

# Recurrent Bilinear Optimization for Binary Neural Networks

Sheng Xu<sup>1†</sup>, Yanjing Li<sup>1†</sup>, Tiancheng Wang<sup>1</sup>, Teli Ma<sup>2</sup>, Baochang Zhang<sup>1,3\*</sup>,  
Peng Gao<sup>2</sup>, Yu Qiao<sup>2</sup>, Jinhu Lü<sup>1,3</sup>, Guodong Guo<sup>4,5</sup>

<sup>1</sup> Beihang University, Beijing, China

<sup>2</sup> Shanghai Artificial Intelligence Laboratory, Shanghai, China

<sup>3</sup> Zhongguancun Laboratory, Beijing, China

<sup>4</sup> Institute of Deep Learning, Baidu Research, Beijing, China

<sup>5</sup> National Engineering Laboratory for Deep Learning Technology and Application,  
Beijing, China

{shengxu, yanjingli, bczhang}@buaa.edu.cn

**Abstract.** Binary Neural Networks (BNNs) show great promise for real-world embedded devices. As one of the critical steps to achieve a powerful BNN, the scale factor calculation plays an essential role in reducing the performance gap to their real-valued counterparts. However, existing BNNs neglect the intrinsic bilinear relationship of real-valued weights and scale factors, resulting in a sub-optimal model caused by an insufficient training process. To address this issue, *Recurrent Bilinear Optimization* is proposed to improve the learning process of *BNNs* (RBONNs) by associating the intrinsic bilinear variables in the back propagation process. Our work is the first attempt to optimize BNNs from the bilinear perspective. Specifically, we employ a recurrent optimization and Density-ReLU to sequentially backtrack the sparse real-valued weight filters, which will be sufficiently trained and reach their performance limits based on a controllable learning process. We obtain robust RBONNs, which show impressive performance over state-of-the-art BNNs on various models and datasets. Particularly, on the task of object detection, RBONNs have great generalization performance. Our code is open-sourced on <https://github.com/SteveTsui/RBONN>.

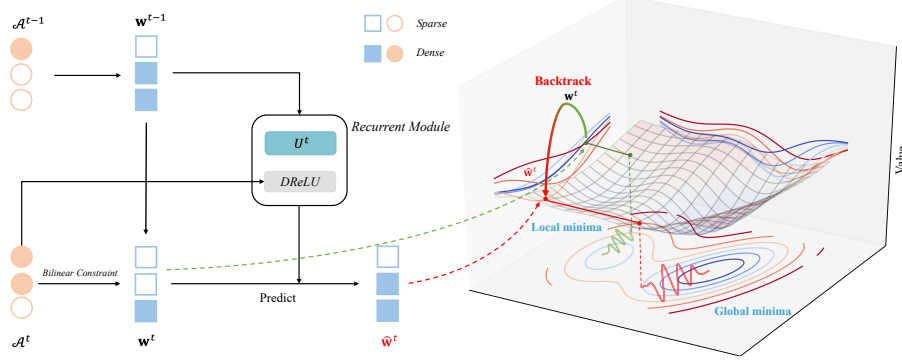
**Keywords:** Binary neural network, Bilinear optimization, Image classification, Object detection

## 1 Introduction

Computer vision has been rapidly promoted, with the widespread application of convolutional neural networks (CNNs) in image classification [37,8], semantic segmentation [9], and object detection [6,22]. It does, however, come with a huge demand for memory and computing resources. These computation and

<sup>†</sup> Equal contribution.

\* Corresponding author.



**Fig. 1.** An illustration of the RBONN framework. Conventional gradient-based algorithms assume that the hidden variables in bilinear models are independent, which causes an insufficient training of  $\mathbf{w}$  due to neglecting the relationship with  $\mathcal{A}$  as shown in the loss surface (right part). Our RBONN can help  $\mathbf{w}$  escape from local minima (green dotted line) and achieve a better solution (red dotted line).

memory costs are incompatible with the computing capabilities of devices, particularly those with low resources, *e.g.*, mobile phones and embedded devices. As a result, substantial research has been invested to reduce storage and computation cost. To accomplish this, a number of compression methods for efficient inference have been proposed, including network pruning [15,13,16], low-rank decomposition [5,20], network quantization [34,29,18], and knowledge distillation [36]. Network quantization, for example, is particularly well suited for using on embedded devices since it decreases the bit-width of network weights and activations. Binarization, a particularly aggressive kind of quantization, reduces CNN parameters and activations into 1 bit, reducing memory usage by  $32\times$  and calculation costs by  $58\times$  [34]. Binarized neural networks (BNNs) are employed for a wide range of applications, such as image classification [34,29,28], object detection [41,43] and point cloud recognition [42]. With high energy-efficiency, they are potent to be directly applied on AI chips. However, due to the limited representation capabilities, BNNs' performance remains considerably inferior to that of their real-valued counterparts.

