Convolution Filter Compression via Sparse Linear Combinations of Quantized Basis

Weichao Lan, Yiu-Ming Cheung, *Fellow, IEEE*, Liang Lan, Juyong Jiang, and Zhikai Hu, *Member, IEEE*

***Abstract*— Convolutional neural networks (CNNs) have achieved significant performance on various real-life tasks. However, the large number of parameters in convolutional layers requires huge storage and computation resources, making it challenging to deploy CNNs on memory-constrained embedded devices. In this article, we propose a novel compression method that generates the convolution filters in each layer using a set of learnable low-dimensional quantized filter bases. The proposed method reconstructs the convolution filters by stacking the linear combinations of these filter bases. By using quantized values in weights, the compact filters can be represented using fewer bits so that the network can be highly compressed. Furthermore, we explore the sparsity of coefficients through L_1-ball projection when conducting linear combination to further reduce the storage consumption and prevent overfitting. We also provide a detailed analysis of the compression performance of the proposed method. Evaluations of image classification and object detection tasks using various network structures demonstrate that the proposed method achieves a higher compression ratio with comparable accuracy compared with the existing state-of-the-art filter decomposition and network quantization methods.**

***Index Terms*— Filter decomposition, network compression, quantization.**

NOMENCALUTURE

$\{(\mathbf{x}, \mathbf{y})\}$	Set of data pairs.
c_{in}, c_{out}	Number of input and output channel for a given convolutional layer.
d	Width/height of a convolution kernel.
n	Length of filter bases, where $n \ll c_{in}$.
$\mathbf{X}_{in}, \mathbf{X}_{out}$	Input and output tensor for a given convolutional layer, respectively, where $\mathbf{X}_{in} \in \mathbb{R}^{w_{in} \times h_{in} \times c_{in}}$ and $\mathbf{X}_{out} \in \mathbb{R}^{w_{out} \times h_{out} \times c_{out}}$.
$\mathbf{W}$	Weight tensor for a given convolutional layer, where $\mathbf{W} \in \mathbb{R}^{d \times d \times c_{in} \times c_{out}}$.
$\mathbf{W}^t$	tth convolution filter, where $\mathbf{W}^t \in \mathbb{R}^{d \times d \times c_{in}}$.
q	Number of blocks that a convolution filter is split into, where $q = \frac{c_{in}}{n}$.
$\{\mathbf{A}_1, \ldots, \mathbf{A}_m\}$	Set of m filter bases with **full-precision** values, where $\{\mathbf{A}_1, \ldots, \mathbf{A}_m\} \in \mathbb{R}^{d \times d \times n}$.
$\{\mathbf{B}_1, \ldots, \mathbf{B}_m\}$	Set of m filter bases with **binarized** values, i.e., the binarized version of $\{\mathbf{A}_1, \ldots, \mathbf{A}_m\}$.
$\{\mathbf{I}_1, \ldots, \mathbf{I}_m\}$	Set of m filter bases with **integer** values, i.e., the multibit version of $\{\mathbf{A}_1, \ldots, \mathbf{A}_m\}$.
G	Sparse coefficient, where $G \in \mathbb{R}^{m \times \frac{c_{in}}{n} \times c_{out}}$.

Received 7 April 2022; revised 20 June 2023, 11 November 2023, 26 February 2024, and 14 June 2024; accepted 8 September 2024. Date of publication 24 September 2024; date of current version 4 June 2025. This work was supported in part by NSFC/Research Grants Council (RGC) Joint Research Scheme under Grant N_HKBU214/21; in part by the General Research Fund of RGC under Grant 12201321, Grant 12202622, and Grant 12201323; in part by RGC Senior Research Fellow Scheme under Grant SRFS2324-2S02; and in part by NSFC under Grant 61906161. *(Corresponding author: Yiu-Ming Cheung.)*

Weichao Lan, Yiu-Ming Cheung, and Zhikai Hu are with the Department of Computer Science, Hong Kong Baptist University, Hong Kong, SAR, China (e-mail: cswclan@comp.hkbu.edu.hk; ymc@comp.hkbu.edu.hk; cszkhu@comp.hkbu.edu.hk).

Liang Lan is with the Department of Interactive Media, Hong Kong Baptist University, Hong Kong, SAR, China (e-mail: lanliang@hkbu.edu.hk).

Juyong Jiang is with Hong Kong University of Science and Technology (Guangzhou), Guangzhou 511458, China (e-mail: jjiang472@connect.hkust-gz.edu.cn).

Digital Object Identifier 10.1109/TNNLS.2024.3457943

I. INTRODUCTION

CURRENTLY, deep convolutional neural networks (CNNs) have become one of the most popular tools in various fields, such as computer vision [1], [2], [3], natural language process [4], [5], and speech recognition [6], due to their effectiveness and remarkable performance. However, a CNN model with good performance usually consists of a large number of parameters, which requires substantial storage space and time-consuming computation. A typical example is the ResNet-50 with 50 layers, which requires over 95 MB of memory to store the model and 3.8 billion floating-point operations (FLOPs) for a single prediction. Although powerful GPUs and CPUs can help deal with computationally complex tasks when training the network, the significant resource consumption still limits the applications of deep CNNs on resource-constrained devices such as mobile phones, smart watches, and other edge devices at the stage of inference. Therefore, to make deep CNNs deployable on resource-constrained devices, it is desired that the number of parameters in CNNs can be reduced significantly through compression.

In recent years, various model compression methods have been proposed, demonstrating that CNN models can still function normally while the memory and computing time will be saved significantly after discarding redundancy. These methods can be roughly categorized into five groups: 1) network pruning [7], [8], [9]: removing redundant components such as channels and filters; 2) network quantization [10], [11]: using the low-bit representation of parameters; 3) filter decomposition [12], [13]: approximating the original filter with lightweight kernels; 4) knowledge distillation [14]: training shallower models based on larger and deeper models; and 5) compact model design [15], [16], [17]: directly designing a more compact network with special operations such as spatial convolution and shift convolution. Typically, the compact

networks can be further pruned or optimized based on actual requirements. Recently, redesigned lightweight models have attracted more attention due to their impressive success in various tasks. These methods aim to establish 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) [18].

Among the various compression methods, network quantization explores the redundancy in representation to reduce the number of bits in parameters, achieving considerable compression. One promising approach is to binarize the weights of convolution filters [10], [19], which has been theoretically and practically proved to be effective for simplifying the process of convolution. However, this approach often suffers from a serious accuracy loss compared to full-precision networks due to the rough estimate, making it challenging to apply to complex and large-scale tasks. Some studies have proposed to improve the performance of binary neural networks by taking the quantization loss and filter loss into account [20], [21]. Another concern is that a single binarized filter is inadequate to represent the filters with real-values, which motivates us to explore the efficient combinations of more binarized filters.

In parallel with network quantization, filter decomposition focuses on the redundancy in tensors and simplifies the networks by discarding unimportant kernel components, providing an optimal solution of a given weight matrix by minimizing the approximation error. Early low-rank approximation methods utilize mathematical operations such as SVD [22], Tucker decomposition [12], and CP-decomposition [23] that decompose the original filters into two to four smaller components. For example, [24] decomposes a 2-D $w \times h$ kernel into the multiplication of two small 1-D vectors with the size of $w \times 1$ and $1 \times h$, respectively. [12] considers the low-rank property of both the input channel and output channel, where a 2-D kernel is 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. In addition, the decomposition often relies on rank selection and involves multiple or complex decompositions that are time-consuming.

In this article, we therefore propose a new method to generate quantized convolution filters by stacking sparse linear combinations of a set of low-dimensional quantized bases. Through quantized weight values, including binary and multi-bit quantization, we can then eliminate the redundancy in representation by reducing the bit number required to store parameters. To achieve a higher compression ratio, the entire weight tensor in a given convolutional layer is first split into several blocks. Each block is estimated as a linear combination of the quantized bases, and then the entire filter is reconstructed by stacking these blocks together. Moreover, the sparsity of coefficients is explored when conducting linear combination to further reduce storage consumption, which can also help avoid overfitting by reducing the number of parameters in the networks [25], [26], [27]. We also develop an effective algorithm to jointly optimize the quantized bases and sparse linear coefficients. Without using complex decomposition operations, the quantized bases in the proposed method will be learned through training, breaking the compression limitation while alleviating the above issues of filter decomposition. Due to the sparsity of the linear combination coefficients and the low memory cost of quantized 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 article are summarized as follows.

1) We propose a novel method to compress convolution filters by stacking the sparse linear combinations of a set of learnable low-dimensional quantized filter bases. The proposed method is flexible and can be easily generalized to meet different compression requirements by adjusting the dimensions of filter bases and the sparsity of coefficients.
2) We develop an effective algorithm to jointly learn and optimize the sparse coefficients of linear combinations and quantized filter bases.
3) We conduct experiments on image classification and object detection tasks, using various network structures on five 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.

