Model Decomposition and Acceleration for Quantized Neural Networks

Journal:	*Transactions on Pattern Analysis and Machine Intelligence*
Manuscript ID	TPAMI-2019-03-0228
Manuscript Type:	Regular
Keywords:	Deep Neural Networks, Model compression and acceleration, Multi-branch Binary Networks, Bitwise Operations, Smart Chips

Model Decomposition and Acceleration for Quantized Neural Networks

Qigong Sun, *Student Member, IEEE,* Xiufang Li, Fanhua Shang, *Member, IEEE,* Hongying Liu, Kang Yang, Licheng Jiao, *Fellow, IEEE,* and Zhouchen Lin, *Fellow, IEEE*

Abstract—The training of deep neural networks (DNNs) always requires intensive resources for both computation and data storage. Thus, DNNs cannot be efficiently applied to mobile phones and embedded devices, which severely limits their applicability in industrial applications. To address this issue, we propose a novel encoding scheme of using $\{-1, +1\}$ to decompose quantized neural networks (QNNs) into multi-branch binary networks, which can be efficiently implemented by bitwise operations (*xnor* and *bitcount*) to achieve model compression, computational acceleration, and resource saving. Based on our method, users can achieve different encoding precisions arbitrarily according to their requirements and hardware resources. The proposed mechanism is highly suitable for the use of FPGA and ASIC in terms of data storage and computation, which provides a feasible idea for smart chips. We validate the effectiveness of our method on large-scale image classification (e.g., ImageNet), object detection, and semantic segmentation tasks. In particular, our method with low-bit encoding can still achieve almost the same performance as its 32-bit counterparts.

Index Terms—Deep Neural Networks, Model compression and acceleration, Multi-branch Binary Networks, Bitwise Operations, Smart Chips

1 Introduction

Deep Neural Networks (DNNs) have been successfully applied in many fields, such as image classification, object detection and natural language processing. Because of the large number of parameters in multilayer networks and complex model architectures, huge storage space and considerable power consumption are needed. For instance, the VGG-16 model has 138.34 million parameters, and requires more than 500MB storage space and 30.94 billion float point operations in the inference process. With the rapid development of chip technology, especially GPU and TPU, the computing power has been dramatically improved. In the rapid evolution of deep learning, most researchers use multiple GPUs or computer clusters to train deeper and even more complex neural networks. Nevertheless, the energy consumption and limitation of computing resources are still significant factors in industrial applications, which are generally ignored in scientific research. In other words, breathtaking results of many DNNs algorithms under the condition of applying GPUs lag behind the demand of the industry. DNNs can hardly be applied in mobile phones and embedded devices (as typical industrial applications) directly due to their limited memory and calculation resources. Therefore, the model compression and acceleration in deep networks are especially important in the future industrial and commercial applications.

In recent years, many solutions have been proposed to improve the energy efficiency of hardware and achieve model compression or computational acceleration, such as network sparsification and pruning [2]–[4], low-rank approximation [5]–[7], architecture design [8], [9] and model quantization [10]–[12]. Network sparse and pruning can dramatically reduce the number of connections in the network, and thus reduce the computational load in the inference process without large accuracy drop. Luo et al. [13] presented a unified framework (ThiNet), which exploited a filter level pruning method to discard less important filters. [7] used low-rank tensor decomposition to remove the redundancy in the kernel, which can serve as a generic tool for speeding up. Since there is some redundant information in the networks, most direct approaches of cutting down those information are to optimize the structure and yield small networks [14], [15]. For example, [8] proposed to use bitwise separable convolutions to build light networks for mobile applications. Most of those networks still utilize floating-point number representations (i.e., full-precision values). However, [16] discussed that the representation of the full-precision weights and activations in networks is not necessary during the training of DNNs, and a nearly identical or slightly better accuracy rate may be obtained under lower-precision representation and calculation. Therefore, many scholars are working on the topic of model quantization to achieve model compression and resource saving. Model quantization is more suitable and can be applied to many network architectures and applications. It can also combine with other strategies (e.g., pruning) to achieve more efficient performance.

Model quantization methods mainly include low-bit quantization approaches (e.g., BNN [17], XNOR-Net [11], TWN [18], TBN [19]) and m-bit quantization approaches (e.g., DoReFa-Net [20], INQ [21], ABC-Net [12], LQ-Nets

Preliminary results of this paper appeared in [1].

- *Q. Sun, X. Li, F. Shang, Y. Liu, K. Yang, and L. Jiao are the Key Laboratory of Intelligent Perception and Image Understanding of the Ministry of Education, International Research Center for Intelligent Perception and Computation, Xidian University, Xi'an Shaanxi Province 710071, China. E-mails: xd_qigongsun@163.com; xfl_xidian@163.com; {fhshang, hyliu}@xidian.edu.cn; yangk12321@gmail.com; lchjiao@mail.xidian.edu.cn.*
- *Z. Lin is with Key Laboratory of Machine Perception (MOE), School of EECS, Peking University, Beijing 100871, P.R. China. E-mail: zlin@pku.edu.cn.*

[22]). Those low-bit quantization methods can achieve extreme model compression, computational acceleration and resource saving. For example, XNOR-Net [17] uses bitwise operations (i.e., *xnor* and *bitcount*) to replace full-precision matrix multiplication, achieving 58× speedups and 32× memory saving in CPU [11]. As discussed in [23], the method attains a higher acceleration ratio on FPGA, which can speed up to about 705× in the peak condition compared with CPU and is 70× faster than GPU. However, most models were proposed for a fixed precision, and cannot be extended to other precision models. The representation capability of binary parameters is insufficient for many practical applications, especially for large-scale image classification (e.g., ImageNet) and regression tasks. Therefore, they suffer from significant performance degradation. The representation capability of m-bit quantization is stronger than low-bit quantization, and can be applied to more complex real-world applications. Some scholars use m-bit quantization to improve both the accuracy and compression ratio of their networks, but seldom consider their computational acceleration [16], [21], [24]. The slow convergence rate in the training process and the lack of efficient acceleration strategies in the inference process are two typical issues in m-bit quantization networks. ABC-Net [12] and LQ-Nets [22] used the linear combination of multiple binary parameters $\{-1, +1\}$ to approximate full-precision weights and activations. Therefore, the complex full-precision matrix multiplication can be decomposed into some simpler operations. [25] and [26] used the same technique to accelerate the training of CNNs and RNNs. In addition, those methods not only increase the number of parameters many times, but also introduce a scale factor to transform the original problem into an NP-hard problem, which makes the solution difficult and highly complex.

In order to extend the efficient bitwise operations (*xnor* and *bitcount*) of low-bit quantization to m-bit quantization without excessively increasing time complexity and space complexity, we propose a novel encoding scheme of using $\{-1, +1\}$ to decompose trained quantized neural networks (QNNs) into multi-branch binary networks. Our method bridges the gap between low-bit and full-precision quantization, and can be applied to many cases (from 1-bit to 8-bit). Thus, our encoding mechanism can improve the utilization of hardware resources, and achieve parameter compression and computation acceleration. In our experiments, we not only validate the performance of our method for image classification on CIFAR-10 and large-scale datasets, e.g., ImageNet, but also implement object detection and semantic segmentation tasks. The typical advantages of our method are summarized as follows:

- **Easy training**. For our networks, we can directly use the high-bit model parameters to initialize a low-bit model for faster training. Hence, our networks can be trained in a short time, and only dozens of times fine-tuning are needed to achieve the accuracies in our experiments. Of course, we can get better performance if we continue training the network. Thus, our multi-precision quantized networks can be easily popularized and applied to engineering practices.
- **Provide multiple options**. Depending on our encoding scheme, our method can provide 64 available encoding options for different precision networks. Naturally, each option with different encoding precisions has different calculation speeds, resource requirements and experimental precisions. Therefore, users can choose the appropriate option for their requirement.
- **Suitable for hardware implementation**. After the process of decomposition, instead of storing all encoding bits in data types, e.g., char, int, float or double, the parameters can be individually stored by bit vectors. Thus, the smallest unit of data in electronic equipments can be reduced to 1-bit from 8-bit, 16-bit, 32-bit or 64-bit, which raises the utilization rate of resources and compression ratios of the model. Then the data can be encoded, calculated and stored in various encoding precisions.
- **Suitable for multiple networks and applications**. In our experiments, we evaluate our method on many networks (e.g., AlexNet, ResNet-18, ResNet-50, InceptionV3, MobileNetV2 and DeepLabV3) and applications (e.g., Image Classification, Object Detection and Semantic Segmentation). Our encoding scheme could also be applied to many other networks and applications.

2 Related Work

Neural network quantization has attracted a lot of attention to promote the industrial applications of deep learning, especially for mobile phones, video surveillance, unmanned aerial vehicle and unmanned driving. Many researchers use quantitative methods to get lighter and more efficient networks. All the research is mainly focusing on the following three issues.