Previous methods [10,24] compute scale factors by approximating the real-valued weight filter  $\mathbf{w}$  such that  $\mathbf{w} \approx \alpha \circ \mathbf{b}^{\mathbf{w}}$ , where  $\alpha \in \mathbb{R}_+$  is the scale factor (vector) and  $\mathbf{b}^{\mathbf{w}} = \text{sign}(\mathbf{w})$  to enhance the representation capability of BNNs. In essence, the approximation can be considered as a bilinear optimization problem with the objective function as

$$\arg \min_{\mathbf{w}, \alpha} G(\mathbf{w}, \alpha) = \|\mathbf{w} - \alpha \circ \mathbf{b}^{\mathbf{w}}\|_2^2 + R(\mathbf{w}),$$

or

$$\arg \min_{\mathbf{w}, \mathcal{A}} G(\mathbf{w}, \mathcal{A}) = \|\mathbf{b}^{\mathbf{w}} - \mathcal{A}\mathbf{w}\|_2^2 + R(\mathbf{w}), \quad (1)$$

where  $\mathcal{A} = \text{diag}(\frac{1}{\alpha_1}, \dots, \frac{1}{\alpha_N})$ ,  $N$  is the number of elements in  $\alpha$ .  $\circ$  denotes the channel-wise multiplication, and  $R(\cdot)$  represents the regularization, typically the  $\ell_1$  or  $\ell_2$  norm.  $G(\mathbf{w}, \mathcal{A})$  includes a bilinear form of  $\mathcal{A}\mathbf{w}$  widely used in the field of computer vision [4,30,14]. Note that the bilinear function is  $\mathcal{A}\mathbf{w}$  rather than  $G(\mathbf{w}, \mathcal{A})$  in Eq. 1. Eq. 1 is rational for BNNs with  $\mathcal{A}$  and  $\mathbf{w}$  as bilinear coupled variables, since  $\mathbf{w}$  is the variable and  $\mathbf{b}^{\mathbf{w}}$  is just the sign of  $\mathbf{w}$ . However, such bilinear constraints will lead to an asynchronous convergence problem and directly affect the learning process of  $\mathcal{A}$  and  $\mathbf{w}$ . We can know that the variable with a slower convergence speed (usually  $\mathbf{w}$ ) is **not as sufficiently** trained as another faster one. Moreover, BNNs are based on non-convex optimization, and will suffer more from local minima problem due to such an asynchronous convergence. A powerful instance is that  $\mathbf{w}$  will tendentiously fall into the local optimum with low magnitude when the magnitude of  $\mathcal{A}$  is much larger than 0 (due to  $\mathbf{b}^{\mathbf{w}} \in \{-1, +1\}$ ). On the contrary,  $\mathbf{w}$  will have a large magnitude and thus slowly converge when elements of  $\mathcal{A}$  are close to 0.

In this paper, we introduce a recurrent bilinear optimization for binary neural networks (RBONNs) by learning the coupled scale factor and real-valued weight end-to-end. More specifically, recurrent optimization can efficiently backtrack the weights, which will be more sufficiently trained than conventional methods. To this end, a Density-ReLU (DReLU) is introduced to activate the optimization process based on the density of the variable  $\mathcal{A}$ . In this way, we achieve a controlled learning process with a backtracking mechanism by considering the interaction of variables, thus avoiding the local minima and reaching the performance limit of BNNs, as shown in Fig. 1. Our contributions can be summarized as

- We are the first attempt to address BNNs as a bilinear optimization problem. A recurrent bilinear optimization is introduced for BNNs (RBONNs), which can more sufficiently train BNNs and approach its performance limit.
- A Density-ReLU (DReLU) is introduced to activate the optimization process based on the interaction of BNN variables, which can efficiently improve the training process of BNNs.
- Extensive experiments show that the proposed RBONN outperforms state-of-the-art BNNs on a variety of tasks, including image classification and object detection. For instance, on ImageNet, the 1-bit ResNet-18 achieved by RBONN obtains 66.7% Top-1 accuracy, outperforming all prior BNNs and achieving a new state-of-the-art.

## 2 Related Work

**Bilinear Models in Deep Learning.** Under certain circumstances, bilinear models can be used in CNNs. One important application, network pruning, is among the hottest topics in the deep learning community [21,30]. The vital feature maps and related channels are pruned using bilinear models [30]. Iterative methods, *e.g.*, the Fast Iterative Shrinkage-Thresholding Algorithm (FISTA) [21] and the Accelerated Proximal Gradient (APG) [14] can be used to prune bilinear-based networks. Many deep learning applications, such as fine-grained

categorization [23,17], visual question answering (VQA) [46], and person re-identification [39], are promoted by embedding bilinear models into CNNs, which model pairwise feature interactions and fuse multiple features with attention.

**Binary Neural Network.** Based on BinaryConnect, BinaryNet [3] trains CNNs with binary parameters. By binarizing the weights and inputs of the convolution layer, the XNOR-Net [34] improves the efficiency of CNNs. Based on a discrete backpropagation process, a binarization approach is proposed in [10] to learn improved BNNs.

ReActNet [28] substitutes the traditional sign function and PReLU [11] with RSign and RReLU based on learnable thresholds, resulting in improved BNN performance. RBNN [19] rotates the real-valued weights for fruitful information, thus improving the feature representation of BNNs. SLB [45] introduces the NAS [26] into the binarization of weights.

Unlike prior work, our work is the first attempt to solve BNNs as a bilinear optimization problem. We achieve training BNNs sufficiently to bridge the performance gap between them and their real-valued equivalents.

### 3 Methodology

In this section, we describe RBONN in detail. We first describe the bilinear model of BNNs, then introduce a recurrent bilinear optimization method to calculate BNNs, followed by a summary of the whole training process. For a better presentation of our approach, we first briefly describe the preliminaries.