II. Related Work

A. Network Quantization

Binary neural networks, whose associated parameters are binary, can greatly reduce the memory cost since they can be stored in a low-bit format. In addition, complex computations such as multiplication can be substituted with cheap operations like addition and subtraction. BinaryConnect (BC) [19] first considers binary weights which limits the values in weight tensors as only 1 or -1. Binarized neural network (BNN) [10] extends the idea of BC by introducing binarized activations. Further work in [28] 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 the training process according to STE strategy [29]. Despite the advantages of binary neural networks on network acceleration and compression, the accuracy loss compared to full-precision networks is still large, making it a challenge to enhance the performance of binary networks in many tasks.

To improve the accuracy of classical binary neural networks, different strategies for quantized algorithms have been explored. For example, Li et al. [30] proposed a ternary weight network (TWN) and obtained the ternary weight with the corresponding scaling factory by minimizing the weight fitting error. Mix-precision quantization methods [11], [31], [32] are also applied to fine-tune the quantized network for efficient allocation of computation resources. Recently,

effective optimization algorithms have been popular for binary neural networks [33], [34]. Furthermore, MCN [20] recovers the unbinarized filters and optimizes the network based on joint losses. Bi-Real Net [35] utilizes shortcuts to transfer the real-valued feature maps before binarization, so that more information in the network can be preserved and the representational ability will also be improved. DSQ [36] uses the three-segment tanh function to fit the quantized function, which solves the problem that the quantized function is not derivable. In addition, Ding et al. [37] introduced distribution loss to adjust regularization and control the activation distributions. For multibit quantization, it has been proven to be effective to quantize the weights and activations to multibit fixed-point numbers such as the Ristretto framework [38], which conducts statistical analysis on the distribution of parameters for quantization. LQ-Nets [39] explores nonuniform quantization, which uses a set of floating-point values as the basis to represent the quantized values, and learns the basis by minimizing the quantization error. Wang et al. [40] binarized the activations to 0 or $+1$, exploring the sparsity of feature representations. The recent work [41] combined network quantization with NAS to explore bit-level sparsity. Although the performance of these quantized networks is encouraging, the expression ability will be degraded due to the weight and activation quantization. Therefore, it is inspired to increase filter variations to compensate for accuracy degradation and obtain better representation [42].

B. Filter Decomposition

In a general neural network, tensor operations occupy most of the computation cost. Subsequently, decomposition or approximation of tensors has been a favorable method for compression and acceleration, which uses several low-rank matrices to approximate the original full-rank weights. Zhang et al. [43] assumed a low-rank property of the output feature to estimate the original weight with the production of two low-rank matrices. PENNI [44] conducts SVD on the weight tensors to obtain a small number of bases with lower dimensions. Binary weighted decomposition (BWD) [45] 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 [46] further approximates full-precision activations with the linear combination of some preset binary activation bases. Compared to XNOR-Net, ABC-Net achieves better results by applying multiple binary weights and activations, but the complexity also increases at the same time [47].

Some works have also explored the concept of the linear combination of convolutional filters [25], [48]. Correlative filters are generated and combined in [49] to optimize the structure of CNN. Octave convolution [50] decomposes the feature maps to save memory and computing resources, and expands the receptive field with better recognition effect through special processing of low-frequency information. Basis Filter [48] proposes to learn a set of filter basis and combine the generated activations to achieve lower complexity. In contrast, the proposed method stacks linear combinations of filter bases themselves rather than combining activations. Furthermore, the basis filter trains the basis based on the original weight and jointly minimizes the approximation error, leading to cumulative memory and computational cost. In contrast, the proposed method is completely independent of original weights that can be trained from scratch without supplementary requirements. Besides, we also binarize the values in filter bases for a higher compression ratio.

C. Compact Models

Designing compact models with fewer parameters and FLOPs has attracted more attention recently. MobileNet [15] applies depthwise convolution and pointwise convolution to improve efficiency. The variants MobileNetV2 [51] introduces residual and linear block, and MobileNetV3 [52] combines compression with NAS [18] for better performance. ShuffleNet [53] utilizes channel shuffle and group convolution to transfer information among different groups. EspNetv2 [16] introduces group point-wise and dilated separable convolution operations, increasing the receptive field of convolution kernels. In addition to cheaper convolution operations, the shift operation [54] has also been applied to build compact networks. Jeon and Kim [55] designed an active shift layer (ASL) to save memory. Then, the sparse shift layer (SSL) in FE-Net [56] dismisses the senseless shift operation for acceleration. MnasNet [57] treats multiple metrics as optimization objectives and achieves better performance than MobileNetV2. Although the recent lightweight models have successfully saved much memory and computation cost, the remaining 1×1 point-wise convolution still contains redundancy that can be further reduced [58].

III. Preliminaries

This section provides a brief introduction to convolution filters. The main notations are listed in Nomenclature.

Convolution Filter: In general, a convolutional layer in a CNN transforms a 3-D 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 $\mathbf{X}_{\text{in}} \in \mathbb{R}^{w_{\text{in}} \times h_{\text{in}} \times c_{\text{in}}}$ and $\mathbf{X}_{\text{out}} \in \mathbb{R}^{w_{\text{out}} \times h_{\text{out}} \times c_{\text{out}}}$, respectively. Then, the weight tensor can be written as a 4-D tensor $\mathbf{W} \in \mathbb{R}^{w \times h \times c_{\text{in}} \times c_{\text{out}}}$, where the four dimensions refer to the width, height of the convolution kernel and the number of input and output channels. Typically, the kernel is square-shaped, where the width w and height h are set at the same value d. Let "Conv" denote the convolution operation. Then, the output of a convolutional layer can be calculated as

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

where $\mathbf{b}$ is the bias and $\sigma(\cdot)$ represents the activation function usually chosen to be Rectified Linear Unit (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

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

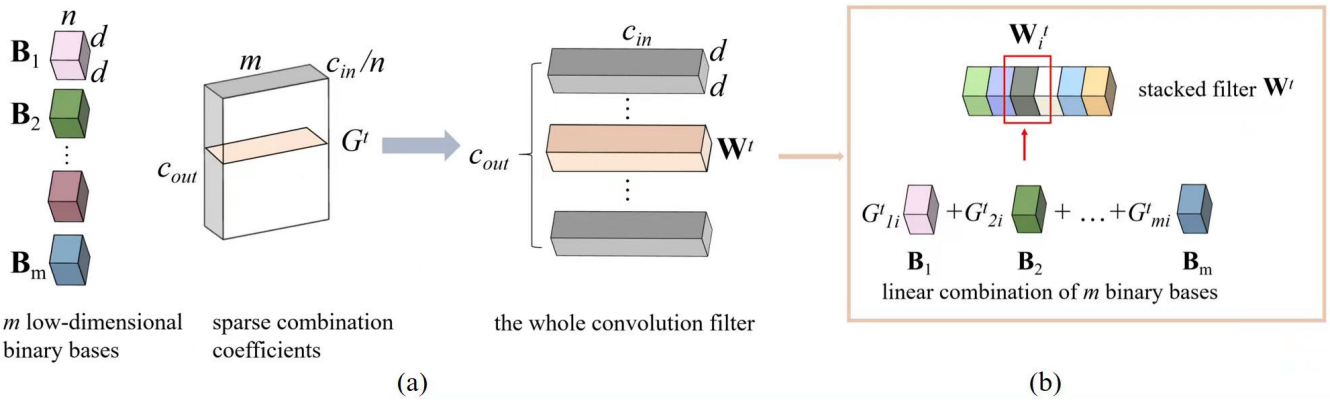


Fig. 1. Proposed method to approximate convolution filters. The convolution filters are reproduced by the linear combinations of a set of low-dimensional quantized filters $\{\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_m\}$ with sparse coefficients G. (a) Convolution filters approximation for a given layer. Each column G^t in G is a coefficient vector to generate the tth convolution filter $\mathbf{W}^t$. (b) Detailed information for approximating. The tth filter $\mathbf{W}^t$ is first split into c_{in}/n blocks, where each block is a linear combination of $\{\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_m\}$.

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 to achieve good performance.

IV. Methodology

A. Convolution Filters Approximation

In the commonly used filter decomposition methods for compressing CNNs, the original large weight matrix is represented as the multiplication of several low-rank matrices [43]. However, these methods still use full-precision bases and fixed rank, which limit the compression performance. To overcome these limitations, we propose a new kind of quantized convolution filter, which contains multiple low-dimensional quantized bases and sparse coefficients to approximate the original convolution filters. Instead of decomposing the original weight tensor, which is complex and laborious, the proposed method generates the filter bases and coefficients randomly at the beginning and learns them directly during training. Moreover, the dimensions of quantized bases in the proposed method can be adjusted in advance according to the specific requirements of the compression ratio. For further compression, we also propose to first split the whole weight tensor into several blocks and then apply the stacking technique.