How to quantize the values of weights or activation in deep networks to achieve model compression? Quantization methods play a significant role in QNNs, and can be used to convert their floating-point weights and activations into their fixed-point (e.g., from 1-bit to 16-bit) counterparts for achieving model compression. [16] used the notation of integer and fractional to denote a 16-bit fixed-point representation, and proposed a stochastic rounding method to quantify values. [24] used 8-bit quantization to convert weights into signed char and activation values into unsigned char, and all the values are integer. For multi-state quantification (from 2-bit to 8-bit), linear quantization is used in [10], [27], [28]. Besides, [29] proposed logarithmic quantization to represent data and used bitshift operations in log-domain to compute dot products. For ternary weight networks [18], the weights are quantized to $\{-\Delta^*, 0, \Delta^*\}$, where $\Delta^* = 0.7 \cdot E(|W|)$. In [30], the positive and negative states are trained together with other parameters. When the states are constrained to 1-bit, [17] applied the sign function to quantize weights and activation values to -1 and 1. [11] also used $\{-\alpha^*, \alpha^*\}$ to represent the binary states, where $\alpha^* = \frac{1}{n}\|W\|_{l_1}$.

How to achieve computational acceleration in the inference process? After quantized training, weights or activation values are represented in a low-bit form (e.g., 1-bit, 2-bit), which has the potential of acceleration computation

and memory saving. The most direct quantization is to convert floating-point parameters into fixed-point (e.g., 8-bit), and we can achieve hardware acceleration for fixed-point based computation [16], [24], [31], [32]. When the weight is extremely quantized to the binary weight $\{-1,+1\}$ as in [33] or ternary weight $\{-1,0,+1\}$ as in [18], [34], the matrix multiplication can be transformed into full-precision matrix addition and subtraction to accelerate computation. Especially when the weight and activation values are binarized, matrix multiplication operations can be transformed into highly efficient *xnor* and *bitcount* operations [11], [17]. Bitwise operations are based on 1-bit units, therefore, they cannot be directly applied to multi-bit computation. [12], [25] used a series of linear combinations of $\{-1,+1\}$ to approximate the parameters of full-precision convolution models, and then converted floating point operations into multiple binary weight operations to achieve model compression and computation acceleration. Miyashita et al. [29] applied logarithmic data representation on activations and weights that transform the dot product from linear domain into log-domain. Therefore, the dot product can be computed by bitshift operation to achieve computational acceleration.

How to train QNNs in the quantized domain? It is clear that some discrete functions, which may be non-differentiable or have zero derivatives everywhere, are needed to quantize weights or activation values in QNNs. The traditional gradient descent methods are unsuitable for the training of deep networks. Therefore, many researchers are devoting themselves to addressing this issue. All the optimization methods can be divided into two categories: quantizing pre-trained models with or without retraining [12], [18], [35], [36] and directly training quantized networks [11], [17], [27], [33]. [10], [17] and [26] used the straight-through estimator (STE) [37] to train deep networks. This method uses nonzero gradient to approximate the function gradient, which is not-differentiable or whose derivative is zero, and then applies the stochastic gradient descent (SGD) to update the parameters. Zhuang et al. [28] proposed a two stage optimization procedure, which first optimizes a net with quantized weights, and then quantized activations, to progressively find good local minima and then decreases the bit-width from high-precision to low-precision. [38], [39] and [28] use high-precision teacher networks to guide low-precision student networks to improve network performance, which is called knowledge distillation. [12], [22], [40] and [41] use an alternating optimization method to train network weights and their scaling factors.

3 Multi-Precision Quantized Neural Networks

In this section, we propose a novel encoding scheme of using $\{-1,+1\}$ to decompose QNNs into multi-branch binary networks. In each branch binary network, we use -1 and $+1$ as the basic elements to efficiently achieve model compression and forward inference acceleration for QNNs. Different from fixed-precision neural networks (e.g., binary, ternary), our method can yield multi-precision networks and make full use of the advantage of bitwise operations to accelerate networks.

3.1 Computational Decomposition

As the basic computation in most layers of neural networks, matrix multiplication costs lots of resources and is also the most time-consuming operation. Modern computers store and process data in binary format, thus non-negative integers can be directly encoded by $\{0,1\}$. Therefore, we propose a novel decomposition scheme to accelerate matrix multiplication as follows: Let $x=[\mathrm{x}^1,\mathrm{x}^2,...,\mathrm{x}^N]^T$ and $w=[\mathrm{w}^1,\mathrm{w}^2,...,\mathrm{w}^N]^T$ be two vectors of non-negative integers, where $\mathrm{x}^i,\mathrm{w}^i\in\{0,1,2,...\}$ for $i=1,2,...,N$. The dot product of those two vectors is written as follows:

$$\begin{aligned} x^T\cdot w &= [\mathrm{x}^1,\mathrm{x}^2,...,\mathrm{x}^N][\mathrm{w}^1,\mathrm{w}^2,...,\mathrm{w}^N]^T \\ &= \sum_{n=1}^{N}\mathrm{x}^n\cdot\mathrm{w}^n. \end{aligned} \tag{1}$$

All of the above operations consist of N multiplications and $(N-1)$ additions. Based on the above $\{0,1\}$ encoding scheme, the vector x can be encoded to binary form using M bits, i.e.,

$$x=[\overbrace{\mathrm{c}_M^1\mathrm{c}_{M-1}^1...\mathrm{c}_1^1}^{\mathrm{x}^1},\overbrace{\mathrm{c}_M^2\mathrm{c}_{M-1}^2...\mathrm{c}_1^2}^{\mathrm{x}^2},...,\overbrace{\mathrm{c}_M^N\mathrm{c}_{M-1}^N...\mathrm{c}_1^N}^{\mathrm{x}^N}]^T. \tag{2}$$

Then the right-hand side of the formulation (2) can be converted into the following form:

$$\begin{bmatrix} \mathrm{c}_M^1 & \mathrm{c}_M^2 & \cdots & \mathrm{c}_M^N \\ \mathrm{c}_{M-1}^1 & \mathrm{c}_{M-1}^2 & \cdots & \mathrm{c}_{M-1}^N \\ \vdots & \vdots & \cdots & \vdots \\ \mathrm{c}_1^1 & \mathrm{c}_1^2 & \cdots & \mathrm{c}_1^N \end{bmatrix} = \begin{bmatrix} c_M \\ c_{M-1} \\ \vdots \\ c_1 \end{bmatrix}, \tag{3}$$

where

$$\mathrm{x}^i = \sum_{m=1}^{M}2^{m-1}\cdot\mathrm{c}_m^i,\quad \mathrm{c}_m^i\in\{0,1\}, \tag{4}$$

$$c_j = [\mathrm{c}_j^1,\mathrm{c}_j^2,...,\mathrm{c}_j^N],\ j=1,2,\ldots,M. \tag{5}$$

In such an encoding scheme, the number of represented states is not greater than 2^M. Besides, we encode another vector w with K-bit representation in the same way. Therefore, the dot product of the two vectors can be computed as follows:

$$\begin{aligned} x^T\cdot w &= \sum_{n=1}^{N}\mathrm{x}^n\cdot\mathrm{w}^n \\ &= \sum_{n=1}^{N}\left(\sum_{m=1}^{M}2^{m-1}\cdot\mathrm{c}_m^n\right)\cdot\left(\sum_{k=1}^{K}2^{k-1}\cdot\mathrm{d}_k^n\right) \\ &= \sum_{m=1}^{M}\sum_{k=1}^{K}2^{m-1}\cdot2^{k-1}\cdot c_m\cdot d_k^T. \end{aligned} \tag{6}$$

From the above formula, the dot product is decomposed into $M\times K$ sub-operations, in which every element is 0 or 1. Because of the restriction of encoding and without using the sign bit, the above representation can only be used to encode non-negative integers. However, it is impossible to limit the weights and the values of the activation functions to non-negative integers.

In order to extend encoding space to negative integer and reduce the computational complexity, we propose a new encoding scheme, which uses $\{-1,+1\}$ as the basic

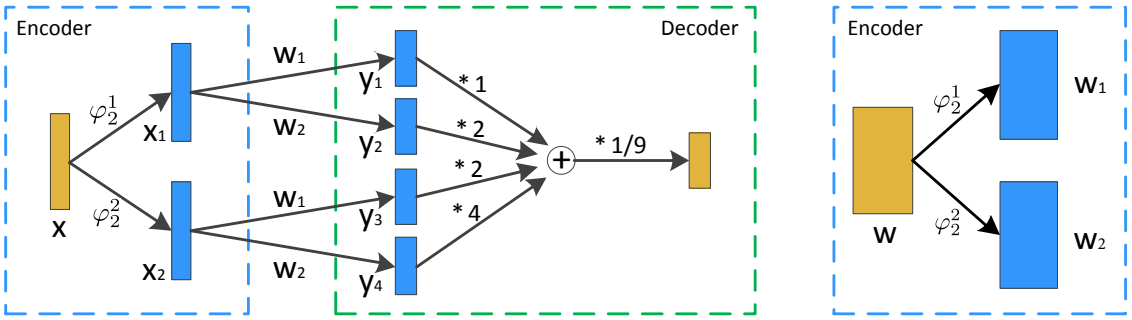


Fig. 1: Architecture of 2-bit encoding in a fully connected layer. Here, 2BitEncoder ($\varphi_2^1(x)$ and $\varphi_2^2(x)$) is used to encode input data and weights in our Encoder stage, and the weighted sum of those four results by fixing scale factors is the final output in our Decoder stage.

elements of our encoder rather than $\{0, 1\}$. Except for the difference of basic elements, the encoding scheme is similar to the rules in Eq. (4), and is formulated as follows:

$$x^i = \sum_{m=1}^{M} 2^{m-1} \cdot \mathrm{c}_m^i, \quad \mathrm{c}_m^i \in \{-1, +1\} \tag{7}$$

where M denotes the number of encode bits, that can represent 2^M states. At this time, the encoding space is extended to $\{..., -3, -1, 1, 3, ...\}$. We can use multiple bitwise operations (i.e., *xnor* and *bitcount*) to effectively achieve the above vector multiplications. Therefore, the dot product of those two vectors can be computed as follows:

$$x^T \cdot w = \sum_{m=1}^{M} \sum_{k=1}^{K} 2^{m-1} \cdot 2^{k-1} \cdot \mathit{XnorPopcount}(c_m, d_k) \tag{8}$$

where $\mathit{XnorPopcount}(\cdot)$ denotes the bitwise operation, and its details will be described below. Therefore, this operation mechanism is suitable for all vector/matrix multiplications.

