}_i)}{\partial \mathbf{B}_i} \frac{\partial \mathbf{B}_i}{\partial \mathbf{A}_i} = \frac{\partial \mathcal{L}(\mathbf{B}_i)}{\partial \mathbf{B}_i}. \quad (11)$$

After obtaining the gradient, we can then update the parameters. In this process, the magnitude of some values in full-precision weight may be beyond binary range, so a clip operation is necessary to eliminate this effect. The whole process for training in the proposed method is summarized in Algorithm 2.

4.4. Time Complexity

The time complexity of the standard convolution operation in a given layer is $O(d^2 \cdot w_{out} \cdot h_{out} \cdot c_{in} \cdot c_{out})$. Instead, the proposed method contains two stage: compute m feature maps where the complexity is $O(d^2 \cdot c_{in} \cdot w_{out} \cdot h_{out} \cdot m)$; then construct new feature map through linear combination and stacking with complexity of $O(q \cdot w_{out} \cdot h_{out} \cdot c_{out} \cdot m \cdot (1 - s))$, where s represents the sparsity of coefficient.

5. Experimental Results

This section demonstrates the evaluation of the proposed method. Experiments are conducted on CIFAR-10, CIFAR-100 [20], Tiny ImageNet [10] and ImageNet-2012 [5]. We first explore the effect of two hyperparameters and compare the accuracy and compression ratio with state-of-the-art binary and decomposition methods, and then analyze the impact of coefficient sparsity.

5.1. Experiment Setting

For comparison, we select binary neural network including BC [3], BNN [4], BWN, XNOR [27] and filter decomposition methods such as PENNI [23] and ABC-Net [24]. SLBF [21] and LegoNet [34] are previous compression methods that stack single binary and full-precision filter respectively. "Our-BLC" denotes the proposed method of linear combination with low-dimensional binary bases. We report several metrics including the accuracy, compression ratio and coefficient sparsity, where the ratio is calculated according to the bit number of parameters. We focus on the convolutional layers for the compression since the parameters of fully-connected layers can be further compressed using other methods like SVD.

Network Structure. We choose VGG-16 [31] and ResNet-18 [11] as our fundamental structure for all comparison methods. In general, VGG-16 contains 13 convolutional layers and 3 fully-connected layers, while the ResNet-18 model includes 18 convolutional layers followed by a fully-connected layer. In our experiments, we simplify the two structures by replacing the fully-connection layers with average pooling layers. All the convolution filters except for the first layer is constructed using the proposed method, i.e. linear combination of low-dimensional binary bases with sparse coefficients. The networks are trained 500 epochs from scratch with initial learning rate of 0.01 that is adjusted with the cosine annealing. In addition, the batch normalization layers with scaling and shifting are applied and Adam [19] with 0.0005 weight decay is used as the optimizer. The parameters for the competitors are the suggested setting as described in the related papers.

Dataset. 1) *CIFAR-10 and CIFAR-100.* The CIFAR-10 dataset consists of 60,000 32×32 colorful images in 10 classes, and each class has 6,000 images with 50,000 train-

Method		Compression Ratio	Sparsity	CIFAR-10 Acc (%)	CIFAR-100 Acc (%)
Full ResNet-18 Net		1×	0	95.19	77.11
BC [3]		31.8×	N.A	93.73	71.15
BWN [27]		31.8×	N.A	93.97	72.92
BNN [4]		31.8×	N.A	90.47	70.34
XNOR-Net [27]		31.8×	N.A	90.14	72.87
ABC-Net [24]		5.4×	N.A	90.64	-
PENNI [23]		32.7×	0.979	94.01	70.90
SLBF [21]		58.7×	N.A	93.82	74.59
LegoNet [34]		17.5 ×	N.A	93.55	72.67
Ours-BLC					
$q = 1$	$k = 0.5$	35.9 ×	0.947	94.10	74.61
	$k = 2$	10.9×	0.968	94.39	76.48
$q = 2$	$k = 0.3$	38.7×	0.924	93.63	73.02
	$k = 0.4$	42.9×	0.954	93.07	73.13
	$k = 0.5$	26.0×	0.934	94.08	73.85
Ablation Study					
$q = 1$	$k = 2$	2.1×	0.557	94.51	76.72
$q = 2$	$k = 2$	8.3×	0.952	94.25	76.59

Table 1. Compression Results on CIFAR-10 and CIFAR-100 using ResNet-18 Architecture.