#### 3.1 Preliminaries

In a specific convolution layer,  $\mathbf{w} \in \mathbb{R}^{C_{out} \times C_{in} \times K \times K}$ ,  $\mathbf{a}_{in} \in \mathbb{R}^{C_{in} \times W_{in} \times H_{in}}$ , and  $\mathbf{a}_{out} \in \mathbb{R}^{C_{out} \times W_{out} \times H_{out}}$  represent its weights and feature maps, where  $C_{in}$  and  $C_{out}$  represents the number of channels.  $(H, W)$  are the height and width of the feature maps, and  $K$  denotes the kernel size. We then have

$$\mathbf{a}_{out} = \mathbf{a}_{in} \otimes \mathbf{w}, \quad (2)$$

where  $\otimes$  is the convolution operation. We omit the batch normalization (BN) and activation layers for simplicity. The 1-bit model aims to quantize  $\mathbf{w}$  and  $\mathbf{a}_{in}$  into  $\mathbf{b}^{\mathbf{w}} \in \{-1, +1\}^{C_{out} \times C_{in} \times K \times K}$  and  $\mathbf{b}^{\mathbf{a}_{in}} \in \{-1, +1\}^{C_{in} \times W_{in} \times H_{in}}$  using the efficient XNOR and Bit-count operations to replace real-valued operations. Following [34], the forward process of the BNN is

$$\mathbf{a}_{out} = \mathbf{b}^{\mathbf{a}_{in}} \odot (\mathcal{A}^{-1} \mathbf{b}^{\mathbf{w}}), \quad (3)$$

where  $\odot$  denotes the efficient XNOR and Bit-count operations. We divide the data flow in BNNs into units for detailed discussions. In BNNs, the original output  $\mathbf{a}_{out}$  is first scaled by a channel-wise scale factor (matrix)  $\mathcal{A} = \text{diag}(\frac{1}{\alpha_1}, \dots, \frac{1}{\alpha_{C_{out}}}) \in \mathbb{R}_+^{C_{out} \times C_{out}}$  to modulate the amplitude of full-precision counterparts. It then enters several non-linear layers, *e.g.*, BN layer, non-linear activation layer,

and max-pooling layer. We omit these for simplification. And then, the output is  $\mathbf{a}_{out}$  via the sign function. Then,  $\mathbf{b}^{\mathbf{a}_{out}}$  can be utilized for the efficient operations of the next layer.

### 3.2 Bilinear Model of BNNs

We formulate the optimization of BNNs as

$$\arg \min_{\mathbf{w}, \mathcal{A}} L_S(\mathbf{w}, \mathcal{A}) + \lambda G(\mathbf{w}, \mathcal{A}), \quad (4)$$

where  $\lambda$  is the hyper-parameter.  $G$  contains the bilinear part as mentioned in Eq. 1.  $\mathbf{w}$  and  $\mathcal{A}$  formulate a pair of coupled variables. Thus, the conventional gradient descent method can be used to solve the bilinear optimization problem as

$$\mathcal{A}^{t+1} = |\mathcal{A}^t - \eta_1 \frac{\partial L}{\partial \mathcal{A}^t}|, \quad (5)$$

$$\begin{aligned} \left(\frac{\partial L}{\partial \mathcal{A}^t}\right)^T &= \left(\frac{\partial L_S}{\partial \mathcal{A}^t}\right)^T + \lambda \left(\frac{\partial G}{\partial \mathcal{A}^t}\right)^T, \\ &= \left(\frac{\partial L_S}{\partial \mathbf{a}_{out}^t} \frac{\partial \mathbf{a}_{out}^t}{\partial \mathcal{A}^t}\right)^T + \lambda \mathbf{w}^t (\mathcal{A}^t \mathbf{w}^t - \mathbf{b}^{\mathbf{w}^t})^T, \\ &= \left(\frac{\partial L_S}{\partial \mathbf{a}_{out}^t}\right)^T (\mathbf{b}^{\mathbf{a}_{out}^t} \odot \mathbf{b}^{\mathbf{w}^t}) (\mathcal{A}^t)^{-2} + \lambda \mathbf{w}^t \hat{G}(\mathbf{w}^t, \mathcal{A}^t), \end{aligned} \quad (6)$$

where  $\eta_1$  is the learning rate,  $\hat{G}(\mathbf{w}^t, \mathcal{A}^t) = (\mathcal{A}^t \mathbf{w}^t - \mathbf{b}^{\mathbf{w}^t})^T$ . Conventional gradient descent algorithm for bilinear models iteratively optimizes one variable while keeping the other fixed. This is actually a sub-optimal solution due to ignoring the relationship of the two hidden variables in optimization. For example, when  $\mathbf{w}$  approaches zero due to the sparsity regularization term  $R(\mathbf{w})$ ,  $\mathcal{A}$  will have a larger magnitude due to  $G$  (Eq. 1). Consequently, both the first and second values of Eq. 6 will be suppressed dramatically, causing the gradient vanishing problem for  $\mathcal{A}$ . Contrarily, if  $\mathcal{A}$  changes little during optimization,  $\mathbf{w}$  will also suffer from the vanished gradient problem due to the supervision of  $G$ , causing a local minima. Due to the coupling relationship of  $\mathbf{w}$  and  $\mathcal{A}$ , the gradient calculation for  $\mathbf{w}$  is challenging.