3.2 Model Decomposition

In various neural networks, matrix multiplication is the basic operation in some commonly used layers (e.g., fully connected layers and convolution layers). Based on the above computational decomposition mechanism of matrix multiplication, we can decompose the quantized networks into a more efficient architecture.

Let M be the number of encoding bits, therefore, the encode space can be defined as $\{-(2^M-1), -(2^M-3), ..., (2^M-3), (2^M-1)\}$, in which each state is an integer number. The boundaries of values are determined by the number of encoding bits, M. However, the parameters in neural networks are usually distributed within the interval $[-l, +l]$. In order to apply our decomposition mechanism to network model decomposition, we use the following formulation for the computation of common layers in neural networks.

$$Op(x, w) = \frac{l}{(2^M-1)(2^K-1)} Op(EnC(x), EnC(w)) \tag{9}$$

where $Op(\cdot)$ denotes the basic multiple layer operations (e.g., fully connected layer and convolution layer), M is the number of encode bits of input data x, K denotes the number of encode bits of weights w, and $EnC(\cdot)$ represents the encoding process. $l/[(2^M-1)(2^K-1)]$ can be viewed as a scaling factor, and l is usually set to 1. If there is a batch normalization layer after the above layer, the scaling factor is not used in the computation of our decomposition scheme.

We use the 2-bit encoding scheme (i.e., $M = K = 2$) for a fully connected layer as an example to introduce the mechanism of our decomposition model, as shown in Fig. 1. We define an "Encoder" that can be used in the 2BitEncoder function ($\varphi_2^1(\cdot)$ and $\varphi_2^2(\cdot)$), as defined below, for encoding input data. For example, x can be encoded by $x_1 \in \{-1, +1\}^N$ and $x_2 \in \{-1, +1\}^N$, where x_2 represents high bit data and x_1 represents low bit data. Therefore, we have the following formula: $x = x_1 + 2x_2$. In the same way, the weight w can be converted into $w_1 \in \{-1, +1\}^{M \times N}$ and $w_2 \in \{-1, +1\}^{M \times N}$. After cross multiplications, we get the four intermediate variables $\{y_1, y_2, y_3, y_4\}$. In other words, each multiplication can be considered as a binarized fully connected layer, whose elements are -1 or $+1$. This decomposition can produce multi-branch layers, thus we call our networks as multi-branch binary networks (MBBNs). For instance, we decompose the 2-bit fully connection operation into four branches binary operations, which can be accelerated by bitwise operations, and then the weighted sum of those four results by the fixed scaling factor (e.g., $1/9$ in Fig. 1) is the final output. Besides fully connected layers, our decomposition scheme is suitable for the convolution and deconvolution layers in various deep neural networks.

3.3 M-bit Encoding Functions

As an important part of various neural networks, activation functions can enhance the nonlinear characterization of networks. In our model decomposition method, the encoding function also plays a critical role and can encode input data to multi-bits (-1 or $+1$). Those numbers represent the encoding of input data. For some other QNNs, several quantization functions have been proposed. However, it is not clear that what the affine mapping is between quantized numbers and encode bits. In this subsection, a list of M-bit encoding functions are proposed to produce the element of each bit that follows the rules for encoding data.

Before encoding, the data should be limited to a fixed numerical range. Table 1 lists the four activation functions.

TABLE 1: Activation functions that are used to limit input data to a fixed numerical range.

$Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$HTanh(x) = \begin{cases} 1, & x > 1 \\ x, & -1 \leqslant x \leqslant 1 \\ -1, & -1 \leqslant x \end{cases}$
$Sigmoid(x) = \frac{1}{1+e^{-x}}$	$HReLU(x) = \begin{cases} 1, & x > 1 \\ x, & 0 \leqslant x \leqslant 1 \\ 0, & x \leqslant 0 \end{cases}$

$HTanh(\cdot)$ is used to project input data to $[-1, +1]$, and it consists of the sign function to achieve binary encoding of weights and activations [17], [23]. Since the convergence of *SGD* obtained by using $ReLU(\cdot)$ is faster than other activation functions, we propose a new activation function $HReLU(\cdot)$ that retains the linear characteristics in the specific range and limits the range of input data to $[0, 1]$. Different from general activation functions mentioned above, the output of our M-bit encoding function defined below should be M numbers, which are -1 or $+1$. Those numbers represent the encoding of input data. Therefore, the dot product can be computed by the formula (9). In addition, in the above described experimental condition, when we use 2-bit to encode the data x and constrain to $[-1, +1]$, there are 4 encoded states, as shown in Table 2. The affine mapping between quantized real numbers and their encoded states is given in the following table.

TABLE 2: Quantized real numbers and their Encoded states.

Number Space	$[-1, -2/3]$	$(-2/3, 0]$	$(0, 2/3)$	$[2/3, 1]$
Quantized Number	-1	$-1/3$	$1/3$	1
Encoded State	$\{-1, -1\}$	$\{-1, +1\}$	$\{+1, -1\}$	$\{+1, +1\}$

From the above results, we can see that there is a linear factor α between quantized real numbers and encoded states (i.e., $\alpha = 3$ for Table 2). When we use the formula (9) to compute the multiplication of two encoded vectors, the value will be expanded α^2 times. Therefore, the result can multiply its scaling factor to get the final result, e.g., $1/9$ shown in Fig. 1. Fig. 2 shows the illustration of 2-bit and 3-bit encoding functions. We can see that those encoding functions are required to be periodic, and the functions have different periods. Naturally, we apply trigonometric functions as the basic encoding functions, which are signed as red lines. Finally, we use the sign function to hard divide to -1 or $+1$. The mathematical expression can be formulated as follows:

$$MBitEncoder(x) = \begin{cases} \varphi_M^m(x) : sign(sin(\frac{2^M-1}{2^m}\pi \cdot x)), \\ m = M, \\ \varphi_M^m(x) : sign(-sin(\frac{2^M-1}{2^m}\pi \cdot x)), \\ m \in \{1, 2, ..., M-1\} \end{cases} \quad (10)$$

when $m = M$, $\varphi_M^M(x)$ is the encoding function of the highest bit of MBitEncoder. For the above 2-bit example, the 2BitEncoder can be formulated as follows:

$$2BitEncoder(x) = \begin{cases} \varphi_2^2(x) : sign(sin(\frac{3}{4}\pi \cdot x)), \\ \varphi_2^1(x) : sign(-sin(\frac{3}{2}\pi \cdot x)) \end{cases} \quad (11)$$

where $\varphi_2^1(x)$ denotes the encoding function of the first bit (x_1^i) of 2BitEncoder, and $\varphi_2^2(x)$ represents the encoder function of the second bit (x_2^i) of 2BitEncoder. The periodicity is clearly different from the others because it needs to denote more states.

4 Training Networks

QNNs face the problem that the derivative is not defined, thus traditional gradient optimization methods are not applicable. [17] presented the *HTanh* function to binarily quantize both weights and activations, and also defined the derivative to support the back-propagation (BP) training process. They used the loss computed by binarized parameters to update full precision parameters. Similarly, [11] also proposed to update the weights with the help of the second parameters. [37] discussed that using STE to train network models containing discrete variables can obtain faster training speed and better performance.