Method		Compression Ratio	Sparsity	CIFAR-10 Acc (%)	CIFAR-100 Acc (%)
Full VGG-16 Net		1×	N.A	93.25	73.55
BC [3]		31.8×	N.A	92.11	70.64
BWN [27]		31.8×	N.A	93.09	69.03
BNN [4]		31.8×	N.A	91.21	67.88
XNOR-Net [27]		31.8×	N.A	90.02	68.63
ABC-Net [24]		6.3×	N.A	89.69	-
PENNI [23]		108.9×	0.970	93.12	64.42
SLBF [21]		60.1×	N.A	91.44	68.80
LegoNet [34]		5.4×	N.A	91.35	70.10
Ours-BLC					
$q = 1$	$k = 0.5$	40.0×	0.949	90.79	66.99
$q = 2$	$k = 0.4$	44.5 ×	0.929	91.02	67.04
	$k = 0.5$	40.4×	0.954	90.99	68.14
	$k = 2$	11.5×	0.962	92.05	69.75
Ablation Study					
$q = 1$	$k = 2$	2.4×	0.528	91.85	68.70
	$k = 2$	9.8×	0.946	91.72	69.34
$q = 2$	$k = 0.5$	5.4×	0.522	91.11	68.08
	$k = 2$	1.4×	0.528	92.29	69.88

Table 2. Compression Results on CIFAR-10 and CIFAR-100 using VGG-16 Architecture.

ing images and remained 10,000 for testing. The difference of CIFAR-100 is that it contains 100 classes.

2) *ImageNet-2012*. ImageNet-2012 is a widely-used dataset with 1000 different classes. It has around 1.2M colored images for training and 50k images for validation where the image size is various.

3) *Tiny-ImageNet*. Tiny-ImageNet is actually a subset of ImageNet. It contains 200 classes where each class has 500 photos for training, 50 for validation and 50 for testing, with lower resolution of 64.

Hyperparameters. $q = \frac{c_{in}}{n}$ represents that the original filter is split into q blocks, and $k = \frac{m}{c_{out}}$ means the ratio

between the amount of binary bases and output channels. We set different values of these two parameters to assess the tradeoff between accuracy and compression ratio. The impact of different coefficient sparsity will also be explored in following section.

5.2. Comparison with the State-of-the-Art Methods

5.2.1 Binary Neural Networks.

In Table 1 and Table 2, we compare the performance of different binary neural networks. When applying ResNet-18, the proposed method achieves better accuracy than other competitors and the improvement is encouraging. We im-

Network	Compression Ratio	Sparsity	Top-1 (%)
Full VGG-16 Net	1×	0	51.85
BC [3]	31.8×	N.A	47.11
BWN [27]	31.8×	N.A	50.50
BNN [4]	31.8×	N.A	47.05
XNOR-Net [27]	31.8×	N.A	49.43
Ours-BLC			
$q = 1$	$k = 0.5$	0.932	48.98
	$k = 2$	0.955	50.87
$q = 2$	$k = 0.4$	0.930	48.01
	$k = 0.5$	0.937	49.61
	$k = 2$	0.946	52.03

Table 3. Compression Results on Tiny-ImageNet using VGG-16 Architecture.

prove the accuracy of binary neural network by around 3% on CIFAR-10 dataset and up to 6% on CIFAR-100, which is even close to full-precision network. For example, the network can be compressed by more than 35 times with comparable accuracy of 74.61% on CIFAR-100. On VGG-16, the strength of our proposed method is explicit on complex CIFAR-100 dataset. For instance, we get a much higher accuracy of 69.75% than ABC-Net [24] with much less memory because the compression ratio can be improved from 6.3 to 11.5 times. On Tiny-ImageNet in Table 3, the proposed method achieves larger accuracy than BC [3] and BWN [27] that both binarize weights under the same compression level of around 32 times. Compared with other binary networks, the proposed method can overcome the limitation and compress the network by over 40 times where the accuracy loss is still acceptable. It is clear that the proposed method outperforms binary networks in terms of accuracy, and we can get a much higher compression ratio with comparable accuracy under some settings.

5.2.2 Filter Decomposition and Compact Model.

The recent work PENNI [23] is a compression method combining network pruning and tensor decomposition, containing four steps called decomposition, retain, pruning and shrinking. Since the result on CIFAR-100 has not been reported in their paper, we implement it as the suggested setting without using pre-trained model. When controlling the coefficient sparsity in our method at the same level as PENNI, that is around 97%, the proposed method can achieve much better performance both on accuracy and compression ratio on ResNet-18. Although we use much more bases than PENNI, the compression ratio is still higher since the values in these bases are binarized and more bases may provide more useful information for learning features to get high accuracy. With respect to compact models like SLBF [21] and LegoNet [34], the accuracy gap compared to fully network is also narrowed. We reduce the accuracy

Network	Compression Ratio	Top-1 (%)	Top-5(%)
Full ResNet-18 Net	1×	69.3	89.2
BWN [27]	31.8×	60.8	83.0
BNN [4]	31.8×	42.2	67.1
XNOR-Net [27]	31.8×	51.2	73.2
ABC-Net [24]	5.4×	65.0	85.9
Ours-BLC			
$q = 2$	$k = 0.4$	42.9×	61.5
	$k = 0.5$	26.0×	62.8
	$k = 2$	8.3×	65.3

Table 4. Compression Results on ImageNet-2012 using ResNet-18 Architecture.

loss to 1% on CIFAR-10 under ResNet-18 with 35.9 times compression. On the whole, the results show the effectiveness of the proposed method on increasing accuracy and compression ratio.