### 3.3 Recurrent Bilinear Optimization

We solve the problem in Eq. 1 from a new perspective that  $\mathbf{w}$  and  $\mathcal{A}$  are coupled. We aim to prevent  $\mathcal{A}$  from going denser and  $\mathbf{w}$  from going sparser, as analyzed above. Firstly, based on the chain rule and its notations in [32], we have the scalar form of the update rule for  $\hat{\mathbf{w}}_{i,j}$  as

$$\begin{aligned} \hat{\mathbf{w}}_{i,j}^{t+1} &= \mathbf{w}_{i,j}^t - \eta_2 \frac{\partial L_S}{\partial \mathbf{w}_{i,j}^t} - \eta_2 \lambda \left( \frac{\partial G}{\partial \mathbf{w}_{i,j}^t} + Tr\left(\left(\frac{\partial G}{\partial \mathcal{A}^t}\right)^T \frac{\partial \mathcal{A}^t}{\partial \mathbf{w}_{i,j}^t}\right) \right), \\ &= \mathbf{w}_{i,j}^{t+1} - \eta_2 \lambda Tr(\mathbf{w}^t \hat{G}(\mathbf{w}^t, \mathcal{A}^t) \frac{\partial \mathcal{A}^t}{\partial \mathbf{w}_{i,j}^t}), \end{aligned} \quad (7)$$

which is based on  $\mathbf{w}_{i,j}^{t+1} = \mathbf{w}_{i,j}^t - \eta_2 \frac{\partial L}{\partial \mathbf{w}_{i,j}^t}$ .  $\hat{\mathbf{w}}^{t+1}$  denotes  $\mathbf{w}$  at the  $t+1$ -th iteration when considering the coupling of  $\mathbf{w}$  and  $\mathcal{A}$ . When computing the gradient of the coupled variable  $\mathbf{w}$ , the gradient of its coupled variable  $\mathcal{A}$  should also be considered using the chain rule. Vanilla  $\mathbf{w}^{t+1}$  denotes the computed  $\mathbf{w}$  at  $t+1$ -th iteration without considering the coupling relationship. Here we denote  $I = C_{out}$  and  $J = C_{in} \times K \times K$  for simplicity. With writing  $\mathbf{w}$  into a row vector  $[\mathbf{w}_1, \dots, \mathbf{w}_I]^T$  and writing  $\hat{G}$  into a column vector  $[\hat{g}_1, \dots, \hat{g}_I]$  and using  $i = 1, \dots, I$  and  $j = 1, \dots, J$ , we can see that  $\mathcal{A}_{i,i}$  and  $\mathbf{w}_{nj}$  are independent when  $\forall n \neq j$ . Omitting superscript  $\cdot^t$ , we have the  $i$ -th component of  $\frac{\partial \mathcal{A}}{\partial \mathbf{w}}$  as

$$\left(\frac{\partial \mathcal{A}}{\partial \mathbf{w}}\right)_i = \begin{bmatrix} 0 & \dots & \cdot & \dots & 0 \\ \frac{\partial \mathcal{A}_{i,i}}{\partial \mathbf{w}_{i,1}} & \dots & \frac{\partial \mathcal{A}_{i,i}}{\partial \mathbf{w}_{i,j}} & \dots & \frac{\partial \mathcal{A}_{i,i}}{\partial \mathbf{w}_{i,J}} \\ \cdot & & \cdot & & \cdot \\ 0 & \dots & \cdot & \dots & 0 \end{bmatrix}, \quad (8)$$

we can derive

$$\mathbf{w}\hat{G}(\mathbf{w}, \mathcal{A}) = \begin{bmatrix} \mathbf{w}_1 \hat{g}_1 & \dots & \mathbf{w}_1 \hat{g}_i & \dots & \mathbf{w}_1 \hat{g}_I \\ \cdot & & \cdot & & \cdot \\ \cdot & & \cdot & & \cdot \\ \cdot & & \cdot & & \cdot \\ \mathbf{w}_I \hat{g}_1 & \dots & \mathbf{w}_I \hat{g}_i & \dots & \mathbf{w}_I \hat{g}_I \end{bmatrix}. \quad (9)$$

Combine Eq. 8 and Eq. 9, we get

$$\mathbf{w}\hat{G}(\mathbf{w}, \mathcal{A})\left(\frac{\partial \mathcal{A}}{\partial \mathbf{w}}\right)_i = \begin{bmatrix} \mathbf{w}_1 \hat{g}_i \frac{\partial \mathcal{A}_{i,i}}{\partial \mathbf{w}_{i,1}} & \dots & \dots & \mathbf{w}_1 \hat{g}_i \frac{\partial \mathcal{A}_{i,i}}{\partial \mathbf{w}_{i,j}} \\ \cdot & & \cdot & \cdot \\ \mathbf{w}_i \hat{g}_i \frac{\partial \mathcal{A}_{i,i}}{\partial \mathbf{w}_{i,1}} & \dots & \dots & \mathbf{w}_i \hat{g}_i \frac{\partial \mathcal{A}_{i,i}}{\partial \mathbf{w}_{i,J}} \\ \cdot & & \cdot & \cdot \\ \mathbf{w}_I \hat{g}_i \frac{\partial \mathcal{A}_{i,i}}{\partial \mathbf{w}_{i,1}} & \dots & \dots & \mathbf{w}_I \hat{g}_i \frac{\partial \mathcal{A}_{i,i}}{\partial \mathbf{w}_{i,J}} \end{bmatrix}. \quad (10)$$