4.1 Training Multi-Branch Binary Networks

Generated by the decomposition of QNNs, MBBNs need to use M-bit encoding functions to get the elements of each bit, which can be used by more efficient bitwise operations to replace arithmetic operations. The sign function of the encoder makes it difficult to implement the BP process. Thus, we approximate the derivative of the encoding function with respect to x as follows:

$$\begin{cases} \frac{\partial \varphi_M^m(x)}{\partial x} = \begin{cases} \frac{2^M-1}{2^m}\pi cos(\frac{2^M-1}{2^m}\pi x), & -1 \leqslant x \leqslant 1 \\ 0 & \text{otherwise,} \end{cases}, \\ m = M, \\ \frac{\partial \varphi_M^m(x)}{\partial x} = \begin{cases} -\frac{2^M-1}{2^m}\pi cos(\frac{2^M-1}{2^m}\pi x), & -1 \leqslant x \leqslant 1 \\ 0 & \text{otherwise,} \end{cases}, \\ m \in \{1, 2, ..., M-1\}. \end{cases} \quad (12)$$

We take the 2-bit encoding as an example to describe the optimization method of MBBNs.

$$\begin{cases} \frac{\partial \varphi_2^2(x)}{\partial x} = \begin{cases} \frac{3}{4}\pi cos(\frac{3}{4}\pi x), & -1 \leqslant x \leqslant 1 \\ 0 & \text{otherwise,} \end{cases} \\ \frac{\partial \varphi_2^1(x)}{\partial x} = \begin{cases} -\frac{3}{2}\pi cos(\frac{3}{2}\pi x), & -1 \leqslant x \leqslant 1 \\ 0 & \text{otherwise.} \end{cases} \end{cases} \quad (13)$$

Besides activations, all weights of networks also need to be quantized to binary values. We retrain the real-valued weight w and binarized weight w_b in the training process, and apply w_b to compute loss and gradient, which is used to update w. w is constrained between -1 to 1 to avoid excessive growth. Different from activations, the binarization function for w is not needed for the encoding function and can be directly defined as follows:

$$Binarize(x) = sign(HTanh(x)). \quad (14)$$

For this function, we define the gradient function of each component to constrain the search space. That is, the input of the sign function can be constrained to $[-1, +1]$ by $HTanh(x)$, and it can also speed up convergence. The parameters of the whole network are updated by *Adam* [42] in the condition of differentiability.

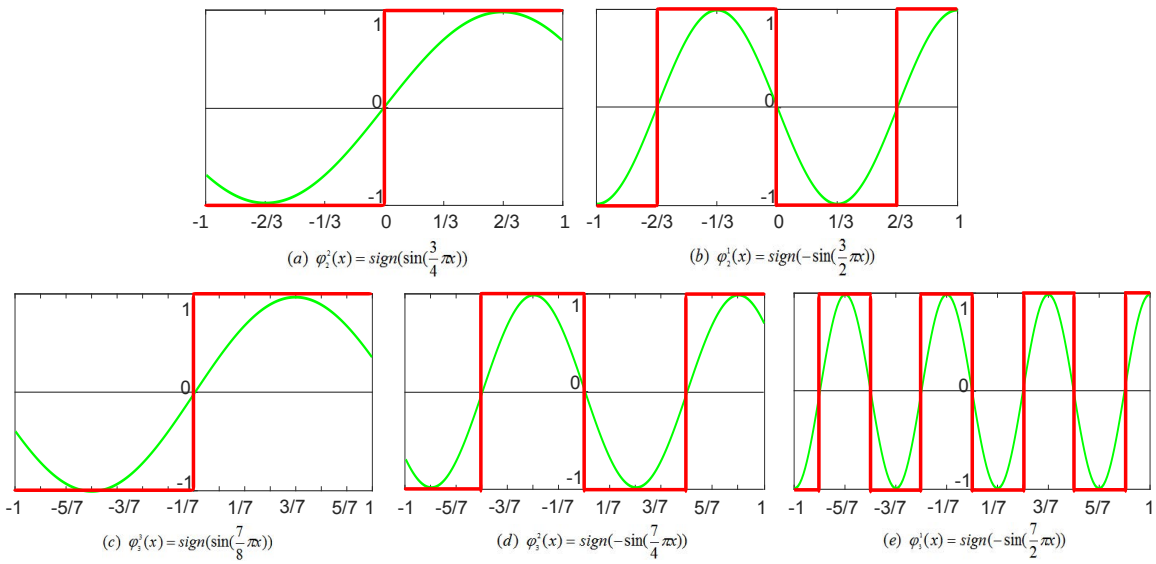


Fig. 2: Encoding functions. (a) and (b) denote the encoding functions of the second bit and the first bit of 2BitEncoder. (c), (d) and (e) denote the encoding functions of the third bit, the second bit and the first bit of 3BitEncoder.

4.2 Training Quantized Networks

The above training scheme is proposed to optimize binary networks, which can be converted into multi-state networks. However, this converter can produce many times more parameters than the original network. If we optimize the binarized network, it may easily fall into local optimal solutions and face slow convergence speed. Based on the affine mapping between quantized numbers and fixed-point integers, we can directly optimize the quantized network and then use multi-branch binary operations in the inference process. Two quantization schemes are usually applied in QNNs [17], [20], [29], named linear quantization and logarithmic quantization. Due to the requirement of our encoding mechanism, linear quantization is used to quantize our networks, and is defined as follows:

$$q_k(x) = 2\left(\frac{\text{round}((2^k-1)(\frac{x+1}{2}))}{2^k-1} - \frac{1}{2}\right) \tag{15}$$

where the rounding operation can be used to quantize a real number $x \in [-1, +1]$ to a certain state. We call it a hard ladder function, which can segment input space to multi-states. Table 2 lists the four states quantized by the formula (11). However, the derivative of this function is almost zero everywhere, so it cannot be used in the training process. Inspired by STE, we use the same technique to speed up computing process and yield better performance. We use the loss computed by quantized parameters to update full precision parameters. Note that for our encoding scheme with low-precision quantization (e.g., binary), we use *Adam* to train our model, otherwise *SGD* is used.

5 Experiments

In this section, we compare the performance of our method with some typical methods for image classification tasks on CIFAR-10 and ImageNet, object detection tasks on PASCAL VOC2007/2012, and semantic segmentation tasks on PASCAL VOC2012. In recent years, many scholars are devoted to improving the performance (e.g., accuracy and compression ratio) of QNNs, while very few researchers have studied their engineering acceleration, which is an important reason for hindering industrial promotion. Therefore, we mainly focus on an acceleration method, which is especially suitable for engineering applications.

5.1 Image Classification

CIFAR-10: CIFAR-10 is an image classification benchmark dataset, which has 50000 training images and 10000 testing images. All the images are 32×32 color images representing airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships and trucks.

We validated our method with different bit encoding schemes, in which activations and weights are equally treated, that is, both of them use the same number of encoding bits. Table 3 lists the results of our method and many state-of-the-art models, including BWN [33], BNN [17], XNOR-Net [11], XNOR-Net [11], TWN [18], DoReFa-Net [20], ABC-Net [12], INQ [21], and SYQ [43]. Here we use the same network architecture as in [33] and [17], except for the encoding functions. We use $HTanh(\cdot)$ as the activation function and employ *Adam* to optimize all parameters of the network. From all the results, we can see that the representation capabilities of our 1-bit and 2-bit encoded networks are enough for small-scale datasets, e.g., CIFAR-10. Our method with low-precision encoding (e.g., 2-bit) achieves nearly the same classification accuracy as its high precision versions (e.g., 8-bit) and full-precision models, while we can attain $\sim 16\times$ memory saving compared with its full-precision counterpart. When activations and weights are constrained to 1-bit, our network structure is similar to BNN [17], and our method yields even better accuracy mainly because of our proposed encoding functions.

ImageNet: We further compared the performance of our method with all the other methods mentioned above on the ImageNet (ILSVRC-2012) dataset [44]. This dataset consists

TABLE 3: Classification accuracies of Lenet on CIFAR-10 and ResNet-18 and ResNet-50 on ImageNet.

Model		Lenet	ResNet-18		ResNet-50	
Method	Compression ratio	CIFAR-10	ImageNet(Top-1)	ImageNet(Top-5)	ImageNet(Top-1)	ImageNet(Top-5)
BWN [33]	32x	90.10%	60.80%	83.00%	-	-
BNN [17]	32x	88.60%	42.20%	67.10%	-	-
XNOR-Net [11]	32x	-	51.20%	73.20%	-	-
TWN [18]	16x	92.56%	61.80%	84.20%	-	-
DoReFa-Net[1/4-bit] [20]	32x	-	59.20%	81.50%	-	-
ABC-Net[5-bit] [12]	6.4x	-	65.00%	85.90%	70.10%	89.70%
INQ[5-bit] [21]	6.4x	-	68.92%	89.10%	74.81%	92.45%
SYQ[2/8-bit] [43]	16x	-	67.70%	87.80%	72.30%	90.90%
Full-Precision	1x	91.40%	68.60%	88.70%	76.10%	93.30%
Encoded activations and weigjts						
MBBN(M=K=1)	32x	90.39%	47.10%	71.70%	55.26%	78.43%
MBBN(M=K=2)	16x	91.06%	56.30%	79.48%	62.22%	82.71%
MBBN(M=K=3)	10.7x	91.27%	58.69%	81.84%	66.81%	87.31%
MBBN(M=K=4)	8x	91.15%	59.57%	82.35%	71.32%	90.13%
MBBN(M=K=5)	6.4x	90.92%	65.09%	86.42%	73.72%	91.90%
MBBN(M=K=6)	5.3x	91.01%	67.04%	87.69%	74.99%	92.50%
MBBN(M=K=7)	4.6x	90.20%	68.37%	88.47%	75.44%	92.81%
MBBN(M=K=8)	4x	90.43%	68.63%	88.70%	76.02%	93.01%

of 1K categories images, and has over $1.2M$ images in the training set and $50K$ images in the validation set. We use Top-1 and Top-5 accuracies to report the classification performance. For large-scale training sets (e.g., ImageNet), it usually costs plenty of time and requires sufficient computing resources for classical full-precision models. Moreover, it will be more difficult to train quantized networks, thus the initialization of parameter values is particularly important. In this paper, we use $HReLU(\cdot)$ as the activation function to constrain activations. In particular, the full-precision model parameters activated by $ReLU(\cdot)$ can be directly used as the initialization parameters for our 8-bit quantized network. After a little number of fine-tuning, our 8-bit quantized networks can be well-trained. Similarly, we use the 8-bit model parameters as the initialization parameters to train 7-bit quantized networks, and so on. There is a special case, if we use $HReLU(\cdot)$ and 1BitEncoder function to encode activations, all the activations will be constrained to 1. Here, we use $HTanh(\cdot)$ as the activation function for 1-bit encoding. Note that we use *SGD* to optimize parameters when the number of encoding bits is not less than 3, and the learning rate is set to 0.1. When the number of the encoding bits is 1 or 2, the convergent speed of *Adam* is faster than *SGD*, as discussed in [11], [17].