5.3. Ablation Study

5.3.1 The Influence of Hyperparameters q and m .

Considering the accuracy loss that may cause by large compression ratio, we only evaluate the number of blocks $q = \{1, 2\}$. To control the variables and explore the effect of hyperparameters $q = \frac{c_{in}}{n}$ and $k = \frac{m}{c_{out}}$, we fix the sparsity between 0.92 and 0.97. Based on the theoretical compression ratio described in Equation (6), larger k (i.e. the ratio between m and c_{out}) will lead to a lower compression ratio, but the impact of q is not certain because it depends on other values such as kernel size and number of channels. We analyze the performance under different settings of hyperparameters q and k , and the experiment results are also consistent with this principle.

CIFAR-10 and CIFAR-100 Results. We tune q in $\{1, 2\}$ and k in $\{0.4, 0.5, 2\}$ for VGG-16 while $\{0.3, 0.4, 0.5, 2\}$ for ResNet-18. The results of ablation study using ResNet-18 and VGG-16 are shown in Tables 1 and 2 respectively. On VGG-16 network, we can obtain an accuracy of 91.72% on CIFAR-10 and 69.34% on CIFAR-100 with 9.8 times compression when setting $q = 1$, $k = 2$ and sparsity as 0.946. The compression ratio will increase to 40 times when k reduces to 0.5 but it will lead to larger accuracy loss. For ResNet-18 in Table 1, when setting $k = 2$ and $q = 0.5$, we can compress the model by 26 times with 3.26% loss compared with fully network on CIFAR-100. The loss can be further narrowed by applying larger $k = 2$, that is, using more binary bases. It is obvious that the network will suffer an accuracy loss if compressed at a high ratio. Therefore, it is essential to consider a tradeoff between accuracy and compression ratio when selecting the hyperparameters in specific tasks. On CIFAR-100 dataset, we also demonstrate the convergence of the proposed method. The convergence curve is shown in Figure 2, which means that the loss

Figure 2. Train Loss of Each Iteration.

Figure 3. Relationship Between Sparsity and Accuracy.

or called objective function value of the proposed method can reach an optimal status after a certain number of training iterations.

Tiny-ImageNet Results. Due to the limitation of computation resources, we select Tiny-ImageNet dataset for evaluation. The VGG-16 network with the same structure as CIFAR experiments is applied, and all the results are reported with top-1 accuracy as summarized in Table 3. It is encouraging that we can obtain an accuracy close to full-precision network with a relative large compression ratio. The network can be compressed by 35.6 times with 48.98% accuracy where the loss of 2.87% is acceptable. In terms of the hyperparameter, using more binary bases will also lead to better accuracy on this dataset. For example, when fixing $q = 2$, the accuracy of $k = 0.4$ is 48.01 % and it can be improved to 49.61% if k increase to 0.5. Compared with fully network, we can obtain a significant accuracy of 52.03% with 9.1 times compression. These results exactly demonstrate the power of the proposed method on compressing CNNs.

ImageNet Results. Table 4 provides the results on ImageNet with ResNet-18 structure. The compression ratio and accuracy is evaluated under three different k values with $q = 2$. We can compress the model by around 26 times with only 3% loss on top-5 accuracy. When reducing the compression ratio to 8.3 times through adjusting k value, the accuracy between the proposed method and full-precision network will be greatly narrowed. Compared to binary neural networks, the proposed method still has better performance on large scale dataset. For example, a larger compression ratio of 42.9 times can be obtained while achieving 1.9% improvement in terms of the top-5 accuracy and 0.7% improvement on top-1 accuracy compared with BWN [27].

5.3.2 Impact of Sparsity.

Tables 1 and 2 also contain some results under different sparsity. For example, on CIFAR-10 using ResNet-18, the accuracy reduces from 94.51% to 94.39% as the compression ratio increases from 2.1 to 10.9 times, where the sparsity changes from 0.557 to 0.968 through adjusting the parameters in Algorithm 1. The same trend can also be verified on VGG-16 such as the setting of $q = 2$ and $k = 2$.

As shown in the results, higher sparsity will lead to slightly lower accuracy while the compression ratio will greatly increase, where the accuracy loss is around 0.1%-0.3%. Thus, in order to obtain a good tradeoff between accuracy and compression ratio, coefficient with higher sparsity may be a good choice.

