TBN: Convolutional Neural Network with Ternary Inputs and Binary Weights

Diwen Wan[1,2], Fumin Shen[1(✉)], Li Liu[2], Fan Zhu[2], Jie Qin[3], Ling Shao[2], and Heng Tao Shen[1]

[1] Center for Future Media and School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China
funmin.shen@gmail.com
[2] Inception Institute of Artificial Intelligence, Abu Dhabi, UAE
[3] Computer Vision Lab, ETH Zurich, Zurich, Switzerland

Abstract. Despite the remarkable success of Convolutional Neural Networks (CNNs) on generalized visual tasks, high computational and memory costs restrict their comprehensive applications on consumer electronics (*e.g.*, portable or smart wearable devices). Recent advancements in binarized networks have demonstrated progress on reducing computational and memory costs, however, they suffer from significant performance degradation comparing to their full-precision counterparts. Thus, a highly-economical yet effective CNN that is authentically applicable to consumer electronics is at urgent need. In this work, we propose a Ternary-Binary Network (TBN), which provides an efficient approximation to standard CNNs. Based on an accelerated ternary-binary matrix multiplication, TBN replaces the arithmetical operations in standard CNNs with efficient **XOR**, **AND** and **bitcount** operations, and thus provides an optimal tradeoff between memory, efficiency and performance. TBN demonstrates its consistent effectiveness when applied to various CNN architectures (*e.g., AlexNet and ResNet*) on multiple datasets of different scales, and provides ∼32× memory savings and 40× faster convolutional operations. Meanwhile, TBN can outperform XNOR-Network by up to 5.5% (top-1 accuracy) on the ImageNet classification task, and up to 4.4% (mAP score) on the PASCAL VOC object detection task.

Keywords: CNN · TBN · Acceleration · Compression
Binary operation

1 Introduction

Along with the overwhelming success of deep learning, convolutional neural networks (CNNs) have demonstrated their capabilities in various computer vision

Electronic supplementary material The online version of this chapter (https://doi.org/10.1007/978-3-030-01216-8_20) contains supplementary material, which is available to authorized users.

V. Ferrari et al. (Eds.): ECCV 2018, LNCS 11206, pp. 322–339, 2018.
https://doi.org/10.1007/978-3-030-01216-8_20

tasks [8,13,14,18,29,31,37–39,41,43,45–47,49]. Effective CNNs normally contain millions of weight parameters, and require billions of high precision operations to be performed for a single classification task. Consequently, either training a CNN model with large-scale data or deploying a CNN model for real-time prediction task has rigid hardware demands (such as GPUs and TPUs) for the overheads in both storage and computing. However, such rigid hardware demands restrict CNNs' comprehensive applications on consumer electronics, such as virtual reality (VR), augmented reality (AR), smart wearable devices and self-driving cars. Significant research efforts have been paid on how to reduce the computational and memory costs of CNNs. Existing CNN lightweighting techniques include pruning [16,17,53], quantization [5,15,35,36,48,54,56], factorization [2,21,23,26,30,51,55], network binarization [7,42], distilling [19] and others [10,54]. Since network binarization can lower 32-bit full-precision values down to 1-bit binary values and allow efficient convolution operations, it has the most potentials in lightweighting the network for practical usages on portable devices. Recent progresses in binarized CNNs [7,42] have provided evidences of successfully reducing the computational and memory costs with the binary replacement. In CNNs, two types of values can be binarized: (1) network weights in the convolutional layers and the fully-connected layers, and (2) input signals to the convolutional layers and the fully-connected layers. Binarizing the network weights can directly result in $\sim$32$\times$ memory saving over the real-valued counterparts, and meanwhile bring $\sim$2$\times$ computational efficiency by avoiding the multiplication operation in convolutions. On the other hand, simultaneously binarizing both weights and the input signals can result in 58$\times$ computational efficiency by replacing the arithmetical operations in convolutions with XNOR and bitcount operations. Despite the significant cost reduction, noticeable accuracy degradation of the binarized CNNs introduced new performance issues for practical usages. Undoubtedly, such performance degradation is due to the quantization errors when brutally binarizing real-values in both network weights and layer-wise inputs.

In this work, we aim to improve the performance of binarized CNNs with ternary inputs to the convolutional and fully-connected layers. The ternary inputs constrain input signal values into -1, 0, and 1, and can essentially reduce the quantization error when binarizing full-precision input signals. By incorporating ternary layer-wise inputs with binary network weights, we propose a Ternary-Binary Network (TBN) that provides an optimal tradeoff between the performance and computational efficiency. The illustration of the pipeline of proposed TBN approach can be found in Fig. 1. In addition, an efficient threshold-based approximated solution is introduced to minimize the quantization error between the full-precision network weights and the binary weights along with a scaling factor, and an accelerated ternary-binary dot product method is introduced using simple bitwise operations (*i.e.*, **XOR** and **AND**) and the **bitcount** operation. Specifically, TBN can provide $\sim$32$\times$ memory saving and 40$\times$ speedup over its real-valued CNN counterparts. Comparing to XNOR-Network [42], with an identical memory cost and slightly sacrificed efficiency, TBN can outperform

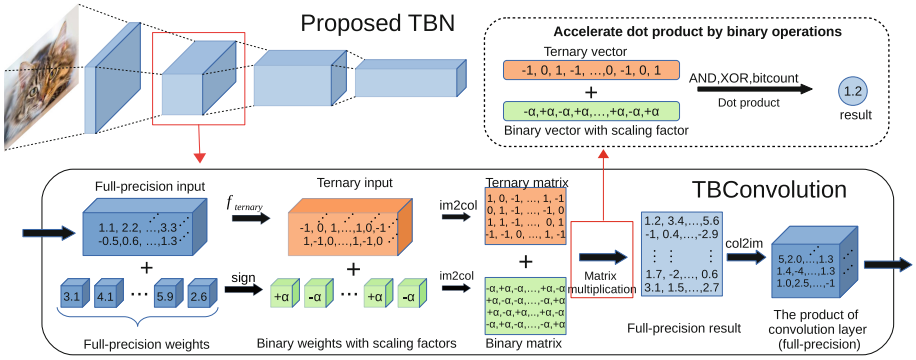

Fig. 1. The illustration of TBN, which is an efficient variation of convolutional neural networks. The weight filters of TBN have binary values and its inputs contain ternary values. The binary operations is used to accelerate the convolution operation. Our TBN is not only efficient in terms of memory and computation but also has good performance. This helps to use CNNs on resource-limited portable devices.