After that, the  $i$ -th component of the trace item in Eq. 7 is then calculated by:

$$Tr[\mathbf{w}\hat{G}\left(\frac{\partial \mathcal{A}}{\partial \mathbf{w}}\right)_i] = \mathbf{w}_i \hat{g}_i \sum_{j=1}^J \frac{\partial \mathcal{A}_{i,i}}{\partial \mathbf{w}_{i,j}} \quad (11)$$

Combining Eq. 7 and Eq. 11, we can get

$$\begin{aligned} \hat{\mathbf{w}}^{t+1} &= \mathbf{w}^{t+1} - \eta_2 \lambda \begin{bmatrix} \hat{g}_1^t \sum_{j=1}^J \frac{\partial \mathcal{A}_{1,1}^t}{\partial \mathbf{w}_{1,j}^t} \\ \cdot \\ \cdot \\ \cdot \\ \hat{g}_I^t \sum_{j=1}^J \frac{\partial \mathcal{A}_{I,I}^t}{\partial \mathbf{w}_{I,j}^t} \end{bmatrix} \otimes \begin{bmatrix} \mathbf{w}_1^t \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{w}_I^t \end{bmatrix} \\ &= \mathbf{w}^{t+1} + \eta_2 \lambda \mathbf{d}^t \otimes \mathbf{w}^t, \end{aligned} \quad (12)$$

where  $\eta_2$  is the learning rate of real-valued weight filters  $\mathbf{w}_i$ ,  $\otimes$  denotes the Hadamard product. We take  $\mathbf{d}^t = -[\hat{g}_1^t \sum_{j=1}^J \frac{\partial \mathcal{A}_{1,1}^t}{\partial \mathbf{w}_{1,j}^t}, \dots, \hat{g}_I^t \sum_{j=1}^J \frac{\partial \mathcal{A}_{I,I}^t}{\partial \mathbf{w}_{I,j}^t}]^T$ , which

is unsolvable and undefined in the back propagation of BNNs. To address this issue, we employ a recurrent model to approximate  $d^t$  and have

$$\hat{\mathbf{w}}^{t+1} = \mathbf{w}^{t+1} + U^t \circ DReLU(\mathbf{w}^t, \mathcal{A}^t), \quad (13)$$

and

$$\mathbf{w}^{t+1} \leftarrow \hat{\mathbf{w}}^{t+1}, \quad (14)$$

where we introduce a hidden layer with channel-wise learnable weights  $U \in \mathbb{R}_+^{C_{out}}$  to recurrently backtrack the  $\mathbf{w}$ . To realize a controllable recurrent optimization, we present  $DReLU$  to supervise such an optimization process. We channel-wise implement  $DReLU$  as

$$DReLU(\mathbf{w}_i, \mathcal{A}_i) = \begin{cases} \mathbf{w}_i & \text{if } (\neg D(\mathbf{w}'_i)) \wedge D(\mathcal{A}_i) = 1, \\ 0 & \text{otherwise,} \end{cases} \quad (15)$$

where  $\mathbf{w}' = \text{diag}(\|\mathbf{w}_1\|_1, \dots, \|\mathbf{w}_{C_{out}}\|_1)$ . And we judge when an asynchronous convergence happens in the optimization based on  $(\neg D(\mathbf{w}'_i)) \wedge D(\mathcal{A}_i) = 1$ , where the density function is defined as

$$D(\mathbf{x}_i) = \begin{cases} 1 & \text{if } \text{ranking}(\sigma(\mathbf{x})_i) > \mathcal{T}, \\ 0 & \text{otherwise,} \end{cases} \quad (16)$$

where  $\mathcal{T}$  is defined by  $\mathcal{T} = \text{int}(C_{out} \times \tau)$ .  $\tau$  is the hyper-parameter denoting the threshold.  $\sigma(\mathbf{x})_i$  denotes the  $i$ -th eigenvalue of diagonal matrix  $\mathbf{x}$ , and  $\mathbf{x}_i$  denotes the  $i$ -th row of matrix  $\mathbf{x}$ . Finally, we define the optimization of  $U$  and as

$$U^{t+1} = |U^t - \eta_3 \frac{\partial L}{\partial U^t}|, \quad (17)$$

$$\frac{\partial L}{\partial U^t} = \frac{\partial L_S}{\partial \mathbf{w}^t} \circ DReLU(\mathbf{w}^{t-1}, \mathcal{A}^t), \quad (18)$$

where  $\eta_3$  is the learning rate of  $U$ . We elaborate on the training process of RBONN outlined in Algorithm 1.

### 3.4 Discussion

In this section, we first review the related methods on “gradient approximation” of BNNs, and then further discuss the difference of RBONN with the related methods and analyze the effectiveness of the proposed RBONN.