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

$$\mathbf{W}_i^t = G_{1i}^t\mathbf{B}_1 + G_{2i}^t\mathbf{B}_2 + \cdots + G_{mi}^t\mathbf{B}_m = \sum_{j=1}^{m} G_{ji}^t\mathbf{B}_j \tag{3}$$

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

$$\begin{aligned}\mathbf{W}^t &= \left[\mathbf{W}_1^t, \mathbf{W}_2^t, \ldots, \mathbf{W}_q^t\right] \\ &= \left[\sum_{j=1}^{m} G_{j1}^t\mathbf{B}_j, \sum_{j=1}^{m} G_{j2}^t\mathbf{B}_j, \ldots, \sum_{j=1}^{m} G_{jq}^t\mathbf{B}_j\right].\end{aligned} \tag{4}$$

The proposed idea concatenates q low-dimensional filter blocks with the depth n to form a high-dimensional filter with depth c_{in}. Each low-dimensional filter block is a linear combination of m binary bases, which are shared across all convolution filters for a given layer. Therefore, in a given convolution layer, we only need to store the binary bases $\{\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_m\}$ 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 nonzero values and their corresponding coordinates. Then, the memory cost for storing G using COO is $nnz(G) \times 4 \times 32$ bits, where $nnz(G)$ denotes the number of nonzeros values in G. Compared with the original weight tensor $\mathbf{W} \in \mathbb{R}^{d\times d\times c_{\text{in}}\times c_{\text{out}}}$ whose memory cost is $d \times d \times c_{\text{in}} \times c_{\text{out}} \times 32$ bits, the compression ratio of proposed method can be computed as

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

Let s to denote the sparsity rate of the coefficient tensor G, we have $nnz(G) = m \times q \times c_{\text{out}} \times (1-s)$. Besides, we use k to denote a preset ratio between the number of binary bases

Algorithm 1 $\epsilon - L_1$ Ball Projection [59]

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

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

and output channels, that is, $k = (m/c_{\text{out}})$. Thus, (5) can be rewritten as

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

It is worth noting that the term $(k/(q \times 32))$ remains constant when the hyperparameters k and q are predetermined. Considering the presence of multiple convolutional layers in a CNN, each with varying input c_{in} or output channels c_{out}, and sparsity s, our approach to obtain the total compression ratio is to count the number of parameters of all binary weights and full-precision coefficients in the entire network. In an experiment, the coefficient sparsity of each layer is controlled at a similar level, we count the sparsity of each layer and calculate the average value to approximately estimate the number of nonzero parameters in coefficients. Then, the final compression ratio is computed as the ratio between the total bit number of parameters in the original and compressed model. For example, assume the total number of parameters in convolutional layers of the original network is N_o. In the compressed model, the number of binary weights and full-precision coefficients is N_b and N_c, respectively, with an average sparsity of s_a. Then, similar to (5), the final compression ratio of the compressed model is

$$\frac{N_o \times 32}{N_c \times (1 - s_a) \times 4 \times 32 + N_b}. \tag{7}$$

B. Sparsity of Coefficient

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

It has been well-known that the sparsity property cannot only save the storage and computational cost of the networks but also prevent overfitting [25], [26], [27]. For example, Liu et al. [60] designed sparse convolution kernels via decomposition to reduce the redundancy in weights. A learnable sparse transform (LST) is proposed in [61], which converts the features into a more sparser domain. NNCS [62] explores sparse representations of weight parameters in the transform domain. In general, achieving sparsity through regularization is one of the popular methods such as L_0 [63] and L_1 [64] regularization. In our proposed method, the sparse coefficients are obtained by $\epsilon - L_1$ ball projection [59]. 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 onto the L_1-ball after finding a proper threshold θ through 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.

Algorithm 2 Training Compressed CNN With Linear Combination of Stacked Small Binary Filters

Input: training data pairs $\{(\mathbf{x}, \mathbf{y})\}$, hyperparameter for $L1$ projection r (radius) and t (tolerance), parameter for binary filter q and m
Output: compressed network

Random Generate proxy variables $\{\mathbf{A}_1, \ldots, \mathbf{A}_m\}$ and coefficient G based on CNN configuration q and m
for iter = 1 to maxIter **do**
  Get a minibatch of training data $\{(\mathbf{x}, \mathbf{y})\}$
  for $l = 1$ to L **do**
    Get m binary filter bases $\{\mathbf{B}_1, \ldots, \mathbf{B}_m\}$ according to Eq. (10)
    Apply Algorithm 1 on each column $\mathbf{g}$ of coefficient G to achieve sparsity
  end for
  Conduct forward propagation
  Compute the loss based on Eq. (8)
  Conduct backward propagation based on Eq. (11)
  Clip the binarized weights in the range of $\{-1, 1\}$
  Update parameters using any popular optimizer
end for

C. Training Process

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

$$\min \sum_{t=1}^{c_{\text{out}}} \mathcal{L}\left(f\left(\sigma\left(\sum_{i=1}^{q} \text{Conv}\left(\mathbf{X}_{\text{in}(i)}, \sum_{j=1}^{m} G_{ij}^t \mathbf{B}_j\right)\right)\right), \mathbf{y}\right)$$
$$\text{s.t. } \mathbf{B}_j \in \{-1, 1\} \tag{8}$$

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. The drawback is that repeated convolution operations 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

$$\sum_{j=1}^{m} G_{ij}^{t}\text{Conv}(\mathbf{X}_{\text{in}(i)}, \mathbf{B}_j). \tag{9}$$

This transform means that all the bases can be first 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 prove based on the law of commutation. In this way, the repeated time-consuming convolution operations can be eliminated so as to reduce computational cost.

In this framework, the computation process for a given convolutional layer in the proposed method contains two stages: compute m feature maps, which requires $d^2 \cdot c_{\text{in}} \cdot w_{\text{out}} \cdot h_{\text{out}} \cdot m$ FLOPs; then construct new feature map through linear combination and stacking with $q \cdot w_{\text{out}} \cdot h_{\text{out}} \cdot c_{\text{out}} \cdot m \cdot (1-s)$ FLOPs, where s represents the sparsity of coefficient.

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 $\{\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_m\} \in \mathbb{R}^{d\times d\times n}$. Then, the binary bases $\{\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_m\} \in \mathbb{R}^{d\times d\times n}$ are obtained through sign function, that is,

$$b = \text{sign}(a) = \begin{cases} 1, & \text{if } a \geq 0 \\ -1, & \text{otherwise} \end{cases} \tag{10}$$

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 [29] is applied since the gradient of the sign$(\cdot)$ function in (10) is almost zero everywhere that is obviously not conducive to backward propagation. The core idea of STE is to map the original continuous float parameters to $\{-1, 1\}$ to calculate the output of the network in forward propagation, and then directly update the original float parameters instead of the binary ones in backward propagation. Let $\mathcal{L}(\cdot)$ still denote the loss function, the process of calculating the gradient under STE can be formulated as

$$\frac{\partial \mathcal{L}(a)}{\partial a} = \frac{\partial \mathcal{L}(b)}{\partial b}\frac{\partial b}{\partial a} = \frac{\partial \mathcal{L}(b)}{\partial b} 1_{|a|\leq 1}. \tag{11}$$

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

$$\frac{\partial \mathcal{L}(\mathbf{A}_i)}{\partial \mathbf{A}_i} = \frac{\partial \mathcal{L}(\mathbf{B}_i)}{\partial \mathbf{B}_i}\frac{\partial \mathbf{B}_i}{\partial \mathbf{A}_i} = \frac{\partial \mathcal{L}(\mathbf{B}_i)}{\partial \mathbf{B}_i}. \tag{12}$$

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 the 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.

V. Ternary and Multibit Quantization

A. Threshold-Based Ternary Quantization

In addition to 1 and -1 in binary neural networks, TWN [30] further introduces zero values to improve expressive ability. Although TWN has an additional "0" state, the computation will not increase since "0" does not require an extra multiplication operation. Similar to binary quantization where the threshold is set to 0, ternary quantization also tries to find a proper threshold. We compute the threshold based on an assumption that the values in weight tensors satisfy the normal or uniform distributions, as described in [30]. The threshold is then estimated as

$$\beta \approx 0.7 \times E(|\mathbf{W}|) = \frac{0.7}{n}\sum_{i=1}^{n}|\mathbf{W}_i|. \tag{13}$$

After obtaining the threshold β, the weight tensor can be quantized to ternary values as

$$f = \text{ternarize}(w) = \begin{cases} 1, & \text{if } w > \beta \\ 0, & \text{if } |w| \leq \beta \\ -1, & \text{if } w < -\beta. \end{cases} \tag{14}$$

B. Multibit Quantization

The weight values in BNN are quantized to $+1$ or -1, which can be represented in a single bit. In this way, the network will be compressed about 32.0×. However, binary quantization will lead to more information and accuracy loss compared to full precision. In contrast, fixed-point numbers reserve more information, where the processing logic is also simple and efficient. A common method is to reduce 32-bit numbers to fixed 8-bit numbers, which include only 2^8 possible values. In the simplest case, we can calculate the minimum and maximum weights of a layer and evenly divide the range between them into 255 intervals, thus the weight values can be set according to the closest edge. Considering the balance between compression ratio and performance, we apply a new function to quantize the low-dimensional convolution filters into multibit

$$\mathbf{I} = \text{integer}(\mathbf{A}) = \text{round}\left(\frac{\mathbf{A}}{\max(|\mathbf{A}|)} \times 2^b\right) \tag{15}$$

where $\mathbf{A}$ is still the original filter with full-precision and b donates the bit numbers we need to compress. Thus, the objective function for optimizing the indicator matrix $\mathbf{G}$ and the set of integer filter bases $\{\mathbf{I}_1, \ldots, \mathbf{I}_m\}$ will be changed to

$$\min \sum_{t=1}^{c_{\text{out}}} \mathcal{L}\left(f\left(\sigma\left(\sum_{i=1}^{q}\sum_{j=1}^{m} G_{ji}^{t}\text{Conv}\left(\mathbf{X}_{\text{in}(i)}, \mathbf{I}_j\right)\right)\right), \mathbf{y}\right)$$
$$\text{s.t. } \mathbf{I}_{ij} \in \{-2^b, 2^b\}. \tag{16}$$