XNOR-Network on both image classification task and object detection task. The main contributions of this paper can be summarized as follows:

- We propose a Ternary-Binary Network, which for the first time elegantly incorporates the ternary layer-wise inputs with binary weights and provides an optimal tradeoff between the performance and computational efficiency.
- We introduce an accelerated ternary-binary dot product method that employs simple **XOR**, **AND** and **bitcount** operations. As a result, TBN can provide $\sim 32\times$ memory saving and $40\times$ speedup over its real-valued CNN counterparts.
- By incorporating with various CNN architectures (including *LeNet-5*, *VGG-7*, *AlexNet*, *ResNet-18* and *ResNet-34*), TBN can achieve the promising image classification and object detection performance on multiple datasets among quantized neural networks. Particularly, TBN outperforms XNOR-Network [42] by up to 5.5% (top-1 accuracy) on the ImageNet classification task, and up to 4.4% (mAP score) on the PASCAL VOC object detection task.

2 Related Work

Abundant research efforts have been paid on how to lightweight standard full-precision CNNs through quantization. In this section, we list some recent relevant work along such a research line, and discuss how they relate to the proposed TBN. We roughly divide these work into the following two categories: (1) quantized weights with real-value inputs and (2) quantized weights and inputs.

Quantized Weights with Real-Value Inputs: Networks with quantized weights can result in a direct reduction in network sizes, however, the efficiency

Table 1. Comparisons between TBN and its closely related methods in terms of input and weight types, numbers of multiply-accumulate operations (MACs) and binary operations required by matrix multiplication, speedup ratios and operation types. More detailed statistical analysis can be found in Sect. 4.3.

Methods		Inputs	Weights	MACs	Binary operations	Speedup	Operations
	Full-precision	$\mathbb{R}$	$\mathbb{R}$	$n \times m \times q$	0	1×	+, x
Quantize weights	TTQ [58]	$\mathbb{R}$	$\{-\alpha^n, 0 + \alpha^p\}$	$n \times m \times q$	0	$\sim 2\times$	+, −
	TWN [33]	$\mathbb{R}$	$\{-\alpha, 0, -\alpha\}$	$n \times m \times q$	0	$\sim 2\times$	+, −
	BWN [42]	$\mathbb{R}$	$\{-\alpha, +\alpha\}$	$n \times m \times q$	0	$\sim 2\times$	+, −
	BC [6]	$\mathbb{R}$	$\{-1, +1\}$	$n \times m \times q$	0	$\sim 2\times$	+, −
Quantize inputs and weights	TNN [1]	$\{-1, 0, 1\}$	$\{-1, 0, 1\}$	0	$8\times n\times m\times q$	15×	AND, bitcount
	GXNOR [9]	$\{-1, 0, 1\}$	$\{-1, 0, 1\}$	0	$5\times n\times m\times q$	15×	AND, bitcount
	BNN [7]	$\{-1, +1\}$	$\{-1, +1\}$	0	$2\times n\times m\times q$	64×	XOR, bitcount
	XNOR [42]	$\{-\beta, +\beta\}$	$\{-\alpha, +\alpha\}$	$2 \times n \times m$	$2\times n\times m\times q$	58×	XOR, bitcount
	HORQ [34]	$\{-\beta, +\beta\} \times 2$	$\{-\alpha, +\alpha\}$	$4 \times n \times m$	$4\times n\times m\times q$	29×	XOR, bitcount
	DoReFa* [57]	$\{0, 1\} \times 2$	$\{0, 1\}$	0	$4\times n\times m\times q$	30×	AND, bitcount
	TBN	$\{-1, 0, +1\}$	$\{-\alpha, +\alpha\}$	$n \times m$	$3\times n\times m\times q$	40×	AND, XOR, bitcount

*We adopt DoReFa Network with 1-bit weight, 2-bit activation.

improvement, which is achieved by avoiding the multiplication operation, is limited if the input signals remain real-valued. The most basic forms of weight quantization either directly constrains the weight values into the binary space $\{-1, 1\}$, *e.g.,* BinaryConnect (BC [6]), or constrain the weight values along with a scaling factor $\{-\alpha, \alpha\}$, *e.g.,* Binary-Weight-Networks(BWNs [42]). Beyond the binary weights, ternary weights are introduced to reduce the quantization error. Ternary Weight Networks (TWNs [33]) quantize the weights into $\{-\alpha, 0, \alpha\}$, while Trained Ternary Quantization (TTQ [58]) achieves better performance by constraining the weights to asymmetric ternary values $\{-\alpha^n, 0, \alpha^p\}$.

Quantized Weights and Inputs: Comparing to the storage, the computational efficiency is a more critical demand for real-time predictions in resource-constrained environments. Since quantizing input signals can potentially replace arithmetical operations with XNOR and bitcount operations and improve the efficiency, networks that attempted to quantize both network weights and layer-wise input signals are proposed. Expectation BackPropagation (EBP [48]), Binarized Neural Networks (BNNs [7]), Bitwise Neural Networks [25] and XNOR-Networks [42] have explored to brutally binarize input signals in addition to the binary weights. Targeting at lessening the quantization errors, high-order quantization methods, *e.g.,* High-Order Residual Quantization (HORQ [34]), multi-bit

quantization methods, *e.g.,* DoReFa-Net [57] and ternary quantization methods, *e.g.,* Gated XNOR Networks (GXNOR [9]).

The proposed TBN also falls in the type of networks that quantize both weights and inputs. Comparing to aforementioned work that aim to compensate the effectiveness of binarized networks with degraded efficiency, TBN for the first time provides an elegant integration between the binary weights and ternary inputs, so as to provide an optimal tradeoff between memory, efficiency and performance. We illustrate comparisons of these measurements between TBN and aforementioned method in Table 1.

There are other kinds of methods to compress and accelerate CNNs, *e.g.* pretrained based methods, distillation and so on. Fixed-point Factorized Networks (FFN [52]) decomposed the weight matrix of pretrained models into two ternary matrices and a non-negative diagonal matrix so that both the computational complexity and the storage requirement of networks are reduced. Ternary neural networks (TNNs [1]) used the teacher networks containing high-precision weights and ternary inputs, to teach the student networks which both weights and inputs are ternary-valued. LBCNN [24] used pre-defined binary convolutional filters to reduce the number of learnable parameters.

3 Ternary-Binary Networks

In this section, we introduce our proposed TBN in detail. Firstly, we present some notations and show how to implement the convolutional operation by matrix multiplication. Secondly, we explain how to obtain the binary weights and ternary inputs by approximating their full-precision counterparts. Given the binary weights and ternary inputs, we further illustrate the multiplication between them. Finally, the whole training procedure of our TBN is elaborated.