In particular, BNN [3] directly unitize the Straight-Through-Estimator in training stage to calculate the gradient of weights and activations as

$$\frac{\partial \mathbf{b}^{\mathbf{w}_{i,j}}}{\partial \mathbf{w}_{i,j}} = 1_{|\mathbf{w}_{i,j}| < 1}, \quad \frac{\partial \mathbf{b}^{\mathbf{a}_{i,j}}}{\partial \mathbf{a}_{i,j}} = 1_{|\mathbf{a}_{i,j}| < 1} \quad (19)$$

**Algorithm 1** RBONN training.

---

**Input:** a minibatch of inputs and their labels, real-valued weights  $\mathbf{w}$ , recurrent model weights  $U$ , scale factor matrix  $\mathcal{A}$ , learning rates  $\eta_1$ ,  $\eta_2$  and  $\eta_3$ .

**Output:** updated real-valued weights  $\mathbf{w}^{t+1}$ , updated scale factor matrix  $\mathcal{A}^{t+1}$ , and updated recurrent model weights  $U^{t+1}$ .

```

1: while Forward propagation do
2:    $\mathbf{b}^{\mathbf{w}^t} \leftarrow \text{sign}(\mathbf{w}^t)$ .
3:    $\mathbf{b}^{\mathbf{a}_{in}^t} \leftarrow \text{sign}(\mathbf{a}_{in}^t)$ .
4:   Features calculation using Eq. 3
5:   Loss calculation using Eq. 4
6: end while
7: while Backward propagation do
8:   Computing  $\frac{\partial L}{\partial \mathcal{A}^t}$ ,  $\frac{\partial L}{\partial \mathbf{w}^t}$ , and  $\frac{\partial L}{\partial U^t}$  using Eq. 6, 7 and 18.
9:   Update  $\mathcal{A}^{t+1}$ ,  $\mathbf{w}^{t+1}$ , and  $U^{t+1}$  according to Eqs. 5, 13, and 17, respectively.
10: end while

```

---

which suffers from an obvious gradient mismatch between the gradient of the binarization function. Intuitively, Bi-Real Net [29] designs an approximate binarization function can help to relieve the gradient mismatch in the backward propagation as

$$\frac{\partial \mathbf{b}^{\mathbf{a}_{i,j}}}{\partial \mathbf{a}_{i,j}} = \begin{cases} 2 + 2\mathbf{a}_{i,j}, & -1 \leq \mathbf{a}_{i,j} < 0, \\ 2 - 2\mathbf{a}_{i,j}, & 0 \leq \mathbf{a}_{i,j} < 1, \\ 0, & \text{otherwise}, \end{cases} \quad (20)$$

which is termed as ApproxSign function and used for back-propagation gradient calculation of the activation. Compared to the traditional STE, ApproxSign has a close shape to that of the original binarization function sign, and thus the activation gradient error can be controlled to some extent. Likewise, CBCN [25] applies an approximate function to address the gradient mismatch from the sign function. MetaQuant [1] introduces Metalearning to learning the gradient error of weights by a neural network. The IR-Net [33] includes a self-adaptive Error Decay Estimator (EDE) to reduce the gradient error in training, which considers different requirements on different stages of the training process and balances the update ability of parameters and reduction of gradient error. RBNN [19] proposes a training-aware approximation of the sign function for gradient backpropagation.

In summary, prior arts focus on approximating the gradient derived from  $\frac{\partial \mathbf{b}^{\mathbf{a}}}{\partial \mathbf{a}_{i,j}}$  or  $\frac{\partial \mathbf{b}^{\mathbf{w}}}{\partial \mathbf{w}_{i,j}}$ . Differently, our approach focuses on a different perspective of gradient approximation, *i.e.*, gradient from  $\frac{\partial G}{\partial \mathbf{w}_{i,j}}$ . Our goal is to decouple  $\mathcal{A}$  and  $\mathbf{w}$  to improve the gradient calculation of  $\mathbf{w}$ . RBONN manipulates  $\mathbf{w}$ 's gradient from its bilinear coupling variable  $\mathcal{A}$  ( $\frac{\partial G(\mathcal{A})}{\partial \mathbf{w}_{i,j}}$ ). More specifically, our RBONN can be combined with prior arts, by comprehensively considering  $\frac{\partial L_S}{\partial \mathbf{a}_{i,j}}$ ,  $\frac{\partial L_S}{\partial \mathbf{w}_{i,j}}$  and  $\frac{\partial G}{\partial \mathbf{w}_{i,j}}$  in the back propagation process.



## 4 Experiments

Our RBONNs are evaluated first on image classification and then on object detection tasks. First, we introduce the implementation details of RBONNs. Then we validate the effectiveness of components in the ablation study. Finally, we illustrate the superiority of RBONNs by comparing our method with state-of-the-art BNNs on various tasks.

### 4.1 Datasets and Implementation Details

**Datasets.** For its huge scope and diversity, the ImageNet object classification dataset [37] is more demanding, which has 1000 classes, 1.2 million training photos, and 50k validation images.

Natural images from 20 different classes are included in the VOC datasets. We use the VOC `trainval2007` and VOC `trainval2012` sets to train our model, which contains around 16k images, and the VOC `test2007` set to evaluate our IDa-Det, which contains 4952 images. We utilize the mean average precision (mAP) as the evaluation matrices, as suggested by [6].