During the training, the original low-dimensional filters are converted to integer filters based on (15) in forward propagation. Similar to binary methods, the STE strategy is also applicable to backward propagation. Then, the gradient of quantized bases in a loss function with integer values will be

$$\frac{\partial \mathcal{L}(\mathbf{A}_i)}{\partial \mathbf{A}_i} = \frac{\partial \mathcal{L}(\mathbf{I}_i)}{\partial \mathbf{I}_i}\frac{\partial \mathbf{I}_i}{\partial \mathbf{A}_i} = \frac{\partial \mathcal{L}(\mathbf{I}_i)}{\partial \mathbf{I}_i}. \tag{17}$$

The training of ternary and multibit networks is similar to binary models, where quantization is applied in the forward and backward process while continuous weights are still used when updating parameters. This is also a common technique in training quantized networks.

VI. Experimental Results

This section demonstrates the evaluation of the proposed method. Experiments are conducted on popular classification datasets, including CIFAR-10, CIFAR-100 [65], Tiny ImageNet [66], and ImageNet [67]. Besides, the Pascal VOC datasets are used for object detection tasks. We first compare the accuracy and compression ratio with state-of-the-art binary and decomposition methods. Then, we explore the effect of hyper-parameters in the proposed method.

A. Experiment Setting

For comparison, we select binary neural networks like BC [19], BNN [10], BWN, XNOR [28], and filter decomposition methods such as PENNI [44] and ABC-Net [46]. In addition, previous compression methods that stack single binary and full-precision filters, such as SLBF [68] and LegoNet [69], are also considered. "BLC" denotes our proposed method of linear combination with binary bases, while "TLC" and "MLC" refer to ternary and multibit quantization, respectively. 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.

1) Network Structure: For the image classification task, we primarily choose VGG-16 [70] and ResNet-18 [1] as our fundamental structures, while ResNet-101 is selected for object detection. In general, VGG-16 contains 13 convolutional layers and three fully connected layers, while the ResNet-18 model includes 18 convolutional layers followed by a fully connected layer. In our experiments, we simplify the structures by replacing the fully connection layers with average pooling layers. All the convolution filters except for the first layer are constructed using the proposed method, i.e., a linear combination of low-dimensional binary bases with sparse coefficients. The networks are trained 500 epochs from scratch with an initial learning rate of 0.01, which is adjusted using cosine annealing. In addition, the batch normalization layers with scaling and shifting are applied and Adam [71] with 0.0005 weight decay is used as the optimizer. The parameters for the competitors are set according to their original papers.

2) Datasets:

a) CIFAR-10 and CIFAR-100: The CIFAR-10 dataset consists of 60 000 32×32 colorful images in ten classes, and each class has 6000 images with 50 000 training images and remained 10 000 for testing. The difference of CIFAR-100 is that it contains 100 classes.

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

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

3) Hyper-Parameters: $q = (c_{\text{in}}/n)$ represents that the original filter is split into q blocks, and $k = (m/c_{\text{out}})$ means the ratio between the number 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.

B. Comparison With the State-of-the-Art Methods

1) Binary Neural Networks: Tables I–III present a comparison of the performance of different quantized networks. On ImageNet, the proposed method achieves much higher top-5 accuracy using ResNet-18. When applying ResNet-50, the proposed method outperforms other competitors with 73.0% top-1 accuracy. The proposed method is also able to compress lightweight models such as MobileNet. It is notable that we only compress the 1×1 pointwise convolution layers in MobileNet since they occupy most of the model parameters. The results show that the proposed method can achieve comparable accuracy to a full-precision MobileNet-V2 network. On Tiny-ImageNet in Table II, the proposed method achieves larger accuracy than other competitors, where the accuracy gap compared with full-precision VGG-16 network is only 0.98%.

On the CIFAR dataset in Table III, the proposed method achieves better accuracy than other methods and the improvement is encouraging when applying ResNet-18. We improve the accuracy of classical binary neural networks such as BNN [10] and XNOR-Net [28] by around 3% on CIFAR-10 dataset and up to 6% on CIFAR-100, which is even close to the full-precision network. For example, the network obtains a comparable accuracy of 74.61% on CIFAR-100. Regarding the recent binarized networks like RBNN [82] and ReCU [75], the proposed method is superior to them with respect to accuracy. It is clear that the proposed method outperforms other quantization methods to some extent, providing a promising way to improve the accuracy of quantized models.

2) Filter Decomposition and Compact Model: In Table IV, we provide the comparison results with decomposition methods. The recent work PENNI [44] is a compression method that combines network pruning and tensor decomposition. It is shown that the proposed method can achieve much better performance on accuracy on ResNet-18. Although we use 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 higher accuracy. With respect to compact models like SLBF [68] and LegoNet [69], the accuracy gap compared to fully network is also narrowed. On the whole, the results demonstrate the effectiveness of our proposed method in improving compression performance.

C. Ablation Study

1) Influence of Hyperparameters q and k: Considering the potential accuracy loss caused by a large compression ratio, we only evaluate the number of blocks $q = \{1, 2\}$. To control the variables and explore the effect of hyper-parameters $q = (c_{\text{in}}/n)$ and $k = (m/c_{\text{out}})$, we fix the sparsity between 0.92 and 0.97. Based on the theoretical compression ratio described in (6), a 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

TABLE I
COMPRESSION RESULTS ON IMAGENET

Network	Method	Bit-width (W/A)	Compressio Ratio	Top-1 Acc (%)	Top-5 Acc (%)
ResNet-18	Full Net	32/32	1×	69.3	89.2
	BNN [10]	1/1	32.0×	42.2	67.1
	BWN [28]	1/32	32.0×	60.8	83.0
	XNOR-Net [28]	1/1	32.0×	51.2	73.2
	ABC-Net [46]	5/5	6.4×	**65.0**	85.9
	XNOR-Net++ [28]	1/1	32.0×	55.5	78.5
	Bi-Real-Net [35]	1/1	32.0×	56.4	79.5
	LQ-Net [39]	1/2	32.0×	62.6	84.3
	CBCN [42]	4/4	8.0×	61.4	82.8
	GroupNet [72]	5/5	6.4×	64.8	85.7
	Si-BNN [73]	1/1	32.0×	59.7	81.8
	ProxyBNN [74]	1/1	32.0×	63.7	84.8
	ReCU [75]	1/1	32.0×	61.0	82.6
	AdaBin [76]	1/1	32.0×	63.1	84.3
	SiMaN [77]	1/1	32.0×	60.1	82.3
	BLC	1/32	26×	62.8	**86.1**
ResNet-50	Full Net	32/32	1×	76.0	93.0
	ABC-Net [46]	5/5	6.4×	70.1	89.7
	LQ-Net [39]	1/2	32.0×	68.7	88.4
	GroupNet [72]	5/5	6.4×	72.8	**90.5**
	BLC	1/32	5.8×	**73.0**	90.2
MobileNet-V2	Full Net	32/32	1×	71.9	90.3
	APoT [78]	5/5	6.4×	67.8	86.4
	UniQ [79]	1/1	32.0×	68.2	-
	RMSMP [80]	4/4	8.0×	69.0	89.1
	BLC	1/32	9.3×	**69.6**	**89.7**

TABLE II
COMPRESSION RESULTS ON TINY-IMAGENET

Network	Bit-width (W/A)	Compression Ratio	Acc (%)
Full VGG-16 Net	32/32	1×	51.85
BC [19]	1/32	32.0×	47.11
BWN [28]	1/32	32.0×	50.50
BNN [10]	1/1	32.0×	47.05
XNOR-Net [28]	1/1	32.0×	49.43
PENNI	32/32	35.1×	46.80
BLC	1/32	10.5×	**50.87**

certain because it depends on other values such as kernel size and number of channels. We thus analyze the performance under different settings of hyper-parameters q and k, and the experiment results are also consistent with this principle. The results are reported in Table V.

a) 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. On the VGG-16 network, we achieve 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 increases to 40× when k reduces to 0.5 but it leads to larger accuracy loss. For ResNet-18, when setting $k = 2$ and $q = 0.5$, we can compress the model by 35.4× with 3.06% loss compared with the full-precision network on CIFAR-100. The loss can be further narrowed by applying larger $k = 2$, which means using more binary bases. We also draw the tradeoff curve of compression ratio and accuracy in Fig. 2. 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 hyper-parameters in specific tasks.

b) Tiny-ImageNet results: The top-1 accuracy under different settings of q and k are reported in Table V(c). It is encouraging that we can obtain an accuracy close to a full-precision network with a relatively large compression ratio. The network can be compressed by 35.6× with 48.98% accuracy, where the loss of 2.87% is acceptable. In terms of the hyperparameters, using more binary bases will also lead to better accuracy. 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 increases to 0.5. Compared with the full-precision 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.

c) ImageNet results: On ResNet-18, we evaluate the compression ratio and accuracy under three different k values with $q = 2$. We are able to compress the model by around 26× with only 3.1% loss on top-5 accuracy. When reducing the compression ratio to 8.3× through adjusting k value, the accuracy gap between the proposed method and the full-precision network will be greatly narrowed.

d) Convergence: On the CIFAR-100 dataset, we also demonstrate the convergence of the proposed method. The convergence curve is shown in Fig. 3, indicating that the loss of the proposed method can reach an optimal status after a certain number of training iterations.