3.1 Convolution with Matrix Multiplication

A convolutional layer can be represented by a triplet $\langle \mathbf{I}, \mathcal{W}, * \rangle$. $\mathbf{I} \in \mathbb{R}^{c \times h_{in} \times w_{in}}$ is the input tensor, where (c, h_{in}, w_{in}) represents *channels*, *heights* and *widths*, respectively. $\mathbf{W} \in \mathbb{R}^{c \times h \times w} = \mathcal{W}_{i(i=1,\ldots,n)}$ is the i^{th} weight filter in $\mathcal{W}$, where n is the number of weight filters, and (h, w) represents the filter size. $*$ represents the convolution operation between $\mathbf{I}$ and $\mathbf{W}$ and the product is $\boldsymbol{C} \in \mathbb{R}^{n \times h_{out} \times w_{out}}$, where $h_{out} = (h_{in} + 2 \cdot p - w)/s + 1$ and $w_{out} = (w_{in} + 2 \cdot p - w)/s + 1$, and (p, s) represent the pad and stride parameter respectively. We refer to the inner product layer as the convolution layer, which is same to XNOR-Network [42].

As adopted in the popular Caffe package [22], we use matrix multiplication to implement the convolution layer $\langle \mathbf{I}, \mathcal{W}, * \rangle$. Specifically, by flattening each filter $\mathbf{W}$ to a row vector of shape $1 \times q$ $(q = c \times h \times w)$, the set of weight filters $\mathcal{W}$ can be reshaped to a matrix $\widetilde{\mathbf{W}} \in \mathbb{R}^{n \times q}$. We use function $ten2mat$ to represent this step, *i.e.* $\widetilde{\mathbf{W}} = ten2mat(\mathcal{W})$. Similarly, transforming each sub-tensor in the input tensor $\mathbf{I}$ with the same size as the filter to a column vector, we get the matrix $\widetilde{\mathbf{I}} \in \mathbb{R}^{q \times m} (m = h_{out} \times w_{out})$ after accumulating these vectors, *i.e.*

$\widetilde{\mathbf{I}} = ten2mat(\mathbf{I})$. Let we denote the product of the matrix multiplication with $\widetilde{\mathbf{I}}$ and $\widetilde{\mathbf{W}}$ as its operands as $\widetilde{\mathbf{C}} \in \mathbb{R}^{n \times m}$, *i.e.* $\widetilde{\mathbf{C}} = \widetilde{\mathbf{W}}\widetilde{\mathbf{I}}$. Finally, we reshape the matrix $\widetilde{\mathbf{C}}$ back to output tensor $\mathbf{C}$. This step is the inverse operation of $ten2mat$, denoted it by $mat2ten$. This is the entire process of implementing a convolutional layer using matrix multiplication, which can be summarized as:

$$\mathbf{C} = mat2ten(\widetilde{\mathbf{W}}\widetilde{\mathbf{I}}), \widetilde{\mathbf{W}} = ten2mat(\mathcal{W}), \widetilde{\mathbf{I}} = ten2mat(\mathbf{I}) \tag{1}$$

3.2 Binary Weights

Following XNOR-Networks [42], we adopt the similar paradigm to estimate the binary weights. Concretely, we use a binary filter $\mathbf{B} \in \{-1, +1\}^{c \times h \times w}$ and a scaling factor $\alpha \in \mathbb{R}^+$ to approximate a full-precision weight filter $\mathbf{W} \in \mathcal{W}$ such that $\mathbf{W} \approx \alpha\mathbf{B}$. The optimal approximation is obtained by solving the optimization problem of minimizing the ℓ_2 distance between full-precision and binary weight filters, *i.e.* $\alpha, \mathbf{B} = \arg\min_{\alpha, \mathbf{B}} \|\mathbf{W} - \alpha\mathbf{B}\|_2^2$. The optimal solution is

$$\mathbf{B} = sign(\mathbf{W}), \ \alpha = \frac{1}{c \times h \times w}\|\mathbf{W}\|_1. \tag{2}$$

The binary weight filters obtained by this simple strategy can reduce the storage of a convolutional layer by $\sim$32$\times$ compared to single-precision filters.

3.3 Ternary Inputs

In XNOR-Networks, the strategy to quantize the inputs of a convolutional layer to binary become quite complex. They taken the sign of input values to get binary input and calculated a matrix $\mathbf{A}$ by averaging the absolute values of elements in the input $\mathbf{I}$ across the channels, *i.e.*, $\mathbf{A} = \frac{\sum \|\mathbf{I}_{:,:,i}\|}{c}$. Then the scaling matrix $\mathbf{K}$ for the input was obtained by convolving the matrix $\mathbf{A}$ by a kernel $\mathbf{k} \in \mathbb{R}^{h \times w}$ with $\mathbf{k}_{ij} = \frac{1}{h \times w}$. However, the scaling factors of the binary inputs does not affect the performance of XNOR-Networks. Indicated by this, we abandon the scaling factors for the inputs to reduce unnecessary computation. On the other hand, TWN [33] with ternary weights has better performance than BWN [42] with binary weights. In order to improve the performance of binarized CNNs, we quantize each element of input tensor $\mathbf{I}$ into a ternary value $\{-1, 0, 1\}$ without the scaling factor.

We propose following threshold-based ternary function $f_{ternary}$ to obtain ternary input tensor $\mathbf{T} \in \{-1, 0, +1\}^{c \times h_{in} \times w_{in}}$:

$$\mathbf{T}_i = f_{ternary}(\mathbf{I}_i, \Delta) = \begin{cases} +1, \mathbf{I}_i > \ \Delta; \\ \ \ 0, |\mathbf{I}_i| \leq \Delta; \\ -1, \mathbf{I}_i < -\Delta; \end{cases} \tag{3}$$

where $\Delta \in \mathbb{R}^+$ is an positive threshold parameter. The value of Δ controls the numbers of -1, 0 and 1 in $\mathbf{T}$, which will highly affect the final accuracy. When

Δ is equal to 0, the function $f_{ternary}$ degenerates to the sign function. So that, a same performance as XNOR-Network will be obtained. However, when Δ is too big, each element in $\mathbf{T}$ will be zero according to Eq. (3), and we will get the worst result. Thus, an appropriate value of Δ is necessary. However, it is hard to obtain optimal Δ. Similar to TWN [33], we use following formula to calculate Δ:

$$\Delta = \delta \times \mathrm{E}(|\mathbf{I}|) \approx \frac{\delta}{c \times h_{in} \times w_{in}} \|\mathbf{I}\|_1 \tag{4}$$

where δ is a constant factor for all layers. We set $\delta = 0.4$ in the experiments. By this means, it is fast and easy to quantize a real-valued input tensor into a ternary architecture.