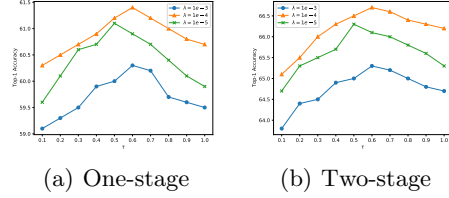
The COCO dataset includes images from 80 different categories. All of our COCO dataset experiments are performed on the object detection track of the COCO `trainval35k` training dataset, which consists of 80k images from the COCO `train2014` dataset and 35k images sampled from the COCO `val2014` dataset. We report the average precision (AP) for IoUs  $\in [0.5 : 0.05 : 0.95]$ , designated as  $\text{mAP}@[.5, .95]$ , using COCO’s standard evaluation metric. For further analyzing our method, we also report  $\text{AP}_{50}$ ,  $\text{AP}_{75}$ ,  $\text{AP}_s$ ,  $\text{AP}_m$ , and  $\text{AP}_l$ .

**Implementation Details.** PyTorch [31] is used to implement RBONN. We run the experiments on 4 NVIDIA GTX 2080Ti GPUs with 11 GB memory. Following [29], we retain the first layer, shortcut, and last layer in the networks as real-valued. We modify the architecture of the BNNs with extra shortcuts, and PReLU [11] following [29] and [10], respectively.

For the image classification task, ResNets [12] and ReActNets [28] are employed as the backbone networks to build our RBONNs. We offer two implementation setups for fair comparison. First, we use **one-stage training** on ResNets, using Adam as the optimization algorithm, and a weight decay of  $1e-5$ .  $\eta_1$  and  $\eta_2$  are both set to  $1e-3$ .  $\eta_3$  is set as  $1e-4$ . The learning rates are optimized by the annealing cosine learning rate schedule. The number of epochs is set as 200. Then, we employ **two-stage training** on ReActNets following [28]. Each stage counts 256 epochs. Thus the number of epochs is set as 512. In this implementation, Adam is selected as the optimizer. And the network is supervised by real-valued ResNet-34 teacher. The weight decay is set following [28]. The learning rates  $\{\eta_1, \eta_2, \eta_3\}$  are set as  $\{1e-3, 1e-4, 1e-4\}$  respectively and annealed to 0 by linear descent.

We use the Faster-RCNN [35] and SSD [27] detection frameworks, which are based on ResNet-18 [12] and VGG-16 [38] backbone, respectively, to train our

RBONN for the object detection task. We pre-train the backbone for image classification using ImageNet ILSVRC12 [37] and fine-tune the detector on the dataset for object detection. For SSD and Faster-RCNN, the batch size is set to 16 and 8, respectively, with applying SGD optimizer. Both  $\eta_1$  and  $\eta_2$  are equal to 0.008. The value of  $\eta_3$  is set to 0.001. We use the same structure and training settings as BiDet [41] on the SSD framework.



**Fig. 2.** Effect of hyper-parameters  $\lambda$  and  $\tau$  on one-stage and two-stage training using 1-bit ResNet-18.

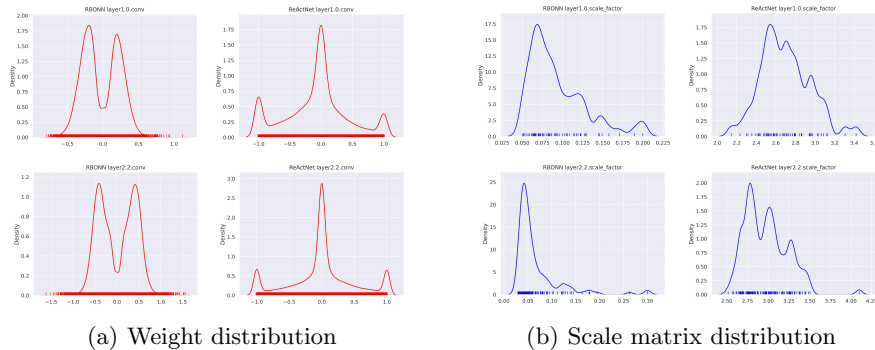
## 4.2 Ablation Study

**Hyper-parameter  $\lambda$  and  $\tau$ .** The most important hyper-parameter of RBONN are  $\lambda$  and  $\tau$ , which control the proportion of  $L_R$  and the threshold of backtracking in recurrent bilinear optimization. On ImageNet for 1-bit ResNet-18, the effect of hyper-parameters  $\lambda$  and  $\tau$  is evaluated under both one-stage and two-stage training. RBONN’s performance is demonstrated in Fig. 2, where  $\lambda$  is varied from  $1e-3$  to  $1e-5$  and  $\tau$  is varied from 1 to 0.1. As can be observed, with  $\lambda$  reducing, performance improves at first before plummeting. When we increase  $\tau$  in both implementations, the same trend emerges. As demonstrated in Fig. 2, when  $\lambda$  is set to  $1e-4$  and  $\tau$  is set to 0.6, 1-bit ResNet-18 generated by our RBONN gets the best performance. As a result, we apply this set of hyper-parameters to the remaining experiments in this paper. Note that the recurrent model does not effect when  $\tau$  is set as 1.