Table 3 lists the performance (e.g., accuracy and compression ratio) of our method and many state-of-the-art models mentioned above. All these results show that our method with 1-bit encoding performs much better than BNN [17]. Similarly, our method with 5-bit encoding significantly outperforms ABC-Net[5-bit] [12]. Moreover, our networks can be trained in a very short time, and to achieve the accuracies in our experiments, only dozens of times fine tuning are needed. Of course, if we continue to train the network, we may get better performance. Different from BWN and TWN, whose weights are only quantized rather than activation values, our method quantifies both weights and activation values, simultaneously. Although BWN and TWN can obtain little higher accuracies than our method with 1-bit quantization model, our method obtains more speedups, and the speedup ratio of existing methods such as BWN and TWN is limited to $\sim 2\times$. Due to limited and fixed expression ability, existing methods (such as BWN, TWN, BNN and XNOR-Net) cannot satisfy higher precision requirements. In particular, our method can provide 64 available encoding choices, and hence our encoded networks with different encoding precisions have different speedup ratios, memory requirements and experimental precisions.

5.2 Object Detection

We also use our proposed method to decompose some trained networks with the Single Shot MultiBox Detector (SSD) framework [45] to solve object detection tasks, in which the coordinate regression task coexists with classification tasks. The regression task has higher demands on the encoding precision, therefore, the application of object detection presents a new challenge for QNNs.

In this experiment, all the models are trained on the PASCAL VOC2007 and VOC2012 trainval set, and tested on the VOC2007 test set. Here, we use the MobileNetV2 [9], InceptionV3 [46] and ResNet-50 [47] as the basic networks to extract features in the SSD framework. Note that we quantize the basic network layers, and keep the other layers in full-precision. All the networks are trained on a large 512×512 input image. In order to quickly get the network

TABLE 4: Comparison of object detection performance of our method with different encoding bits.

Backbone	Precision	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
InceptionV3	full(32-bit)	78.8	80.8	86.8	81.5	72.1	53.6	86.9	87.7	87.4	62.0	81.4	75.3	88.0	87.1	86.0	79.3	58.7	81.5	72.9	87.2	80.2
	M=K=8	78.0	79.0	85.8	79.8	72.4	49.8	83.5	86.8	86.5	60.7	84.2	75.4	87.5	86.6	84.4	78.9	57.5	81.9	73.3	86.7	78.8
	M=K=6	77.2	79.1	85.3	78.9	67.8	51.4	85.1	86.4	87.8	59.3	82.7	73.3	87.0	85.5	83.7	77.7	56.8	78.8	70.7	87.9	78.7
	M=K=4	66.6	71.1	76.2	62.7	59.2	36.0	75.0	80.9	76.2	40.8	71.6	67.4	73.8	78.8	74.6	71.1	41.9	64.9	63.2	79.3	66.9
	BWN	44.8	56.5	57.2	26.8	34.6	17.4	51.6	59.6	51.2	22.3	41.5	55.3	42.1	63.4	54.9	53.3	15.7	44.0	40.2	60.2	50.2
ResNet-50	full(32-bit)	77.2	79.5	85.0	81.2	71.1	55.3	83.7	87.3	87.3	60.4	84.5	72.6	85.0	83.5	82.9	80.3	50.7	78.7	72.4	85.4	77.6
	M=K=8	78.4	80.7	86.5	79.8	70.8	59.4	85.6	87.5	88.7	64.1	84.1	72.1	85.4	85.4	86.5	80.6	53.1	79.1	73.8	85.4	78.9
	M=K=6	77.1	78.6	84.8	77.9	70.0	55.6	86.8	87.0	86.9	61.6	82.6	72.5	81.1	84.1	83.6	80.5	50.4	80.1	73.1	85.5	78.3
	M=K=4	66.4	70.7	79.2	58.4	56.9	41.3	73.5	81.5	77.0	47.6	66.6	67.6	68.4	77.0	76.6	73.2	40.0	64.3	64.1	77.3	67.4
	BWN	45.6	53.1	49.8	30.0	37.9	20.4	53.7	69.4	47.8	23.4	44.0	51.5	47.8	53.6	59.3	57.9	21.9	42.6	36.4	61.2	49.9
MobileNetV2	full(32-bit)	72.6	77.8	82.6	71.2	64.4	48.5	79.0	86.0	81.4	54.0	77.9	65.1	79.3	79.3	78.7	75.5	49.5	77.3	65.3	84.5	74.0
	M=K=8	71.5	77.3	81.3	70.8	64.3	46.0	77.6	84.5	79.9	51.9	75.8	66.5	78.4	78.4	78.2	75.6	47.5	74.0	68.4	82.2	70.4
	M=K=6	70.5	77.2	81.0	68.5	60.4	45.3	77.1	84.4	79.1	51.8	73.4	65.2	77.9	77.9	77.3	75.1	46.2	72.3	66.5	81.1	71.4
	M=K=4	55.1	68.7	69.6	42.7	43.9	27.7	62.9	76.5	56.9	29.6	56.2	56.1	51.8	51.8	68.1	64.1	26.5	55.8	49.6	72.1	58.4
	BWN	21.8	34.7	18.6	9.3	16.6	9.1	34.3	34.3	20.4	7.5	12.9	26.3	16.0	35.5	32.2	26.2	9.1	15.9	18.3	34.1	24.0

Fig. 3: Detection results of our method with different precisions based on the InceptionV3 network on some examples of the VOC2007 test set.

TABLE 5: Comparison of object detection results of our method with different encoding bits.

Backbone	Precision	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
InceptionV3	M=8 K=4	69.9	70.4	78.0	68.5	59.8	40.7	79.2	82.1	83.3	48.7	72.8	74.6	77.6	80.0	81.1	72.3	41.4	67.0	68.2	80.3	71.3
	M=4 K=8	69.4	74.1	79.3	67.3	61.3	41.2	74.7	82.3	79.6	48.1	74.7	67.8	75.2	80.2	75.4	73.5	41.8	72.6	66.6	81.8	70.4
	M=6 K=3	67.6	80.8	86.6	79.7	73.2	53.8	84.4	86.9	87.6	60.4	83.5	76.3	87.8	86.2	85.5	78.1	56.0	79.9	70.8	87.0	77.8
	M=3 K=6	64.7	71.1	74.9	53.8	55.9	33.7	72.3	79.4	77.0	43.2	66.8	68.8	69.1	76.1	74.0	70.3	37.1	61.7	67.1	78.4	64.0

parameters of convergence, we adopt the same training strategy in the above experiments, in which high-bit model parameters are used to initialize low-bit models. We use the average precision (AP) to measure the performance of the detector on each category and mean average precision (mAP) to evaluate multi-target detector performance. Table IV shows the object detection results of our method and its counterparts, including its full-precision counterpart and BWN. It is clear that our method with 8-bit encoding scheme can yield very similar performance as their full-precision counterparts. Especially for ResNet-50, we achieve the best results of 78.4% mAP in 8-bit precision that is better than its full-precision counterpart by 1.2% mAP. When we use 6-bit to encode both activation values and weights, the evaluation index drops slightly. If the number of encode bits is constrained to 4, the performance of this task has visibly deteriorated. Especially for the MobileNetV2, when the number of the encoding bits is converted to 4 from 6, the mAP result drops monotonically from 70.5% to 55.1%. For some categories (e.g., bottle, chair and plant), the AP results dramatically drop. In our experiments, we also implemented the binary weight form of different basic networks. Obviously, our method with 4-bit encoding has a higher accuracy advantage over BWN [33]. For object detection tasks, fixed extremely low-bit precision network seems powerless. Especially for the MobileNetV2, BWN achieves the result of 21.8% mAP that is not suitable for industrial applications.