3.4 Ternary-Binary Dot Product

Once we obtain the binary weights and ternary inputs, how to achieve effective ternary-binary multiplication is our next target. As we know, the matrix multiplication is based on the dot product. That is to say, the entry $C_{ij} \in \widetilde{\mathbf{C}}$ is the result of dot product between the i^{th} row of weight matrix $\widetilde{\mathbf{W}}$ and the j^{th} column of input matrix $\widetilde{\mathbf{I}}$. *I.e.*, $C_{ij} = \widetilde{\boldsymbol{W}}_{\boldsymbol{i}} \cdot \widetilde{\boldsymbol{I}}_{\boldsymbol{j}}$ where $\cdot$ is the dot product, $\widetilde{\boldsymbol{W}}_{\boldsymbol{i}} = [\widetilde{W}_{i1}, \ldots, \widetilde{W}_{iq}]$ and $\widetilde{\boldsymbol{I}}_{\boldsymbol{j}} = [\widetilde{I}_{1j}, \ldots, \widetilde{I}_{qj}]$. We can use binary operations to accelerate the dot product with a binary vector and a ternary vector as its operands. Let us use $\alpha \boldsymbol{b}$ to denote the binary filter $\mathbf{B}$ corresponding to $\widetilde{\boldsymbol{W}}_{\boldsymbol{i}}$, where $\widetilde{\boldsymbol{W}}_{\boldsymbol{i}} \approx \alpha \boldsymbol{b}, \boldsymbol{b} = ten2mat(\mathbf{B}) \in \{-1, 1\}^q$ and α is the scaling factor. Similarly, the ternary vector $\boldsymbol{t} \in \{-1, 0, +1\}^q$ corresponds to $\widetilde{\boldsymbol{I}}_{\boldsymbol{j}}$. So we can implement this special dot product efficiently with the following formula:

$$C_{ij} = \alpha(c_t - 2 \times \mathbf{bitcount}((\boldsymbol{b}\ \mathbf{XOR}\ \boldsymbol{t}')\ \mathbf{AND}\ \boldsymbol{t}'')), \tag{5}$$

where we decompose vector $\boldsymbol{t}$ into two vector $\boldsymbol{t}' \in \{-1, 1\}^q$ and $\boldsymbol{t}'' \in \{0, 1\}^q$ as follows:

$$\boldsymbol{t}'_i = \begin{cases} 1, & \boldsymbol{t}_i = 1 \\ -1, & \text{otherwise} \end{cases}, \quad \boldsymbol{t}''_i = \begin{cases} 0, & \boldsymbol{t}_i = 0 \\ 1, & \text{otherwise} \end{cases}, \quad i = 1, \ldots, q \tag{6}$$

so that $t_i = t'_i \times t''_i$. $c_t = \mathbf{bitcount}(\boldsymbol{t}'') = \|\boldsymbol{t}\|_1$ is a constant which is independent of $\boldsymbol{b}$. In Eq. (5), the operation **bitcount** return the count of number of bits set to 1 and **XOR**, **AND** are the logic operations. It should be noted that 1 in $\boldsymbol{b}, \boldsymbol{t}', \boldsymbol{t}''$ is considered to be logic true, and the others (*i.e.* $0, -1$) are regard as logic false. So we can implement the efficient matrix multiplication.

3.5 Training TBN

With above strategies, we can get an very fast convolutional layer with ternary inputs and binary weights (TBConvolution), and Algorithm 1 demonstrates how TBConvolution works. In the TBN, there is a batch normalization layer [20] to normalize the inputs before each TBConvolution, so that the number of -1, 0, 1 for ternary inputs is more balanced. A non-linear activation layer (*e.g.* ReLU)

Algorithm 1. TBConvolution($\mathcal{W}, \mathbf{I}$)

Input: A set of weight filters $\mathcal{W} \in \mathbb{R}^{n \times c \times h \times w}$, the input tensor $\mathbf{I} \in \mathbb{R}^{c \times h_{in} \times w_{in}}$ and convolutional parameters including the stride s and pad p
Output: The convolutional result $\mathbf{C} \in \mathbb{R}^{n \times h_{out} \times w_{out}}$
for i^{th} filter $\mathbf{W}$ in $\mathcal{W}$ **do**
  $\alpha = \frac{1}{c \times h \times w} \|\mathbf{W}\|_1$ // `calculate the scaling factor w.r.t. Eq. (2)`
  $\mathbf{B} = sign(\mathbf{W})$ // `get the binary filter w.r.t. Eq. (2)`
  $\mathbf{W} \approx \alpha \mathbf{B}$
end for
$\Delta = \frac{\delta}{c \times h_{in} \times w_{in}} \|\mathbf{I}\|_1$ // `calculate the threshold parameter w.r.t. Eq. (4)`
$\mathbf{T} = f_{ternary}(\mathbf{I}, \Delta)$ // `get the ternary input w.r.t. Eq. (3)`
$\widetilde{\mathbf{W}} = ten2mat(\mathcal{W})$ and $\widetilde{\mathbf{I}} = ten2mat(\mathbf{T})$ // `covert weights and input tensors to matrices w.r.t. Eq. (1)`
$\widetilde{\mathbf{C}} = \widetilde{\mathbf{W}}\widetilde{\mathbf{I}}$ // `accelerate matrix multiplication w.r.t. Eq. (5)`
$\mathbf{C} = mat2ten(\widetilde{\mathbf{C}})$ // `convert the product to the tensor w.r.t. Eq. (1)`

Algorithm 2. Training an L-layer TBN

Input: A minibatch of inputs and targets $(\mathbf{X}_0, \mathbf{Y})$, cost function $C(\mathbf{Y}, \hat{\mathbf{Y}})$, current weights $\widehat{\mathcal{W}}(t) = \{\mathcal{W}_l(t)\}_{l=1}^{L}$, and current learning rate $\eta(t)$
Output: updated weight $\widehat{\mathcal{W}}(t+1)$ and updated learning rate $\eta(t+1)$
for $l = 1$ to L **do**
  $\mathbf{I}_l = \text{BatchNormalization}(\mathbf{X}_{l-1})$
  $\mathbf{X}_l = \text{TBConvolution}(\mathcal{W}_l(t), \mathbf{I}_l)$
end for
$\frac{\partial C}{\partial \mathcal{W}} = \text{Backward}(\frac{\partial C}{\partial \mathbf{X}_L}, \widehat{\mathcal{W}}(t))$ // `standard backward propagation with Eq. (7)`
$\widehat{\mathcal{W}}(t+1) = \text{UpdateParameters}(\widehat{\mathcal{W}}(t), \frac{\partial C}{\partial \mathcal{W}}, \eta(t))$ // `Any optimizer (e.g. ADAM)`
$\eta(t+1) = \text{UpdateLearningRate}(\eta(t), t)$ // `Any learning rate scheduling function`
Note: $\mathcal{W}_l$ here is identical to the l^{th} layer TBN weights $\mathcal{W}$ (mentioned in Section 3.1)

after each TBConvolution is optional because ternary quantization can play the role of the non-linear activation function. Other layers (*e.g.* pooling and dropout) can be inserted after TBConvolution (or before the batch normalization layer). To train TBN, the full-precision gradient is adopted and we use straight-through estimator [4] to compute the gradient of binary and ternary quantization function, *i.e.*

$$\frac{\partial sign}{\partial r} = \frac{\partial f_{ternary}}{\partial r} = \mathbf{1}_{|r|<1} = \begin{cases} 1, |r| < 1 \\ 0, \text{otherwise} \end{cases} \tag{7}$$

Similar to the strategy used in BNN [7], XNOR-Networks [42] and HORQ [34], we do not apply our approach on the first or last layer. Algorithm 2 demonstrates the procedure for training an L-layer Ternary-Binary Network. We can use any optimizer (*e.g.* ADAM [27]) to train TBNs.