TABLE III
COMPRESSION WITH QUANTIZATION METHODS ON CIFAR-10 AND CIFAR-100

Network	Method	Bit-width (W/A)	Compression Ratio	CIFAR-10 Acc (%)	CIFAR-100 Acc (%)
VGG-16	Full Net	32/32	1×	93.25	73.55
	BWN [28]	1/32	32.0×	**93.09**	69.03
	BNN [10]	1/1	32.0×	91.21	67.88
	XNOR-Net [28]	1/1	32.0×	90.02	68.63
	BNN-DL [37]	1/1	32.0×	89.87	67.01
	IR-Net [81]	1/1	32.0×	90.56	67.15
	RBNN [82]	1/1	32.0×	91.04	67.19
	ReCU [75]	1/1	32.0×	90.73	67.78
	AdaBin [76]	1/1	32.0×	91.98	68.06
	SiMaN [77]	1/1	32.0×	92.01	67.99
	BLC	1/32	11.5×	92.05	**69.75**
ResNet-18	Full Net	32/32	1×	95.19	77.11
	BC [19]	1/32	32.0×	93.73	71.15
	BWN [28]	1/32	32.0×	93.97	72.92
	BNN [10]	1/1	32.0×	90.47	70.34
	XNOR-Net [28]	1/1	32.0×	90.14	72.87
	BNN-DL [37]	1/1	32.0×	92.66	72.45
	IR-Net [81]	1/1	32.0×	93.69	73.21
	RBNN [82]	1/1	32.0×	**94.39**	72.95
	ReCU [75]	1/1	32.0×	93.24	73.82
	AdaBin [76]	1/1	32.0×	93.52	74.16
	SiMaN [77]	1/1	32.0×	93.35	74.46
	BLC	1/32	35.9×	94.10	**74.61**

TABLE IV
COMPRESSION WITH DECOMPOSITION METHODS ON CIFAR-10 AND CIFAR-100

Network	Method	Bit-width (W/A)	Compression Ratio	CIFAR-10 Acc (%)	CIFAR-100 Acc (%)
VGG-16	Full Net	32/32	1×	93.25	73.55
	ABC-Net [46]	5/5	6.3×	89.69	67.54
	PENNI [44]	32/32	108.9×	**93.12**	64.42
	SLBF [68]	1/32	60.1×	91.44	68.80
	LegoNet [69]	32/32	5.4×	91.35	70.10
	BLC	1/32	11.5×	92.05	**69.75**
ResNet-18	Full Net	32/32	1×	95.19	77.11
	ABC-Net [46]	5/5	5.4×	90.64	71.01
	PENNI [44]	32/32	32.7×	94.01	70.90
	SLBF [68]	1/32	58.7×	93.82	74.59
	LegoNet [69]	32/32	17.5×	93.55	72.67
	BLC	1/32	35.9×	**94.10**	**74.61**

e) Training and inference time: To evaluate the impact of q and k on training and inference time, we set $q = 1$ and k in $\{0.2, 0.5\}$ and conduct experiments on CIFAR-100 using VGG-16. As shown in Table VI, the proposed method can also achieve lower inference latency than a full-precision network. It can also be observed that the time increases as k increases due to the consumption of selecting and combining the generated intermediate feature maps for each block. Since we have introduced the coefficient matrix G and applied sparsity operation on it, which will actually increase extra computation costs to some extent, such as the cost to manage and maintain the sparsity structure. Besides, like other implementations of binary networks, we still use the 32-bit floating points format to store the parameters and conduct computing in our experiments, because the current python framework (i.e., Pytorch and TensorFlow) does not support storing the data in binary form and bit type operations, which prevents us from fully realizing the potential benefits of using lower precision formats for parameter storage and computations. Consequently, the observed improvements in training and inference are less significant compared to the theoretical expectations. However, for quantized networks, low-bit computation operations still hold promise for reducing latency by utilizing some BNN library frameworks or hardware like FPGA and ASIC. In addition to applying quantization accelerators, several strategies will also be considered later to mitigate the negative impact of extra computations, such as optimizing the implementation of sparsity operations.

2) Impact of Sparsity: Table V also presents the comparison 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×, where the sparsity changes from 0.557 to 0.968 through adjusting the parameters in Algorithm 1. The same trend can

TABLE V

COMPRESSION OF DIFFERENT SETTINGS. (a) VGG-16 ON CIFAR-10 (93.25)/CIFAR-100 (73.55). (b) RESNET-18 ON CIFAR-10 (95.19)/CIFAR-100 (77.11). (c) VGG-16 ON TINY-IMAGENET (51.85). (d) RESNET-18 ON IMAGENET TOP-1 (69.3)/TOP-5 (89.2)

q	k	Sparsity	Compression Ratio	Acc (%)
		(a)		
1	0.5	0.949	40.0×	90.79 / 66.99
1	2	0.528	2.4×	91.85 / 68.70
1	2	0.946	9.8×	91.72 / 69.34
2	0.4	0.929	44.5×	91.02 / 67.04
2	0.5	0.522	5.4×	91.11 / 68.08
2	0.5	0.954	40.4×	90.99 / 68.14
2	2	0.528	1.4×	92.29 / 69.88
2	2	0.962	11.5×	92.05 / 69.75
		(b)		
1	0.4	0.954	42.9×	93.07 / 73.13
1	0.5	0.947	35.9×	94.10 / 74.61
1	2	0.557	2.1×	94.51 / 76.72
1	2	0.968	10.9×	94.39 / 76.48
2	0.3	0.924	38.7×	93.63 / 73.02
2	0.5	0.951	35.4×	93.97 / 74.05
2	2	0.952	8.3×	94.25 / 76.59
		(c)		
1	0.5	0.932	35.6×	48.98
1	2	0.955	10.5×	50.87
2	0.4	0.930	45.1×	48.01
2	0.5	0.937	32.3×	49.61
2	2	0.946	9.1×	52.03
		(d)		
1	0.5	0.939	15.6×	63.9 / 86.8
2	0.4	0.943	42.9×	61.5 / 84.9
2	0.5	0.936	26.0×	62.8 / 86.1
2	2	0.948	8.3×	65.3 / 87.6

also be observed on VGG-16 with the setting of $q = 2$ and $k = 2$. The results show that higher sparsity will lead to slightly lower accuracy while substantially increasing the compression ratio, where the accuracy loss ranges from 0.1% to 0.3%. Thus, in order to obtain a good tradeoff between accuracy and compression ratio, a coefficient with higher sparsity may be a good choice.

To analyze the impact of coefficient sparsity in detail, we fix $q = 2$ and $k = 0.5$ and adjust the threshold in Algorithm 1 to obtain different sparsity. We conduct experiments in four groups on the CIFAR dataset using VGG-16 and ResNet-18. Fig. 4 depicts the relationship between sparsity and accuracy. It can be observed that the four lines show a similar trend. As the sparsity increases from 0 to around 1, the accuracy first reaches a maximum and then starts to decrease. This trend indicates that coefficients that are too sparse or too dense both result in poor performance. Although it is difficult to find the exact maximum and the corresponding sparsity, we can still identify an appropriate setting according to specific demands, such as lower memory cost or better accuracy.

D. Ternary and Multibit Quantization

To explore better performance, we also evaluate the ternary and multibit quantization method on CIFAR-100. In addition

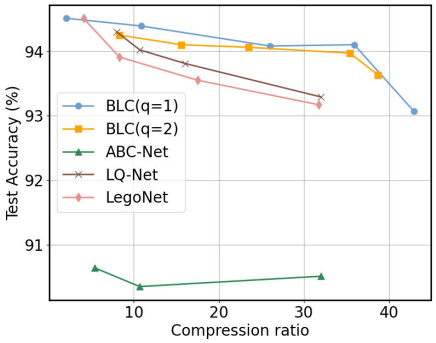


Fig. 2. Tradeoff curve of accuracy and compression ratio on CIFAR-10 using ResNet-18.