To compare the performance of different models with different precisions more conveniently, Fig. 3 shows some detection results on the examples of the VOC2007 test set with different precisions based on the InceptionV3 network. Detections are shown with scores higher than 0.5. The detection output pictures, which are tested by different precisions, are shown in the same column. From all the results, we can see that 8-bit and 6-bit encoding precisions have similar performance as their full-precision counterparts. There is almost no obvious difference between those detection examples. However, when the encoding precision is constrained to 4-bit, we can see that some offsets are produced on the bounding box (e.g., bike). Meanwhile, some small objects and obscured targets may be missed (e.g., cow and sheep).

In fact, it's not necessary to constrain both activation values and weights to the same bit precision. Table 5 shows the results of our method with four different bit precisions based on the InceptionV3 network. For the first case, we constrain activation values and weights to 8-bit and 4-bit, respectively. For the second case, we constrain activation values and weights to 4-bit and 8-bit, respectively. The two cases have the same computational complexity, however, they have different compression ratios and results. The results of the third case, whose activation values and weights are constrained to 6-bit and 3-bit, respectively, are obviously better than the fourth case, whose activation values and weights are constrained to 3-bit and 6-bit, respectively. Therefore, the diversity of activation values is more important than weights. In order to mitigate the performance degradation of QNNs, we should encode activation values with higher precision than weights to achieve better performance.

For the object detection task, our method with a relatively low encoding precision (e.g., 6-bit) yields good performance on the SSD framework. Similarly, it can be applied to other frameworks, e.g., R-CNN [48], Fast R-CNN [49], SPP-Net [50] and YOLO [51].

5.3 Semantic Segmentation

As a typical task in the field of unmanned driving, image semantic segmentation has attracted many scholars and companies to invest in research. The application scenario for this task usually relies on small devices, therefore, the model compression and computational acceleration seems particularly important for real-world applications. We evaluate the performance of our method on the published PASCAL VOC2012 semantic segmentation benchmark, which is consisted of 20 foreground object classes and one background class. We use pixel accuracy (pixAcc) and mean IoU (mIoU) as the metrics to report the performance.

We use the DeepLabV3 [52] based on ResNet-101 [47] as the basic network in this experiment. Similarly, we use different encoding precisions to evaluate the semantic segmentation performance of our method. Fig. 5 shows the evaluating indicators of different encoding precisions. The dotted line denotes the mIoU of the full-precision network, and the solid line denotes the pixAcc of the full-precision network. From the other two curves, we can see that with the increase of encoding precisions, the semantic segmentation performance is getting better and better. When the encoding precision is 7-bit, our method achieves the best performance (98.9% mIoU and 95.2% pixAcc), which is much better than full-precision counterpart (98.5% mIoU and 92.9% pixAcc). Fig. 4 shows some visualization results on the VOC2012 val set with different encoding precisions. Obviously, the segmentation results can be more precise by using higher encoding bits. For example, from the fourth image we can see that as the increase of encoding precision the detail of segmentation of horse's legs is getting more and more clear.

In this experiment, we evaluated the performance of our method based on the DeepLabV3 framework. However, it can also be applied to FCN [53] and PSP [54]. Besides semantic segmentation tasks, our method can be applied to other real-word tasks, e.g., the instance segmentation task.

5.4 Efficiency Analysis

Courbariaux et al. [17] have used a $8192\times8192\times8192$ matrix multiplication on a GTX750 Nvidia GPU to improve the performance of XNOR kernel, and have achieved $23\times$ speedup ratio than the baseline kernel. Therefore, we will use the same matrix multiplication to validate the performance of our method by theoretical analysis and experiment.

In this subsection, we analyze the speed-up ratios of our method with different encoding bits. Suppose there are two vectors $x = [x^1, x^2, ..., x^{64}]$ and $w = [w^1, w^2, ..., w^{64}]$, which are composed by 64 real numbers. If we use the 32-bit floating-point data type to represent each element of the two vectors, the element-wise vector multiplication should require 64 multiplications. However, after each element of the two vectors is quantized to a binary state, we use 1-bit to denote their elements, and can use two 64-bit data type to represent those two vectors. Then the element-wise multiplication of those two vectors can be computed at once.

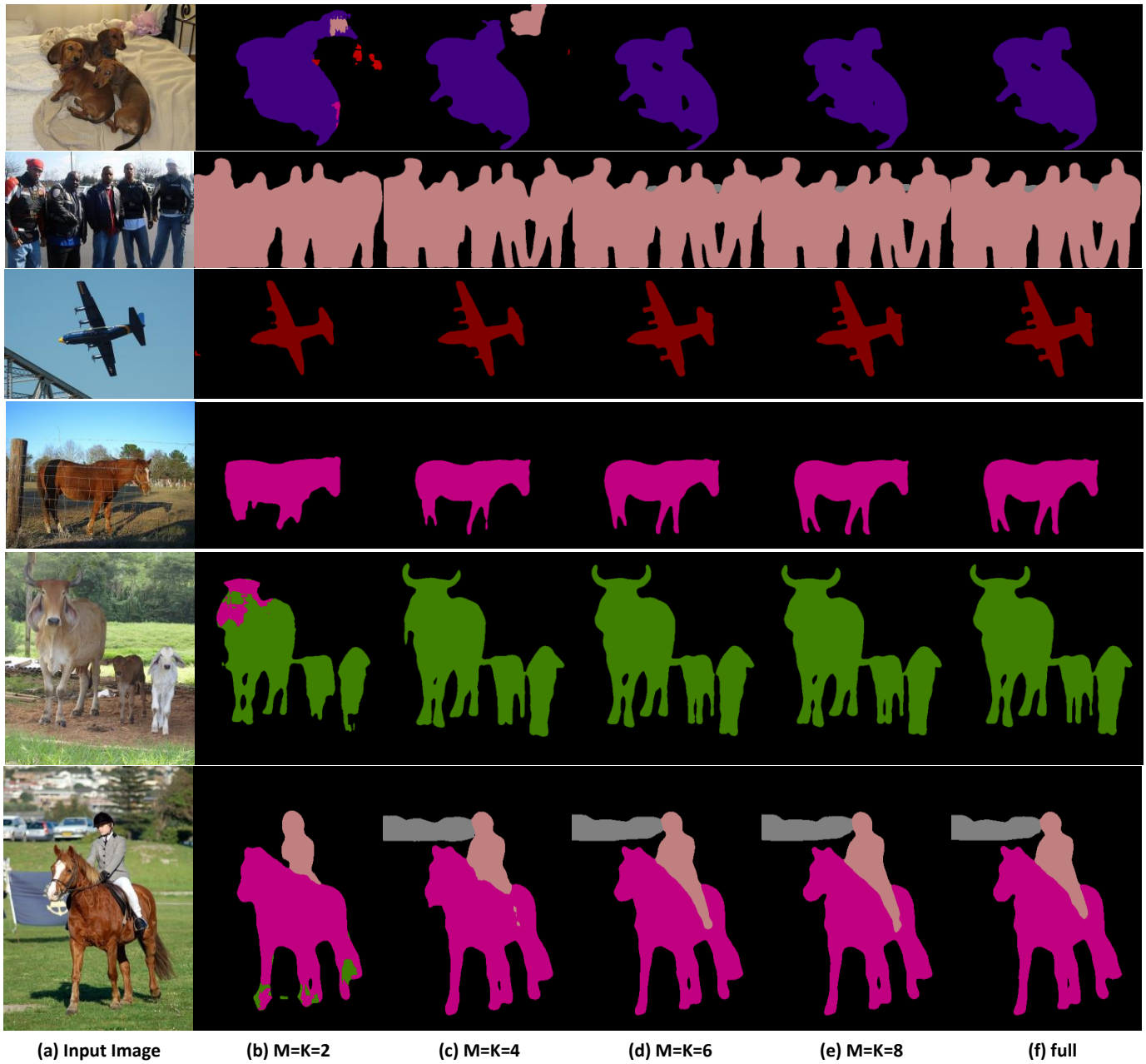


Fig. 4: Visualization results on the VOC2012 val set with different encoding precisions.

The details are shown in Fig. 6. The general computing platform (i.e., CPU and GPU) can perform a 64-bit binary operation in one clock cycle. In addition, an Intel SSE (e.g., AVX and AVX-512) instruction can perform 128 (or 256, 512) bits binary operations. This mechanism is very suitable for the FPGA platform, which is designed for arithmetic and logical operations. Therefore, our method can achieve model compression, computational acceleration and resource saving in the inference process.

We use S_{MK} to denote the speedup ratio of our method by constraining activation values to M-bit and the values of weights to K-bit, where $M, K \in \{1, 2, ..., 8\}$. S_{MK} can be formulated as follows:

$$S_{MK} = \frac{N\gamma}{MK(\gamma + 2\lceil \frac{N}{L} \rceil) + (MK - 1)\beta}, \tag{16}$$

where γ denotes the speedup ratio of multiply-accumulate operation (MAC) compared with bitwise operations, and β denotes the speedup ratio of addition compared with bitwise operations. If we use $O(1)$ to represent the time complexity of 1-bit operations, we can use $O(MK)$ to denote the time complexity of our method by constraining activation values to M-bit and the values of weights to$K$-bit. Following TBN [19], we can know that $\gamma = 1.91$ and $L = 64$. According to the speedup ratio achieved by BWN [33], we can safely assume $\beta = \gamma/2$. Thus, the matrix multiplication after encoded by 2-bit can obtain at most $\sim 15.13\times$ speedups over its full-precision counterpart.