4 Experiments

As the proposed TBN uses ternary inputs with binary weights to simultaneously reduce the approximation error caused by quantization but maintain the reasonable performance to some extend, the goal of our experiments is mainly to answer the following three research questions:

- **Q1:** How does TBN perform compared to other quantized/squeezed deep networks (*i.e.*, XNOR-Networks, HORQ) in different tasks (*i.e.*, image recognition and object detection)?
- **Q2:** How fast can TBN accelerate compared to other quantized networks?
- **Q3:** How is the performance of TBN influenced by different components (*e.g.* the sparsity of ternary inputs, the usage of activation functions)?

4.1 Image Classification

Datasets and Configurations: We evaluate the performance of our proposed approach on four different datasets, *i.e.* MNIST [32], CIFAR-10 [28], SVHN [40], ImageNet (ILSVRC2012) [44], and compare it with other methods. As previously mentioned, our TBN can accommodate any network architectures. Hence,

Table 2. The classification accuracies of different CNNs trained with various models on the four datasets. Both "top-1/top-5" accuracies are presented for the ImageNet dataset. "-" indicates that the results are not provided in their original papers.

Dataset models		MNIST LeNet-5	CIFAR-10 VGG-7	SVHN VGG-7	ImageNet AlexNet	ImageNet ResNet-18	ImageNet ResNet-34
	Full-precision	99.48	92.88	97.68	57.2/80.2	69.3/89.2	73.3/91.4
Quantize weights	BC [6]	98.82	91.73	97.85	35.5/61.0	-	-
	BWN [42]	99.38	92.58	97.46	56.8/79.4	60.8/83.0	-
	TWN [33]	99.38	92.56	-	54.5/76.8	65.3/86.2	-
	TTQ [58]	-	-	-	57.5/79.7	66.6/87.2	-
Other methods	FFN [52]	-	-	-	55.5/79.0	-	-
	LCNN-fast [3]	-	-	-	44.3/68.7	51.8/76.8	-
	LCNN-accurate [3]	-	-	-	55.1/78.1	62.2/84.6	-
	LBCNN [24]	99.51	92.66	94.50	54.9/-	-	-
Quantize inputs and weights	TNN [1]	98.33	87.89	97.27	-	-	-
	GXNOR [9]	99.32	92.50	97.37	-	-	-
	BNN [7]	98.60	89.85	97.47	27.9/50.42	-	-
	DoReFa-Net* [57]	-	-	97.6	47.7/-	-	-
	BinaryNet [50]	-	-	-	46.6/71.1	-	-
	HORQ [34]	99.38	91.18	97.41	-	55.9/78.9	-
	XNOR-Network [42]	99.21	90.02	96.96	44.2/69.2	51.2/73.2	55.9/79.1
	TBN	99.38	90.85	97.27	49.7/74.2	55.6/79.0	58.2/81.0

*We adopt DoReFa-Net with 1-bit weight, 2-bit activation and 32-bit gradient for fair comparison.

we performance the following evaluations with different networks on the above datasets. Note that we adopt the Adam optimizer with Batch Normalization to speed up the training, and ReLU is adopted as the activation function in all the following experiments. In addition, all the deep networks used in this section are training from the scratch.

Results on MNIST with LeNet-5: The LeNet-5 [32] architecture we used is "32-C5 + MP2 + 64-C5 + MP2 + 512-FC + 10-FC + SVM". It is composed of two convolutional layers with size 5×5, a fully connected layer and a SVM classifier with 10 labels. Specifically, there is no pre-processing, data-augmentation or pre-training skills to remain the challenge. The learning rate starts at 0.001 and is divided by 10 at epoch 15, 30, 45 with the mini-batch size 200. We report the best accuracy on the testing set. From the results shown in Table 2, we observe that our TBN has the same performance as HORQ but outperforms XNOR-Network by 0.17%. In fact, on the MNIST dataset, there are subtle difference between those methods (less than 1%).

Results on CIFAR-10 with VGG-7: To train the networks on CIFAR-10 dataset, we follow the same data augmentation scheme in ResNet [18]. In detail, we use the VGG inspired architecture, denoted as VGG-7, by: "$2\times$(128-C3) + MP2 + $2\times$(256-C3) + MP2 + $2\times$(512-C3) + MP2 + 1024-FC + 10-FC + Softmax", where C3 is a 3×3 convolutional block, MP2 is a max-pooling layer with kernel size 2 and stride 2, and Softmax is a softmax loss layer. We train this model for 200 epochs with a mini-batch of 200. The learning rate also starts at 0.001 and is scaled by 0.5 every 50 epochs. The results are given in Table 2. The accuracy of our TBN on CIFAR-10 is higher than XNOR-Network's and BNNs. However, compared with HORQ, and GXNOR, the performance of TBN is slightly worse since more quantization for both inputs and weights are adopted in our methods.

Results on SVHN with VGG-7: We also use VGG-7 networks for SVHN. Because SVHN is a much larger dataset than CIFAR-10, we only train VGG-7 for 60 epochs. From, results presented in Table 2, it is easily discovered that the performances between TBN, HORQ, XNOR-Networks, BNN, GXNOR and TNN is almost at the same level (Fig. 2).

Results on ImageNet with AlexNet: In this experiment, we report our classification performance in terms of top-1 and top-5 accuracies using AlexNet. Specifically, AlexNet is with 5 convolutional layers and two fully-connected layers. We train the network for 100 epochs. The learning rate starts at 0.001 and is divided by 0.1 every 25 epochs. Figures 2(a) and (b) demonstrate the classification accuracy for training and inference along with the training epochs. The solid lines represent training and validation accuracy of TBN, and dashed lines show the accuracy of XNOR-Network. The final accuracy of AlexNet is showed