To analyze the impact of coefficient sparsity in detail, we fix $q = 2$ and $k = 0.5$ on CIFAR-100 using VGG-16. Figure 3 depicts the relationship between sparsity and accuracy. As the sparsity increases from 0 to around 1, the accuracy will start to decrease after reaching a maximum. This trend means that coefficients that are too sparse or too dense both lead to terrible performance. Though it is actually difficult to find the exact maximum and the corresponding sparsity, we can still get a proper setting according to specific demands such as lower memory cost or better accuracy.

6. Conclusion

In this paper, we have proposed a novel method for compressing CNNs. Specifically, we have redesigned the convolution filters by stacking the linear combination of a set of low-dimensional binary bases with corresponding coefficients, on which the sparsity is explored to avoid overfitting using L1-ball projection algorithm. Detailed analysis about the memory cost and time complexity is also provided. In the experiments, we have compared the performance of our method with the state-of-the-art binary neural networks and filter decomposition methods. The results have shown that the proposed method can achieve higher compression ratio with comparable accuracy under the different settings. Further, the ratio can be adjusted through the hyperparameters and sparsity to meet up special requirements. That is, our method is flexible and can help get a good tradeoff between accuracy and compression. Hence, it provides a promising way to deploy CNNs on resource-constrained platforms.

References

- [1] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on Audio, Speech, and Language processing*, 22(10):1533–1545, 2014. 1
- [2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013. 2, 5
- [3] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *arXiv preprint arXiv:1511.00363*, 2015. 2, 5, 6, 7
- [4] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016. 2, 5, 6, 7
- [5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 5
- [6] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532, 2020. 2
- [7] Misha Denil, Babak Shakibi, Laurent Dinh, Marc’Aurelio Ranzato, and Nando De Freitas. Predicting parameters in deep learning. *arXiv preprint arXiv:1306.0543*, 2013. 1
- [8] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 1
- [9] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, 2015. 4
- [10] Lucas Hansen. Tiny imagenet challenge submission. *CS 231N*, 2015. 5
- [11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 1, 5
- [12] 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. 1
- [13] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 1
- [14] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018. 1
- [15] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014. 1
- [16] Xiao Jin, Baoyun Peng, Yichao Wu, Yu Liu, Jiaheng Liu, Ding Liang, Junjie Yan, and Xiaolin Hu. Knowledge distillation via route constrained optimization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1345–1354, 2019. 1
- [17] Ryuji Kamiya, Takayoshi Yamashita, Mitsuru Ambai, Ikuro Sato, Yuji Yamauchi, and Hironobu Fujiyoshi. Binary-decomposed dcnn for accelerating computation and compressing model without retraining. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 1095–1102, 2017. 2
- [18] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015. 1, 2
- [19] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5
- [20] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 5
- [21] Weichao Lan and Liang Lan. Compressing deep convolutional neural networks by stacking low-dimensional binary convolution filters. *arXiv preprint arXiv:2010.02778*, 2020. 5, 6, 7
- [22] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014. 1
- [23] Shiyu Li, Edward Hanson, Hai Li, and Yiran Chen. Penni: Pruned kernel sharing for efficient cnn inference. In *International Conference on Machine Learning*, pages 5863–5873, 2020. 2, 5, 6, 7
- [24] Xiaofan Lin, Cong Zhao, and Wei Pan. Towards accurate binary convolutional neural network. *arXiv preprint arXiv:1711.11294*, 2017. 2, 5, 6, 7
- [25] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l0 regularization. *arXiv preprint arXiv:1712.01312*, 2017. 4
- [26] You Qiaoben, Zheng Wang, Jianguo Li, Yinpeng Dong, Yungang Jiang, and Jun Zhu. Composite binary decomposition networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4747–4754, 2019. 1
- [27] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Proceedings of European Conference on Computer Vision*, pages 525–542. Springer, 2016. 2, 5, 6, 7, 8
- [28] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015. 1

- [29] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014. [1](#)
- [30] David Sculley. Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web*, pages 1177–1178, 2010. [4](#)
- [31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. [5](#)
- [32] Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017. [1](#)
- [33] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. Cars: Continuous evolution for efficient neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1829–1838, 2020. [1](#)
- [34] Zhaohui Yang, Yunhe Wang, Chuanjian Liu, Hanting Chen, Chunjing Xu, Boxin Shi, Chao Xu, and Chang Xu. Legonet: Efficient convolutional neural networks with lego filters. In *International Conference on Machine Learning*, pages 7005–7014, 2019. [5](#), [6](#), [7](#)
- [35] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*, 2017. [1](#)
- [36] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 38(10):1943–1955, 2015. [1](#), [2](#), [3](#)
- [37] Wenyu Zhao, Teli Ma, Xuan Gong, Baochang Zhang, and David Doermann. A review of recent advances of binary neural networks for edge computing. *IEEE Journal on Miniaturization for Air and Space Systems*, 2020. [2](#)