Besides the theoretical analysis, we also implemented the multiplication of two matrixes via our $\{-1, +1\}$ encoding scheme on GTX 1080 GPU. The experiment design and

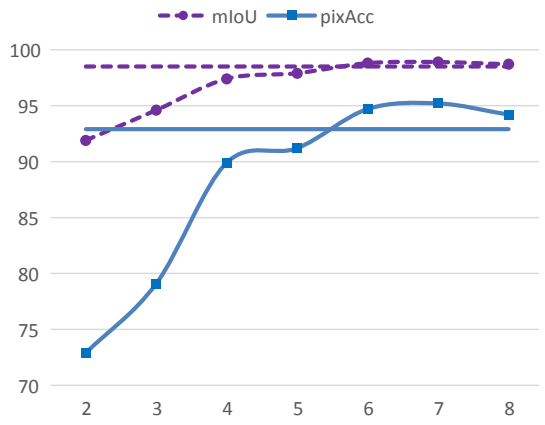


Fig. 5: The evaluating indicators of different encoding precisions for semantic segmentation. The dotted line and solid line denote the results of mIoU and pixAcc, respectively.

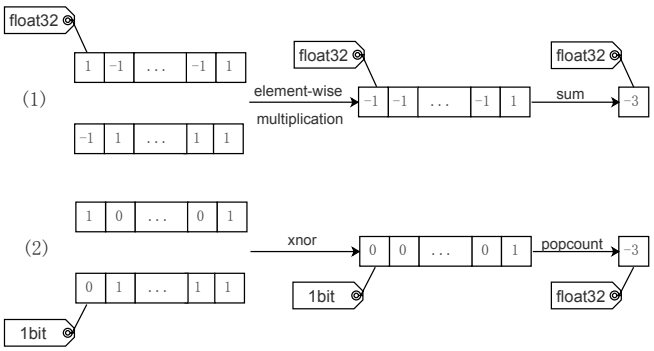


Fig. 6: The computing process of dot produce based on common multiplication and bitwise operations.

analysis is referred to paper [17]. In our experiments, we first get the encoding numbers of two matrices, and then store those numbers by bits. The matrix multiplication after encoded by 2-bit has obtained at most $\sim 15.89\times$ speedup ratio than the baseline kernel (a quite unoptimized matrix multiplication kernel). Because of the parallel architecture of GPU, the experimental result is slightly higher than theoretical speedup ratio. Fig. 7 shows the speedup ratios of different encoding precisions (from 1-bit to 6-bit), where

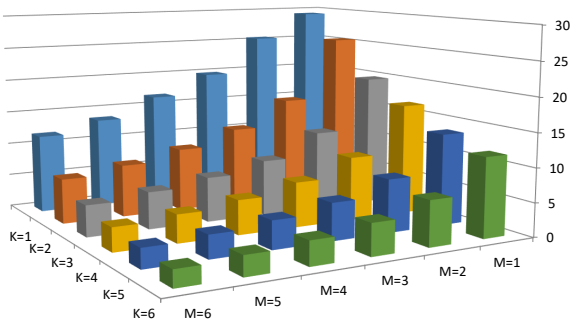


Fig. 7: The speedup ratios of different encoding precisions (from 1-bit to 6-bit).

M and K denote the number of encoding bits. From all the results, we can see that the computational complexity is increasing gradually with the increase of encoding bits. Thus, users can easily achieve different encoding precisions arbitrarily according to their requirements (e.g., accuracy and speed) and hardware resources (e.g., memory). Code is available at: https://github.com/qigongsun/BMD.

6 Discussion and Conclusion

6.1 $\{0,1\}$ Encoding and $\{-1,+1\}$ Encoding

As described in [20], there exists an affine mapping between quantized numbers and fixed-point integers. The quantized numbers are usually restricted to the closed interval $[-1,1]$. For example, the mapping is formulated as follows:

$$\mathrm{x}^q = \frac{2}{2^M-1}\mathrm{x}^{\{0,1\}} - 1, \tag{17}$$

where x^q denotes a quantized number and $\mathrm{x}^{\{0,1\}}$ denotes the fixed-point integer encoded by 0 and 1. We use a K-bit fixed-point integer to represent a quantized number w^q. The product can be formulated as follows:

$$\begin{aligned}\mathrm{x}^q \cdot \mathrm{w}^q = &\frac{4}{(2^M-1)(2^K-1)}\mathrm{x}^{\{0,1\}}\cdot\mathrm{w}^{\{0,1\}} - \\ &\frac{2}{2^M-1}\mathrm{x}^{\{0,1\}} - \frac{2}{2^K-1}\mathrm{w}^{\{0,1\}} + 1.\end{aligned} \tag{18}$$

The right-hand side of (18) is a polynomial, which has four terms. And each term has its own scaling factor. The computation of $\mathrm{x}^{\{0,1\}}\cdot\mathrm{w}^{\{0,1\}}$ can be accelerated by bitwise operations, however, the polynomial and scaling factor will increase the computational complexity.

For our proposed quantized binary encoding scheme (i.e., $\{-1,+1\}$), the product of two numbers is defined as

$$\mathrm{x}^q \cdot \mathrm{w}^q = \frac{1}{(2^M-1)(2^K-1)}\mathrm{x}^{\{-1,+1\}}\cdot\mathrm{w}^{\{-1,+1\}}, \tag{19}$$

where $\mathrm{x}^{\{-1,+1\}}$ and $\mathrm{w}^{\{-1,+1\}}$ denote the fixed-point integers encoded by -1 and $+1$. Obviously, compared with the above encoding of $\{0,1\}$, the product can be more efficiently calculated by using our proposed encoding scheme.

6.2 Linear Approximation and Quantization

As described in [12], [25], [26], the weight w can be approximated by the linear combination of K binary subitems $\{\mathrm{w}_1,\mathrm{w}_2,...,\mathrm{w}_K\}$ and $\mathrm{w}_i \in \{-1,+1\}^N$, which can replace arithmetic operations with more efficient bitwise operations. In order to obtain the combination, we need to solve the following problem

$$\min_{\{\alpha_i,\mathrm{w}_i\}_{i=1}^K} \left\| \mathrm{w} - \sum_{i=1}^{K}\alpha_i\mathrm{w}_i \right\|^2, \ \mathrm{w}\in\mathbb{R}^N. \tag{20}$$

When this approximation is used in neural networks, w_i can be considered as model weights. However, the scaling factor α_i is introduced in this approximation, and such a scheme will expand the parameters K times. Therefore, this approximation can convert the original model to a complicated binary network, which is hard to train [55] and easily falls into local optimal solutions.

For our method, we use the quantized parameters w^q to approximate w as follows:

$$\mathrm{w} \approx \frac{1}{2^K-1}\mathrm{w}^q,\ \mathrm{w} \in [-1,+1]^N, \tag{21}$$

where w^q is a positive or negative odd number, and its absolute value is not larger than 2^K-1. Different from the above linear approximation, our method can achieve the quantized weights, and directly get the corresponding encoding elements. Thus, our networks can be more efficiently trained via our quantization scheme than the linear approximation.

6.3 Conclusions

In this paper, we proposed a novel encoding scheme of using $\{-1,+1\}$ to decompose QNNs into multi-branch binary networks, in which we used bitwise operations (*xnor* and *bitcount*) to achieve model compression, acceleration and resource saving. In particular, we can use the high-bit model parameters to initialize a low-bit model and achieve good results in various applications. Thus, users can easily achieve different encoding precisions arbitrarily according to their requirements (e.g., accuracy and speed) and hardware resources (e.g., memory). This special data storage and calculation mechanism could yield great performance in FPGA and ASIC, and thus our mechanism is feasible for smart chips. Future works will focus on improving the hardware implementation, and exploring some ways to automatically select proper bits for various network architectures (e.g., VGG and ResNet).

7 Acknowledgments

This work was partially supported by the State Key Program of National Natural Science of China (No. 61836009), Project supported the Foundation for Innovative Research Groups of the National Natural Science Foundation of China (No. 61621005), the National Natural Science Foundation of China (Nos. U1701267, 61871310, 61573267, 61502369, 61876220, 61876221, 61473215, 61571342, 61625301, and 61731018), the Fund for Foreign Scholars in University Research and Teaching Programs (the 111 Project) (No. B07048), the Major Research Plan of the National Natural Science Foundation of China (Nos. 91438201 and 91438103), National Basic Research Program of China (973 Program) (No. 2015CB352502), the Program for Cheung Kong Scholars and Innovative Research Team in University (No. IRT_15R53), and the Science Foundation of Xidian University (No. 10251180018).

References

[1] Q. Sun, F. Shang, K. Yang, X. Li, Y. Ren, and L. Jiao, "Multi-precision quantized neural networks via encoding decomposition of {-1,+1}," in *Proc. 33rd AAAI Conf. Artif. Intell.*, 2019.

[2] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Proc. Adv. Neural Inf. Process. Syst.*, 1993, pp. 164–171.