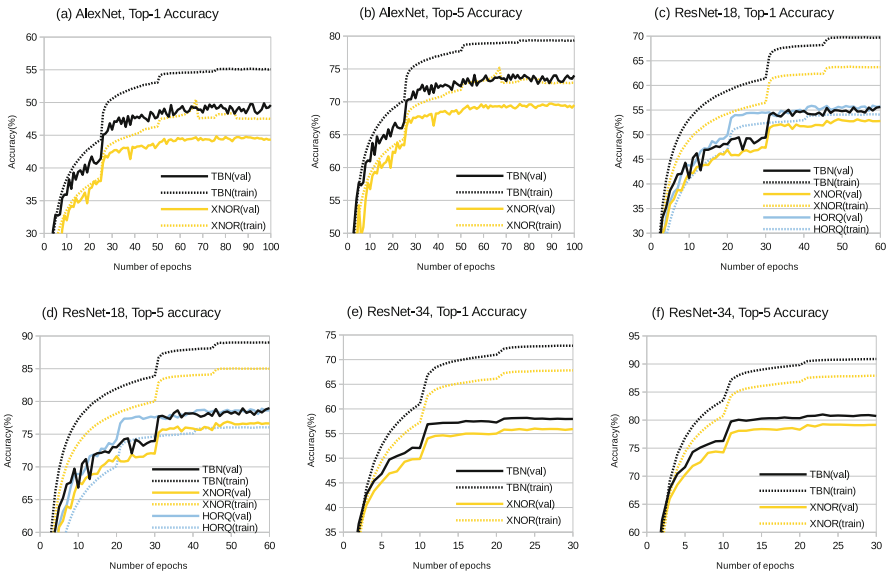

Fig. 2. (a) and (b) Compare the top-1 and top-5 accuracies between TBN and XNOR-Networks on AlexNet; (c) and (d) compare the top-1 and top-5 accuracies between TBN, HORQ and XNOR-Networks on ResNet-18; (e) and (f) compare the top-1 and top-5 accuracies between TBN and XNOR-Networks on ResNet-34.

in Table 2, which illustrates our TBN outperforms XNOR-Network by the large margin (5.5% on top-1 accuracy and 5.0% on top-5 accuracy) (Table 3).

Results on ImageNet with ResNet: In addition to the AlexNet, we also train two Ternary-Binary Networks for both ResNet-18 and ResNet-34 [18] architectures on the ImageNet dataset. We run the training algorithm for 60 epochs with a mini-batch size of 128. The learning rate starts at 0.001 and is scaled by 0.1 every 20 epochs. ResNet-34 adopts the same training strategy, but is only trained with 30 epochs in total and the learning rate is decayed every 10 epochs. Figures 2(c)–(f) demonstrate the classification accuracies (top-1 and top-5) of ResNet-18 and ResNet-34 respectively, along with the epochs for training and inference. The final results are reported in Table 2, which show that Ternary-Binary Network is better than XNOR-Networks (ResNet-18: by 4.4%/4.8% on top-1/top-5, ResNet-34: by 2.3%/1.9% on top-1/top-5). Meanwhile, the performance of our TBN is competitive to that of HORQ (top1: 55.6% vs. 55.9%; top-5 79.0% vs. 78.9% on ResNet-18).

4.2 Object Detection

We also evaluate the performance of TBN on the object detection task. Various modified network architectures are used, including Faster-RCNN [43], and

Table 3. The performance (in mAP) comparison of TBN, XNOR-Networks and full-precision CNN models for object detection. All methods are trained on the combination of VOC2007 and VOC2012 `trainval` sets and tested on the VOC2007 `test` set.

Method base network	Full-precision VGG-16	Full-precision ResNet-34	XNOR-Networks ResNet-34	TBN ResNet-34
Faster R-CNN	73.2	75.6	54.7	59.0
SSD 300	74.3	75.5	55.1	59.5

Single Shot Detector (SSD [38]). We change the base network of these architectures to the TBN with ResNet-34. We compare the performance with XNOR-Networks and full-precision networks. We evaluate all methods on the PASCAL VOC dataset [11,12], which is a standard recognition benchmark with detection and semantic segmentation challenges. We train our models on the combination of VOC2007 `trainval` and VOC2012 `trainval` (16,551 images) and test on VOC2007 `test` (4,952 images).

The comparison results for object detection are illustrated in Table 4(b). As can be seen from the table, the models based on ResNet-34 can achieve better performance both on Faster R-CNN and SSD. Our TBN with ResNet achieves up to 4.4% higher than XNOR-Networks in terms of mAP, although there is a large margin compared to full-precision networks.

4.3 Efficiency

In this section, we will illustrate the efficiency comparison between different methods. Suppose matrix multiplication $\widetilde{\mathbf{C}} = \widetilde{\mathbf{W}}\widetilde{\mathbf{I}}$, where $\widetilde{\mathbf{W}} \in \mathbb{R}^{n\times q}$, $\widetilde{\mathbf{I}} \in \mathbb{R}^{q\times m}$ and $\widetilde{\mathbf{C}} \in \mathbb{R}^{n\times m}$. To calculate $\widetilde{\mathbf{C}}$, there are $n \times m \times q$ multiply-accumulate operations (MACs) required. If the matrix $\widetilde{\mathbf{I}}$ is quantized to ternary values and the matrix $\widetilde{\mathbf{W}}$ is binary-valued, matrix multiplication requires $n \times m$ multiplications and $n \times m \times q$ **AND**, **XOR** and **bitcount** operations respectively, according to Eq. (5). We provide a comparison between our approach and the related works using quantized inputs and weights in Table 1. Compared with XNOR-Networks, our approach increases $n \times m \times q$ binary operations and saves $n \times m$ MACs, while HORQ needs twice the number of MACs and binary operations as XNOR networks.

The general computation platform (*i.e.*, CPU, GPU, ARM) can perform an L-bits binary operation in one clock cycle ($L = 64$ typical[1]). Assume the ratio between the speed of performing an L-bits binary operation and a multiply-accumulate operation is γ, *i.e.*

$$\gamma = \frac{\text{average time required by MAC}}{\text{average time required by } L\text{-bits binary operation}} \tag{8}$$

[1] An Intel SSE(, AVX, AVX-512) instruction can perform 128(, 256, 512) bits binary operation.

Therefore, the speedup ratio of Ternary-Binary Networks is:

$$S = \frac{\gamma nmq}{\gamma nm + 3nm\lceil \frac{q}{L} \rceil} = \frac{\gamma q}{\gamma + 3\lceil \frac{q}{L} \rceil} \tag{9}$$

It shows that speedup ratio depends on q, L, while γ is determined by machine. For a convolutional layer, $q = c \times h \times w$, that is to say, S is independent of the input size. According to the speedup ratio achieved by XNOR-Network, we can safely assume $\gamma = 1.91$. To maximize the speedup, q should be several times of L. In Fig. 3, we illustrate the relationship between the speedup ratio and q, L. It shows that we can obtain higher speedup ratio by increasing q or L.

Table 1 compares the speedup ratio achieved by different methods, in which parameters are fixed as: $\gamma = 1.91, L = 64$ and $q = c \times h \times w = 2304$[2]. When real-valued inputs with either binary or ternary weight, the MAC operation can be replaced by only addition and subtraction, and achieving $\sim 2\times$ speedup [42]. While, the methods which both weights and inputs are quantized achieve high speedup ($\geq 15\times$) by using binary operation. Specifically, more bits used by weights or inputs, the lower speedup ratio is but getting the lower the approximation error. Using our approach, we gain $40\times$ theoretical speedup ratio, which is $11\times$ higher than HORQ. See **Supplementary Material** for more details.