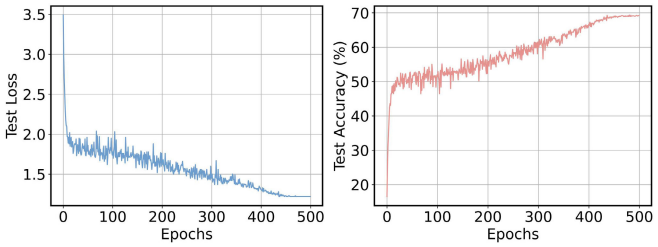


Fig. 3. Convergence curve of test loss/accuracy on CIFAR-100 using VGG-16. The curves trend to be stable at around 450 epochs, showing that our model can manage to reach convergence during training.

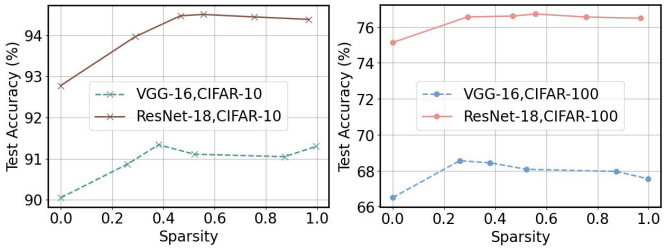


Fig. 4. Relationship between sparsity and accuracy. We fix q and k to explore the relationship. The sparsity values are adjusted from 0 to 1 by applying different thresholds in L_1-ball projection.

TABLE VI

COMPARISON ON TRAINING AND INFERENCE TIME. THE VALUES ARE OBTAINED ON NVIDIA RTX 2070 GPU

Method	Training Time (s)	Inference Time (s)
Full Net	18.95	3.02
BNN [10]	18.89	2.91
XNOR-Net [28]	18.56	2.62
AdaBin [76]	18.03	2.50
SiMaN [77]	18.77	2.69
BLC(q=1,k=0.5)	18.81	2.74
BLC(q=1,k=0.2)	18.27	2.59

to the sparse coefficient, ternary quantization further leads to sparse weights by introducing zero values. According to the results in Table VII, the network with ternary weights reserves more useful information, resulting in a slightly improved accuracy compared to binary weights. For instance, when fixing $q = 1$ and $k = 0.5$, the accuracy of TLC increases by approximately 1% on CIFAR-100 using VGG-16. By quantizing the weight into the 4-bit format, the accuracy gap with full-precision networks decreases from 3.8% to 3.3% under the

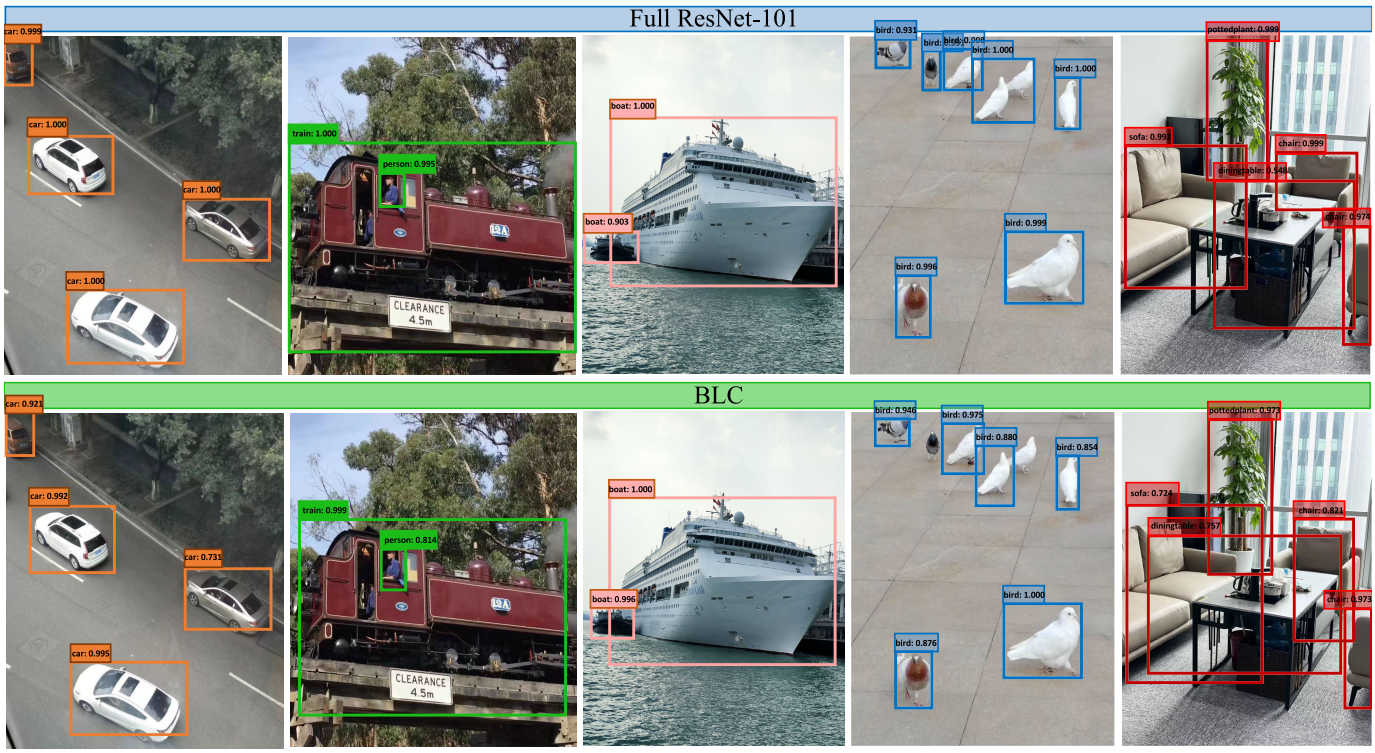


Fig. 5. Detection results on PASCAL VOC 2007. Images in the top row display the detected objects of the full-precision ResNet-101 network, while images in the bottom row are obtained by our model, that is, quantized ResNet-101 with fewer weight parameters.

TABLE VII
TERNARY AND MULTIBIT QUANTIZATION ACCURACY ON CIFAR-100

Parameter		BLC	TLC	MLC
$q = 1$	$k = 0.5$	66.99	68.05	67.96
	$k = 2$	69.34	69.54	69.16
$q = 2$	$k = 0.5$	68.14	68.68	68.29
	$k = 2$	69.75	70.15	70.20

TABLE VIII
RESULTS ON PASCAL VOC USING RESNET-101

Method	Bit-width (W/A)	mAP (%)
VOC2007_ResNet-101		
Full Net	32/32	74.8
XNOR-Net [28]	1/1	51.5
BLC	1/32	62.2
VOC2007+2012_ResNet-101		
Full Net	32/32	79.8
XNOR-Net [28]	1/1	56.7
BLC	1/32	75.0
VOC2007+2012_VGG-16		
Full Net	32/32	72.4
BNN [10]	1/1	42.0
XNOR-Net [28]	1/1	50.2
Bi-Real-Net [35]	1/1	63.8
DIR-Net [83]	1/1	67.1
AdaBin [76]	1/1	64.0
BLC	1/32	67.5

setting of $q = 2$ and $k = 2$, showing that multibit quantization is also beneficial to higher accuracy. These results explicitly illustrate the effectiveness of the proposed method in achieving compression with various quantization techniques.

E. Object Detection

We also apply the proposed method to an object detection task to verify its generalization ability. The Pascal VOC with 20 categories is selected as the benchmark dataset, and the models are constructed as ResNet-101 and VGG-16 backbone with Faster R-CNN [2] framework. Specifically, we first evaluate the proposed model on the VOC2007 train/val set (5011 images) and test on the VOC2007 test set (4952 images) with a batch size of 4. The models are pretrained on ImageNet for initialization. To provide more comprehensive results, we also trained the models on the larger VOC2007 + 2012 train/val set (16 551 images in total) and tested them on the VOC2007 dataset. Since the task is more challenging than classification, we retained the first and last block layers in ResNet-101 as full-precision following the setting in [84]. All the models are trained for ten epochs, and SGD with a weight decay of 1e-4 is applied as the optimizer. The initial learning rate is set as 4e-3 and decreases by a factor of 10 at the 8th epoch. We adopt the mean average precision (mAP) as an evaluation metric, which is widely used to assess the model performance in object detection tasks [85].