[3] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.

[4] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[5] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 1269–1277.

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

[7] C. Tai, T. Xiao, Y. Zhang, X. Wang *et al.*, "Convolutional neural networks with low-rank regularization," *arXiv preprint arXiv:1511.06067*, 2015.

[8] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[9] 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.*, 2018, pp. 4510–4520.

[10] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[11] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: Imagenet classification using binary convolutional neural networks," in *Proc. 14th Eur. Conf. Comput. Vis.*, 2016, pp. 525–542.

[12] X. Lin, C. Zhao, and W. Pan, "Towards accurate binary convolutional neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 345–353.

[13] J.-H. Luo, H. Zhang, H.-Y. Zhou, C.-W. Xie, J. Wu, and W. Lin, "ThiNet: Pruning CNN filters for a thinner net," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2018.

[14] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 448–456.

[15] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[16] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1737–1746.

[17] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4107–4115.

[18] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," in *Proc. Int. Conf. Learn. Representations*, 2017.

[19] D. Wan, F. Shen, L. Liu, F. Zhu, J. Qin, L. Shao, and H. T. Shen, "TBN: Convolutional neural network with ternary inputs and binary weights," *Proc. 15th Eur. Conf. Comput. Vis.*, vol. 1, no. 2, pp. 0–6, 2018.

[20] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[21] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044*, 2017.

[22] D. Zhang, J. Yang, D. Ye, and G. Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *Proc. 15th Eur. Conf. Comput. Vis.*, 2018, pp. 365–382.

[23] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "FP-BNN: Binarized neural network on FPGA," *Neurocomput*, vol. 275, 2017.

[24] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," in *Proc. Adv. Neural Inf. Process. Syst. Workshop*, vol. 1, 2011, p. 4.

[25] Y. Guo, A. Yao, H. Zhao, and Y. Chen, "Network sketching: Exploiting binary structure in deep CNNs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 4040–4048.

[26] C. Xu, J. Yao, Z. Lin, W. Ou, Y. Cao, Z. Wang, and H. Zha, "Alternating multi-bit quantization for recurrent neural networks," *arXiv preprint arXiv:1802.00150*, 2018.

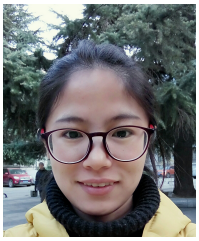
[27] P. Wang, Q. Hu, Y. Zhang, C. Zhang, Y. Liu, J. Cheng *et al.*, "Two-step quantization for low-bit neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4376–4384.

[28] B. Zhuang, C. Shen, M. Tan, L. Liu, and I. Reid, "Towards effective low-bitwidth convolutional neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 7920–7928.

[29] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," *arXiv preprint arXiv:1603.01025*, 2016.
[30] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," in *Proc. Int. Conf. Learn. Representations*, 2017.
[31] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2704–2713.
[32] X. Geng, J. Fu, B. Zhao, J. Lin, M. M. S. Aly, C. Pal, and V. Chandrasekhar, "Dataflow-based joint quantization of weights and activations for deep neural networks," *arXiv preprint arXiv:1901.02064*, 2019.
[33] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.
[34] P. Wang and J. Cheng, "Fixed-point factorized networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 4012–4020.
[35] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2849–2858.
[36] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044*, 2017.
[37] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.
[38] A. Mishra and D. Marr, "Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy," *arXiv preprint arXiv:1711.05852*, 2017.
[39] A. Polino, R. Pascanu, and D. Alistarh, "Model compression via distillation and quantization," *arXiv preprint arXiv:1802.05668*, 2018.
[40] Q. Hu, P. Wang, and J. Cheng, "From hashing to cnns: Training binary weight networks via hashing," in *Proc. 32rd AAAI Conf. Artif. Intell.*, 2018.
[41] Q. Hu, G. Li, P. Wang, Y. Zhang, and J. Cheng, "Training binary weight networks via semi-binary decomposition," in *Proc. 15th Eur. Conf. Comput. Vis.*, 2018, pp. 637–653.
[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[43] J. Faraone, N. Fraser, M. Blott, and P. H. Leong, "Syq: Learning symmetric quantization for efficient deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4300–4309.
[44] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.
[45] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *Proc. 14th Eur. Conf. Comput. Vis.*, 2016, pp. 21–37.
[46] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.
[47] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. 14th Eur. Conf. Comput. Vis.* Springer, 2016, pp. 630–645.
[48] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 580–587.
[49] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 91–99.
[50] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *Proc. 13th Eur. Conf. Comput. Vis.*, 2014, pp. 346–361.
[51] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 779–788.
[52] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, 2018.
[53] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 640–651, 2014.
[54] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2881–2890.
[55] H. Li, S. De, Z. Xu, C. Studer, H. Samet, and T. Goldstein, "Training quantized nets: A deeper understanding," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5811–5821.

Qigong Sun (S'15) received the B.S. degrees in intelligence science and technology from Xidian University, Xi'an, China in 2015. He is currently pursuing the Ph.D. degree in circuit and system from Xidian University, Xi'an China.

Currently, he is a member of Key Laboratory of Intelligent Perception and Image Understanding of Ministry of Education, and International Research Center for Intelligent Perception and Computation, Xidian University, Xi'an, China. His research interests include deep learning and image processing.



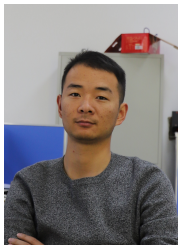
Xiufang Li received the master's degree in safety science and engineering from Xi'an University of Science and Technology, Xi'an, China in 2017. She is currently pursuing the Ph. D. degree in circuit and system from Xidian University, Xi'an China.

Currently, she is a member of Key Laboratory of Intelligent Perception and Image Understanding of Ministry of Education, and international Research Center for Intelligent Perception and computation, Xidian University, Xi'an China. Her research interests include deep learning and images processing.

Fanhua Shang (M'14) received the Ph.D. degree in Circuits and Systems from Xidian University, Xi'an, China, in 2012.

He is currently a professor with the School of Artificial Intelligence, Xidian University, China. Prior to joining Xidian University, he was a Research Associate with the Department of Computer Science and Engineering, The Chinese University of Hong Kong. From 2013 to 2015, he was a Post-Doctoral Research Fellow with the Department of Computer Science and Engineering, The Chinese University of Hong Kong. From 2012 to 2013, he was a Post-Doctoral Research Associate with the Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA. His current research interests include machine learning, big data, deep learning, and computer vision.

Hongying Liu (M'10) received her B.E. and M.S. degrees in Computer Science and Technology from Xi'An University of Technology, China, in 2006 and 2009, respectively, and Ph.D. in Engineering from Waseda University, Japan in 2012. Currently, she is a faculty member at the School of Artificial Intelligence, and also with the Key Laboratory of Intelligent Perception and Image Understanding of Ministry of Education, Xidian University, China. She is a member of IEEE. Her major research interests include image processing, intelligent signal processing, machine learning, etc.



Kang Yang received the B.S. degrees in electronic and information engineering from Xianyang Normal University, Xianyang, China in 2014. He is currently pursuing the M.S. degree in Electronics and Communication Engineering from Xidian University, Xi'an, China. Currently, he is a member of Key Laboratory of Intelligent Perception and image Understanding of Ministry of Education, and International Research Center for Intelligent Perception and Computation, Xidian University, Xi'an. His research interests include deep learning and object detection.

Licheng Jiao (SM'89-F'17) received the B.S.degree from Shanghai Jiaotong University, Shanghai, China, in 1982 and the M.S. and Ph.D. degree from Xi'an Jiaotong University, Xi'an, China, in 1984 and 1990, respectively.

Since 1992, he has been a Professor with the school of Electronic Engineering, Xidian University, Xi'an, where he is currently the Director of Key Laboratory of Intelligent Perception and Image Understanding of the Ministry of Education of China. He is in charge of about 40 important scientific research projects and has published more than 20 monographs and a hundred papers in international journals and conferences. His research interests include image processing, natural computation, machine learning, and intelligent information processing.

Dr. Jiao is a member of the IEEE Xi'an Section Execution Committee, the Chairman of the Awards and Recognition Committee, the Vice Board Chairperson of the Chinese Association of Artificial Intelligence, a Councilor of the Chinese Institute of Electronics, a committee member of the Chinese Committee of Neural Networks, and an expert of the Academic Degrees Committee of the State Council. He is a fellow of the IEEE.

Zhouchen Lin (SM'08-F'17) received the PhD degree in applied mathematics from Peking University in 2000.

He is currently a professor with the Key Laboratory of Machine Perception, School of Electronics Engineering and Computer Science, Peking University. His research interests include computer vision, image processing, machine learning, pattern recognition, and numerical optimization. He is an area chair of CVPR 2014/2016/2019, ICCV 2015, NIPS 2015/2018/2019, and AAAI 2019, and a senior program committee member of AAAI 2016/2017/2018 and IJCAI 2016/2018/2019. He is an associate editor of the IEEE Transactions on Pattern Analysis and Machine Intelligence and the International Journal of Computer Vision. He is a fellow of the IAPR and the IEEE.