Remark. Why Not Use Ternary Weights with Binary Inputs: Actually, a ternary (2-bit) weights network with binary inputs uses the same scheme as TBN to accelerate CNN, but requires twice as much storage space as TBN. Since TBN has higher compression rate, we choose the TBN from these two equivalent approaches.

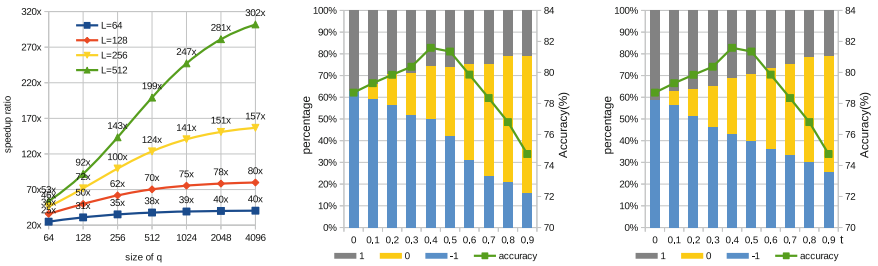

Fig. 3. (a) The relationship between speedup ratio and q under different L; (b) the classification accuracy with varying δ in Eq. (4), *i.e.* sparsity of ternary inputs. The percentage stacked histogram shows the percentage of the average number of -1, 0, 1 w.r.t. the inputs of the second convolutional layer; (c) the classification accuracy and percentage stacked histogram w.r.t. the inputs of the third convolutional layer.

[2] For the majority of convolutional layer in ResNet [18] architecture, it's kernel size is 3×3 and input channel size is 256, so we fix $q = 256 \times 3^2 = 2304$.

4.4 Analysis of TBN Components

Sparsity of Ternary Inputs: To explore the relationship between sparsity and accuracy, we vary δ in Eq. (4) and train a **Simple Network** structure on CIFAR-10: "32-C5 + MP3 + 32-C5 + MP3 + 64-C5 + AP3 + 10FC + Softmax". We adopt this kind of structure because of its simplicity and flexibility for the performance comparison. The learning strategy is the same as VGG-7 in Sect. 4.1. The classification accuracies with different degrees of sparsities are shown in Fig. 3(b) and (c). As can been seen from the two figures, when δ grows from 0 (which is the case of XNOR-Networks) to 0.4, both the number of zeros and accuracy increase accordingly. However, when δ further increases, the model capacity is reduced and the error rate is increased quickly. Therefore, we set $\delta = 0.4$ in our experiments.

Table 4. (a) The classification performance of TBN while using non-linear layers with different activation functions (on the CIFAR-10 dataset). "None" denotes that we don't use non-linear layer; (b) the comparison of accuracies (%) on CIFAR-10 after quantizing the inputs of first/last convolutional layers. ✓ indicates that we quantize the first/last layers, and ✗ indicates that we use full-precision inputs and weights.

(a)

Base Networks	Non-Linear Layers			
	None	ReLU	Sigmoid	PReLU
ResNet-20	81.36	82.15	79.12	84.34
VGG-7	89.49	90.85	89.78	90.10
Simple Network	75.92	81.21	78.67	81.14

(b)

First	Last	XNOR	TBN	BWN	HORQ
✗	✗	79.11	81.21	82.66	81.12
✗	✓	71.64	76.88	81.86	76.64
✓	✗	62.85	65.57	76.61	68.69
✓	✓	52.41	58.86	73.66	62.55
Full-Precision		85.51			

Effect of Activation Function: Here, we explore the influence of different activation functions on our TBN framework. Specifically, we incorporate three non-linear activation functions, *i.e.* ReLU, Sigmoind and PReLU. 'Simple Network' in the table indicates the simple base network architecture used in the above paragraph. As shown in Table 4(a), the accuracy can be improved when using the non-linear activation functions, and using PReLU could achieve the best performance. However, the improvement is subtle, mainly because the ternary quantization function in our TBN already plays the role of activation function.

Quantizing the First/Last Layer? As shown in our framework, we avoid the quantization step on the first and last layers of the networks. The reasons are two-fold: Firstly, the inputs of the first layer have much fewer channels (*i.e.* $c = 3$), thus the speedup ratio in efficiency is not considerably high. Secondly, if the inputs of the first or last layer are quantized, the performance will drop significantly, which can be seen from Table 4(b). Note that all the results here are obtained based on the previously mentioned Simple Network. As we can see

from the table, the accuracies of the four networks decrease consistently by a large margin after quantizing their first/last layers, and the performance drop is especially obvious when the first layer is quantized.

5 Conclusion

In this paper, we for the first time incorporated binary network weights and ternary layer-wise inputs as a lightweighted approximation to standard CNNs. We claim the ternary inputs along with the binary weights can provide an optimal tradeoff between memory, efficiency and performance. An accelerated ternary-binary matrix multiplication that employs highly efficient **XOR**, **AND** and **bitcount** operations was introduced in TBN, which achieved ∼32× memory saving and 40× speedup over its full-precision CNN counterparts. TBN demonstrated its consistent effectiveness when applied to various CNN architectures on multiple datasets of different scales, and it also outperformed the XNOR-Network by up to 5.5% (top-1 accuracy) on the ImageNet classification task, and up to 4.4% (mAP score) on the PASCAL VOC object detection task.

Acknowledgments. This work was supported in part by the National Natural Science Foundation of China under Project 61502081 and Project 61632007, the Fundamental Research Funds for the Central Universities under Project ZYGX2014Z007.

References

1. Alemdar, H., Leroy, V., Prost-Boucle, A., Pétrot, F.: Ternary neural networks for resource-efficient AI applications. In: 2017 International Joint Conference on Neural Networks (IJCNN), pp. 2547–2554. IEEE (2017)
2. Ambai, M., Matsumoto, T., Yamashita, T., Fujiyoshi, H.: Ternary weight decomposition and binary activation encoding for fast and compact neural network. In: Proceedings of International Conference on Learning Representations (2017)
3. Bagherinezhad, H., Rastegari, M., Farhadi, A.: LCNN: lookup-based convolutional neural network. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (2017)
4. Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432 (2013)
5. Courbariaux, M., Bengio, Y., David, J.: Low precision arithmetic for deep learning. CoRR, abs/1412.7024 **4** (2014)
6. Courbariaux, M., Bengio, Y., David, J.P.: Binaryconnect: training deep neural networks with binary weights during propagations. In: Proceedings of Advances in Neural Information Processing Systems, pp. 3123–3131 (2015)
7. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: training deep neural networks with weights and activations constrained to +1 or −1. In: Proceedings of Advances in Neural Information Processing Systems, pp. 4107–4115 (2016)
8. Dai, J., Li, Y., He, K., Sun, J.: R-FCN: Object detection via region-based fully convolutional networks. In: Proceedings of Advances in Neural Information Processing Systems, pp. 379–387 (2016)