We first compare the mAP values of our model with full-precision ResNet-101 in Table VIII. Although there is still a relatively large margin of mAP compared to full-precision models on VOC2007, the proposed method outperforms XNOR-Net and manages to compress the model by around 10×. Some detection examples obtained by trained models on VOC2007 are shown in Figs. 5 and 6. It is illustrated that the

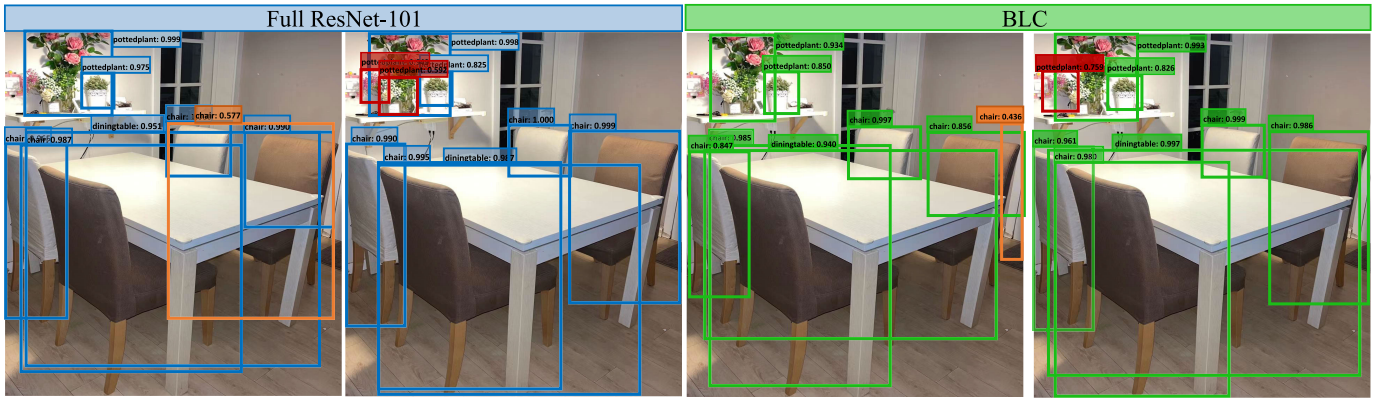


Fig. 6. Comparison of detection results trained by VOC2007 and VOC2007 + 2012. Left: Full ResNet-101. Right: the proposed method BLC. The false positives and missing objects are highlighted in orange and red, respectively.

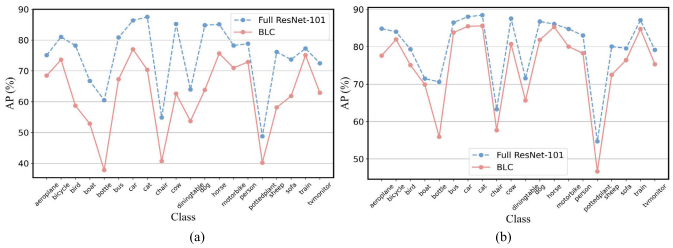


Fig. 7. Testing AP results of different classes obtained by the trained models on (a) VOC2007 and (b) VOC2007 + 2012.

proposed method can successfully detect the obvious objects, and even achieve similar accuracy as the full-precision network on some classes as shown in Fig. 7. It is also encouraging that the accuracy gap of each class will be greatly narrowed when training with more data on VOC2007 + 2012, where we obtain 75.0% mAP and 4.5× compression using ResNet-101. When applying VGG-16, the proposed method achieves 8.7× compression and 67.5% mAP, outperforming the other competitors. Fig. 6 also provides a snapshot for comparison of detecting results on VOC2007 and VOC2007 + 2012. Compared with the models trained on fewer data samples, the same model trained with more data can help eliminate the false positives (orange) and detect the missing objects (red). Moreover, the margin of objects can also be detected more accurately, suggesting the possibility of improving the performance of binary models on object detection tasks.

VII. Conclusion

In this article, we have proposed a novel method for compressing CNNs. Specifically, we have redesigned the convolution filters by stacking a linear combination of a set of low-dimensional quantized bases with corresponding coefficients, where the sparsity is explored to further save memory using L_1-ball projection algorithm. Detailed analysis of the memory cost and computation process has been provided as well. In the experiments, we have compared the performance of the proposed method with the state-of-the-art binary neural networks and filter decomposition methods. The results have shown that the proposed method can achieve a higher compression ratio with comparable accuracy under the different settings. Furthermore, the compression ratio can be adjusted through the hyper-parameters to meet up specific requirements, showing the flexibility of the proposed method to help achieve a good tradeoff between accuracy and compression. This provides a promising way to deploy CNNs on resource-constrained platforms.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, vol. 16, 2016, pp. 770–778.

[2] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," 2015, *arXiv:1506.01497*.

[3] M. Li, Z. Hu, Y. Lu, W. Lan, Y.-M. Cheung, and H. Huang, "Feature fusion from head to tail for long-tailed visual recognition," in *Proc. AAAI Conf. Artif. Intell.*, 2024, vol. 38, no. 12, pp. 13581–13589.

[4] W. Yin, K. Kann, M. Yu, and H. Schütze, "Comparative study of CNN and RNN for natural language processing," 2017, *arXiv:1702.01923*.

[5] Z. Hu, Y.-M. Cheung, Y. Zhang, P. Zhang, and P.-L. Tang, "Component-level Oracle bone inscription retrieval," in *Proc. Int. Conf. Multimedia Retr.*, May 2024, pp. 647–656.

[6] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Trans. Audio, Speech, Lang., Process.* vol. 22, no. 10, pp. 1533–1545, Oct. 2014.

[7] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.

[8] Z. Chen, T.-B. Xu, C. Du, C.-L. Liu, and H. He, "Dynamical channel pruning by conditional accuracy change for deep neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 2, pp. 799–813, Feb. 2021.

[9] T. Hao, X. Ding, J. Han, Y. Guo, and G. Ding, "Manipulating identical filter redundancy for efficient pruning on deep and complicated CNN," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Oct. 12, 2024, doi: 10.1109/TNNLS.2023.3298263.

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

[11] Z. Dong, Z. Yao, D. Arfeen, A. Gholami, M. W. Mahoney, and K. Keutzer, "HAWQ-V2: Hessian aware trace-weighted quantization of neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 18518–18529.

[12] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," 2015, *arXiv:1511.06530*.

[13] K. Wu, Y. Guo, and C. Zhang, "Compressing deep neural networks with sparse matrix factorization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 10, pp. 3828–3838, Oct. 2020.

[14] X. Jin et al., "Knowledge distillation via route constrained optimization," in *Proc. IEEE Int. Conf. Comput. Vis.*, Jul. 2019, pp. 1345–1354.

[15] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[16] S. Mehta, M. Rastegari, L. Shapiro, and H. Hajishirzi, "EspNetV2: A light-weight, power efficient, and general purpose convolutional neural network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Aug. 2019, pp. 9190–9200.

[17] W. Lan, Y.-M. Cheung, J. Jiang, Z. Hu, and M. Li, "Compact neural network via stacking hybrid units," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 1, pp. 103–116, Jan. 2024.

[18] Z. Yang et al., "CARS: Continuous evolution for efficient neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Sep. 2020, pp. 1829–1838.

[19] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," 2015, *arXiv:1511.00363*.

[20] X. Wang et al., "Modulated convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2018, pp. 840–848.

[21] B. Zhang, R. Wang, X. Wang, J. Han, and R. Ji, "Modulated convolutional networks," *IEEE Trans. Neural Netw. Learn. Syst.*, Mar. 9, 2021, doi: 10.1109/TNNLS.2021.3060830.

[22] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," 2013, *arXiv:1306.0543*.

[23] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned CP-decomposition," 2014, *arXiv:1412.6553*.

[24] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," 2014, *arXiv:1405.3866*.

[25] X. Ruan et al., "EDP: An efficient decomposition and pruning scheme for convolutional neural network compression," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 10, pp. 4499–4513, Oct. 2021.

[26] S. Wiedemann, K.-R. Müller, and W. Samek, "Compact and computationally efficient representation of deep neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 3, pp. 772–785, Mar. 2020.

[27] R. C. Gerum, A. Erpenbeck, P. Krauss, and A. Schilling, "Sparsity through evolutionary pruning prevents neuronal networks from overfitting," *Neural Netw.*, vol. 128, pp. 305–312, Aug. 2020.

[28] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 525–542.

[29] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," 2013, *arXiv:1308.3432*.

[30] F. Li, B. Liu, X. Wang, B. Zhang, and J. Yan, "Ternary weight networks," 2016, *arXiv:1605.04711*.

[31] W. Chen, P. Wang, and J. Cheng, "Towards mixed-precision quantization of neural networks via constrained optimization," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 5350–5359.

[32] Z. Liu, X. Zhang, S. Wang, S. Ma, and W. Gao, "Evolutionary quantization of neural networks with mixed-precision," in *Proc. ICASSP*, 2021, pp. 2785–2789.

[33] K. Helwegen, J. Widdicombe, L. Geiger, Z. Liu, K.-T. Cheng, and R. Nusselder, "Latent weights do not exist: Rethinking binarized neural network optimization," in *Proc. Adv. Neural Inform. Process. Syst.*, vol. 32, 2019, pp. 7531–7542.

[34] M. Alizadeh, J. Fernández-Marqués, N. D. Lane, and Y. Gal, "An empirical study of binary neural networks' optimisation," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 3123–3131.

[35] Z. Liu, B. Wu, W. Luo, X. Yang, W. Liu, and K.-T. Cheng, "Bi-real Net: Enhancing the performance of 1-bit CNNs with improved representational capability and advanced training algorithm," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 722–737.

[36] R. Gong et al., "Differentiable soft quantization: Bridging full-precision and low-bit neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 4852–4861.

[37] R. Ding, T.-W. Chin, Z. Liu, and D. Marculescu, "Regularizing activation distribution for training binarized deep networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 11408–11417.

[38] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, "Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 11, pp. 5784–5789, Nov. 2018.

[39] D. Zhang, J. Yang, D. Ye, and G. Hua, "LQ-Nets: Learned quantization for highly accurate and compact deep neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 365–382.

[40] P. Wang, X. He, and J. Cheng, "Toward accurate binarized neural networks with sparsity for mobile application," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 1, pp. 272–284, Jan. 2022.

[41] B. Lyu et al., "Designing efficient bit-level sparsity-tolerant memristive networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 9, pp. 11979–11988, Sep. 2023.

[42] C. Liu et al., "Circulant binary convolutional networks: Enhancing the performance of 1-bit DCNNs with circulant back propagation," in *Proc. CVPR*, 2019, pp. 2691–2699.

[43] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 10, pp. 1943–1955, Oct. 2015.

[44] S. Li, E. Hanson, H. Li, and Y. Chen, "PENNI: Pruned kernel sharing for efficient CNN inference," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 5863–5873.

[45] R. Kamiya, T. Yamashita, M. Ambai, I. Sato, Y. Yamauchi, and H. Fujiyoshi, "Binary-decomposed DCNN for accelerating computation and compressing model without retraining," in *Proc. IEEE Int. Conf. Comput. Vis. Workshops (ICCVW)*, Oct. 2017, pp. 1095–1102.

[46] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–9.

[47] W. Zhao, T. Ma, X. Gong, B. Zhang, and D. Doermann, "A review of recent advances of binary neural networks for edge computing," *IEEE J. Miniaturization Air Space Syst.*, vol. 2, no. 1, pp. 25–35, Mar. 2021.

[48] Y. Li, S. Gu, L. Van Gool, and R. Timofte, "Learning filter basis for convolutional neural network compression," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 5622–5631.

[49] H. Wang, P. Chen, and S. Kwong, "Building correlations between filters in convolutional neural networks," *IEEE Trans. Cybern.*, vol. 47, no. 10, pp. 3218–3229, Oct. 2017.

[50] Y. Chen et al., "Drop an octave: Reducing spatial redundancy in convolutional neural networks with octave convolution," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 3435–3444.

[51] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4510–4520.

[52] A. Howard et al., "Searching for MobileNetV3," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Oct. 2019, pp. 1314–1324.

[53] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 6848–6856.

[54] Z. Yan, X. Li, M. Li, W. Zuo, and S. Shan, "Shift-Net: Image inpainting via deep feature rearrangement," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 1–17.

[55] Y. Jeon and J. Kim, "Constructing fast network through deconstruction of convolution," 2018, *arXiv:1806.07370*.

[56] W. Chen, D. Xie, Y. Zhang, and S. Pu, "All you need is a few shifts: Designing efficient convolutional neural networks for image classification," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 7234–7243.

[57] M. Tan et al., "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Oct. 2019, pp. 2820–2828.

[58] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, "GhostNet: More features from cheap operations," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 1580–1589.

[59] D. Sculley, "Web-scale K-means clustering," in *Proc. World Wide Web Conf.*, 2010, pp. 1177–1178.

[60] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 806–814.

[61] L. Li, K. Wang, S. Li, X. Feng, and L. Zhang, "LST-Net: Learning a convolutional neural network with a learnable sparse transform," in *Proc. Eur. Conf. Comput. Vis.* Springer, 2020, pp. 562–579.

[62] W. Gao, Y. Guo, S. Ma, G. Li, and S. Kwong, "Efficient neural network compression inspired by compressive sensing," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 2, pp. 1965–1979, Feb. 2022.

[63] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through L_0 regularization," 2017, *arXiv:1712.01312*.

[64] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015, *arXiv:1506.02626*.

[65] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," CIFAR, Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009.
[66] L. Hansen, "Tiny ImageNet challenge submission," *CS 231N*, 2015.
[67] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2009, pp. 248–255.
[68] W. Lan and L. Lan, "Compressing deep convolutional neural networks by stacking low-dimensional binary convolution filters," 2020, *arXiv:2010.02778*.
[69] Z. Yang et al., "LegoNet: Efficient convolutional neural networks with lego filters," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7005–7014.
[70] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[71] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[72] B. Zhuang, C. Shen, M. Tan, L. Liu, and I. Reid, "Structured binary neural networks for accurate image classification and semantic segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 413–422.
[73] P. Wang, X. He, G. Li, T. Zhao, and J. Cheng, "Sparsity-inducing binarized neural networks," in *Proc. AAAI Conf. Artif. Intell.*, 2020, vol. 34, no. 7, pp. 12192–12199.
[74] X. He et al., "ProxyBNN: Learning binarized neural networks via proxy matrices," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 223–241.
[75] Z. Xu et al., "ReCU: Reviving the dead weights in binary neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 5198–5208.
[76] Z. Tu, X. Chen, P. Ren, and Y. Wang, "AdaBin: Improving binary neural networks with adaptive binary sets," in *Proc. Eur. Conf. Comput. Vis.* Tel Aviv-Yafo, Israel: Springer, 2022, pp. 379–395.
[77] M. Lin et al., "SiMaN: Sign-to-magnitude network binarization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 5, pp. 6277–6288, May 2023.
[78] Y. Li, X. Dong, and W. Wang, "Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks," 2019, *arXiv:1909.13144*.
[79] P. Pham, J. A. Abraham, and J. Chung, "Training multi-bit quantized and binarized networks with a learnable symmetric quantizer," *IEEE Access*, vol. 9, pp. 47194–47203, 2021.
[80] S.-E. Chang et al., "RMSMP: A novel deep neural network quantization framework with row-wise Xixed schemes and multiple precisions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 5251–5260.
[81] H. Qin et al., "Forward and backward information retention for accurate binary neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 2250–2259.
[82] M. Lin et al., "Rotated binary neural network," in *Proc. Adv. Neural Inform. Process. Syst.*, vol. 33, 2020, pp. 7474–7485.
[83] H. Qin, X. Zhang, R. Gong, Y. Ding, Y. Xu, and X. Liu, "Distribution-sensitive information retention for accurate binary neural network," *Int. J. Comput. Vis.*, vol. 131, no. 1, pp. 26–47, Jan. 2023.
[84] Z. Wang, Z. Wu, J. Lu, and J. Zhou, "BiDet: An efficient binarized object detector," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 2049–2058.
[85] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *Proc. IEEE*, vol. 111, no. 3, pp. 257–276, Mar. 2023.

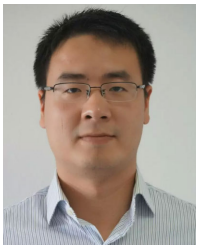
Weichao Lan received the B.S. degree in electronics and information engineering from Sichuan University, Chengdu, China, in 2019, and the Ph.D. degree from the Department of Computer Science, Hong Kong Baptist University, Hong Kong, in 2024, under the supervision of Prof. Yiu-Ming Cheung.

Her present research interests include network compression and acceleration and lightweight models.

Yiu-Ming Cheung (Fellow, IEEE) received the Ph.D. degree from the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, in 2000.

He is currently the Chair Professor with the Department of Computer Science, Hong Kong Baptist University, Hong Kong. His research interests include machine learning and visual computing, and their applications in data science, pattern recognition, multiobjective optimization, and information security.

Prof. Cheung is a fellow of American Association for the Advancement of Science (AAAS), the Institution of Engineering and Technology (IET), and British Computer Society (BCS). Also, he is an Awardee of RGC Senior Research Fellow. He is the Editor-in-Chief of IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE, and an Associate Editor for several prestigious journals, including IEEE TRANSACTIONS ON CYBERNETICS, IEEE TRANSACTIONS ON COGNITIVE AND DEVELOPMENTAL SYSTEMS, *Pattern Recognition*, *Pattern Recognition Letters*, and *Neurocomputing*, to name a few. For more details, please refer to: https://www.comp.hkbu.edu.hk/~ymc.

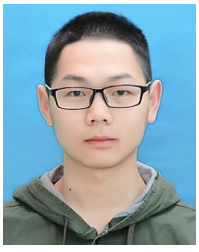


Liang Lan received the Ph.D. degree in computer sciences from Temple University, Philadelphia, PA, USA, in 2012.

He is currently a Senior Lecturer with the Department of Interactive Media, School of Communication, Hong Kong Baptist University, Hong Kong. His research interests include artificial intelligence, machine learning, and their applications in social science, media, and communications.

Juyong Jiang received the B.S. degree in computer science and technology from Hohai University, Nanjing, China, in 2020. He is currently pursuing the Ph.D. degree in data science and analytics thrust with Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China, under the supervision of Prof. Sunghun Kim.

He was a Research Assistant with the Department of Computer Science, Hong Kong Baptist University, Hong Kong, from 2021 to 2022. His current research interests include data mining, large language models, and code generation.



Zhikai Hu (Member, IEEE) received the B.S. degree in computer science from China Jiliang University, Hangzhou, China, in 2015, and the M.S. degree in computer science from Huaqiao University, Xiamen, China, in 2019. He is currently pursuing the Ph.D. degree with the Department of Computer Science, Hong Kong Baptist University, Hong Kong, under the supervision of Prof. Yiu-Ming Cheung.

His present research interests include information retrieval, pattern recognition, and data mining.