9. Deng, L., Jiao, P., Pei, J., Wu, Z., Li, G.: Gated XNOR networks: deep neural networks with ternary weights and activations under a unified discretization framework. arXiv preprint arXiv:1705.09283 (2017)
10. Denil, M., Shakibi, B., Dinh, L., De Freitas, N., et al.: Predicting parameters in deep learning. In: Proceedings of Advances in Neural Information Processing Systems, pp. 2148–2156 (2013)
11. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes Challenge 2007 (VOC 2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html
12. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes Challenge 2012 (VOC 2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html
13. Girshick, R.: Fast R-CNN. In: Proceedings of IEEE Conference on Computer Vision, pp. 1440–1448 (2015)
14. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587 (2014)
15. Gupta, S., Agrawal, A., Gopalakrishnan, K., Narayanan, P.: Deep learning with limited numerical precision. In: Proceedings of International Conference on Machine Learning, pp. 1737–1746 (2015)
16. Han, S., Mao, H., Dally, W.J.: Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding. In: Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman coding (2016)
17. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Proceedings of Advances in Neural Information Processing Systems, pp. 1135–1143 (2015)
18. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
19. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. In: NIPS 2014 Deep Learning and Representation Learning Workshop (2014)
20. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of International Conference on Machine Learning, pp. 448–456 (2015)
21. Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up convolutional neural networks with low rank expansions. In: BMVC (2014)
22. Jia, Y., et al.: Caffe: convolutional architecture for fast feature embedding. arXiv preprint arXiv:1408.5093 (2014)
23. Jin, J., Dundar, A., Culurciello, E.: Flattened convolutional neural networks for feedforward acceleration. arXiv preprint arXiv:1412.5474 (2014)
24. Juefei-Xu, F., Boddeti, V.N., Savvides, M.: Local binary convolutional neural networks. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (2017)
25. Kim, M., Smaragdis, P.: Bitwise neural networks. CoRR abs/1601.06071 (2015)
26. Kim, Y.D., Park, E., Yoo, S., Choi, T., Yang, L., Shin, D.: Compression of deep convolutional neural networks for fast and low power mobile applications. arXiv preprint arXiv:1511.06530 (2015)
27. Kingma, D., Ba, J.: Adam: a method for stochastic optimization. In: Proceedings of International Conference on Learning Representations (2015)

28. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images (2009)
29. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Proceedings of Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
30. Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., Lempitsky, V.: Speeding-up convolutional neural networks using fine-tuned CP-decomposition. In: Proceedings of International Conference on Learning Representations (2015)
31. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
32. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
33. Li, F., Zhang, B., Liu, B.: Ternary weight networks. In: The 1st International Workshop on Efficient Methods for Deep Neural Networks (2016)
34. Li, Z., Ni, B., Zhang, W., Yang, X., Gao, W.: Performance guaranteed network acceleration via high-order residual quantization. In: Proceedings of IEEE Conference on Computer Vision (2017)
35. Lin, D., Talathi, S., Annapureddy, S.: Fixed point quantization of deep convolutional networks. In: Proceedings of International Conference on Machine Learning, pp. 2849–2858 (2016)
36. Lin, Z., Courbariaux, M., Memisevic, R., Bengio, Y.: Neural networks with few multiplications. In: Proceedings of International Conference on Learning Representations (2016)
37. Liu, L., Shao, L., Shen, F., Yu, M.: Discretely coding semantic rank orders for image hashing. In: Proceeding of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2862–2871 (2017)
38. Liu, W., et al.: SSD: single shot multibox detector. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9905, pp. 21–37. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46448-0_2
39. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 3431–3440 (2015)
40. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning. In: NIPS Workshop on Deep Learning and Unsupervised Feature Learning, vol. 2011, p. 5 (2011)
41. Pinheiro, P.O., Collobert, R., Dollár, P.: Learning to segment object candidates. In: Proceedings of Advances in Neural Information Processing Systems, pp. 1990–1998 (2015)
42. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: XNOR-Net: ImageNet classification using binary convolutional neural networks. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9908, pp. 525–542. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46493-0_32
43. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. In: Proceedings of Advances in Neural Information Processing Systems, pp. 91–99 (2015)
44. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M.: Imagenet large scale visual recognition challenge. Int. J. Comput. Vis. **115**(3), 211–252 (2015)
45. Shen, F., Gao, X., Liu, L., Yang, Y., Shen, H.T.: Deep asymmetric pairwise hashing. In: Proceedings of the 2017 ACM on Multimedia Conference, pp. 1522–1530. ACM (2017)

46. Shen, F., Xu, Y., Liu, L., Yang, Y., Huang, Z., Shen, H.T.: Unsupervised deep hashing with similarity-adaptive and discrete optimization. IEEE Trans. Pattern Anal. Mach. Intell. (2018)
47. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: Proceedings of International Conference on Learning Representations (2015)
48. Soudry, D., Hubara, I., Meir, R.: Expectation backpropagation: parameter-free training of multilayer neural networks with continuous or discrete weights. In: Proceedings of Advances in Neural Information Processing Systems, pp. 963–971 (2014)
49. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proc. IEEE Conf. Comp. Vis. Patt. Recogn. pp. 1–9 (2015)
50. Tang, W., Hua, G., Wang, L.: How to train a compact binary neural network with high accuracy? In: Proceedings of AAAI Conference on Artificial Intelligence, pp. 2625–2631 (2017)
51. Wang, P., Cheng, J.: Accelerating convolutional neural networks for mobile applications. In: Proceedings of the 2016 ACM on Multimedia Conference, pp. 541–545. ACM (2016)
52. Wang, P., Cheng, J.: Fixed-point factorized networks. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (2017)
53. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: Proceedings of Advances in Neural Information Processing Systems, pp. 2074–2082 (2016)
54. Wu, J., Leng, C., Wang, Y., Hu, Q., Cheng, J.: Quantized convolutional neural networks for mobile devices. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 4820–4828 (2016)
55. Zhang, X., Zou, J., He, K., Sun, J.: Accelerating very deep convolutional networks for classification and detection. IEEE Trans. Pattern Anal. Mach. Intell. **38**(10), 1943–1955 (2016)
56. Zhou, A., Yao, A., Guo, Y., Xu, L., Chen, Y.: Incremental network quantization: towards lossless CNNs with low-precision weights. In: Proceedings of International Conference on Learning Representations (2017)
57. Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y.: Dorefa-Net: training low bitwidth convolutional neural networks with low bitwidth gradients. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (2016)
58. Zhu, C., Han, S., Mao, H., Dally, W.J.: Trained ternary quantization. In: Proceedings of International Conference on Learning Representations (2017)