**Weight and Scale factor Distribution.** We first analyze the weight distribution of training ReActNet [28] and RBONN for comparison to analyze the sparsity of  $\mathbf{w}$ . For a 1-bit ResNet-18, we analyze the 1-st and 6-th 1-bit convolution layer of ResNet-18. The distribution of weights (before binarization) for ReActNet and our RBONN is shown in the left section of Fig. 3. The weight values for ReActNet can be seen to be closely mixed up around the zero centers, and the value magnitude remains sparse. Thus the binarization results are far less robust to any possible disturbance. In contrast, our RBONN gains weight forming a bi-modal distribution, which achieves its robustness against disturbances. Moreover, we plot the distribution of non-zero elements in scale matrix  $\mathcal{A}$  in the right part of Fig. 3. The scale values of our RBONN is less dense compared with ReActNet. Thus, the result demonstrates that our RBONN prevents  $\mathcal{A}$  from going denser and  $\mathbf{w}$  from going sparser, which validates our motivation.

## 4.3 Image Classification

We first show the experimental results on ImageNet with ResNet-18 [12] backbone in Tab. 1. We compare RBONN with BNN [2], XNOR-Net [34], Bi-Real Net



**Fig. 3.** Weight (red) and scale matrix (blue) distribution of the RBONN and ReActNet in 1-bit ResNet-18 with two-stage training.

[29], IR-Net [33], BONN[47], RBCN [24], and RBNN [19]. We also report multi-bit DoReFa-Net [48], and TBN [40] for further reference. RBONN outperforms all of the evaluated binary models in both Top-1 and Top-5 accuracy, as shown in Tab. 1. RBONN achieves 61.4% and 83.4% in Top-1 and Top-5 accuracy using ResNet-18, respectively, with 1.8% and 1.9% increases over state-of-the-art RBNN. In this paper, we use memory usage and OPs following [29] in comparison to other tasks for further reference. We also analyze the inference speed on hardware in Sec. 4.5.

Furthermore, we compare with ReActNet [28] using the same architecture. It uses full-precision parameters, data augmentation, knowledge distillation, and a computationally intensive two-step training setting with 512 epochs in total. We use the same implementation as [28] to evaluate our RBONN to ReActNet. As shown in Tab. 2, our method still achieves an impressive 0.8% Top-1 accuracy improvement on the same ResNet-18 backbone, which verifies the effectiveness of our method. Also, our method outperforms state-of-the-art ReCU [44] by 0.3% Top-1 accuracy. Moreover, we evaluate the performance of our RBONN on another strong backbone, *i.e.*, ReActNet-A. Our strategy improves Top-1 accuracy by 1.2% on ReActNet-A, which is substantial on the ImageNet dataset classification challenge.

In a word, when compared to several BNN methods, our RBONN achieves the best performance on the large-scale ImageNet dataset, proving that our method achieves a new state-of-the-art on image classification tasks.

#### 4.4 Object Detection

**PASCAL VOC.** On the PASCAL VOC datasets, we compare the proposed RBONN against existing state-of-the-art BNNs, such as XNOR-Net [34], Bi-Real-Net [29], and BiDet [41], on the same framework for object detection. The detection result of multi-bit quantized networks DoReFa-Net [48] is also

Network	Method	W/A	OPs ( $\times 10^8$ )	Top-1	Top-5
ResNet-18	Real-valued	32/32	18.19	69.6	89.2
	DoReFa-Net	1/4	2.44	59.2	81.5
	TBN	1/2	1.81	55.6	79.0
	BNN	1/1	1.63	42.2	67.1
	XNOR-Net			51.2	73.2
	Bi-Real Net			56.4	79.5
	IR-Net			58.1	80.0
	BONN			59.3	81.6
	RBCN			59.5	81.6
	RBNN			59.6	81.6
	<b>RBONN</b>			<b>61.4</b>	<b>83.5</b>

**Table 1.** A performance comparison with SOTAs on ImageNet with one-stage training. W/A denotes the bit length of weights and activations. We report the Top-1 (%) and Top-5 (%) accuracy performances.

Network	Method	W/A	OPs( $\times 10^8$ )	Top-1	Top-5
ResNet-18	Real-valued	32/32	18.19	69.6	89.2
	ReActNet	1/1	1.63	65.9	-
	ReCU			66.4	86.5
	<b>RBONN</b>			<b>66.7</b>	<b>87.0</b>
ReActNet-A	Real-valued	32/32	48.32	72.4	-
	ReActNet	1/1	0.87	69.4	-
	<b>RBONN</b>			<b>70.6</b>	<b>89.0</b>

**Table 2.** A performance comparison with ReActNet [28] on ImageNet using two-stage training. W/A denotes the bit length of weights and activations. We report the Top-1 (%) and Top-5 (%) accuracy performances.

reported. In Tab. 3, we show the results for 1-bit Faster-RCNN [35] on VOC `test2007` from lines 2 to 7. With  $50.63\times$  and  $19.87\times$  rate, our RBONN greatly accelerates and compresses the Faster-RCNN with ResNet-18 backbone. We see significant improvements with our RBONN over other methods as compared to 1-bit approaches. With the same memory utilization and FLOPs, our RBONN outperforms XNOR-Net, Bi-Real-Net, and BiDet by 17.0%, 7.2%, and 5.9% mAP, which is substantial on the object detection task.

For the SSD [27] with VGG-16 backbone, The bottom section in Tab. 3 shows that our RBONN can save the computation and storage by  $14.76\times$  and  $4.81\times$ , as compared to real-valued alternatives. The difference in performance is rather slight (69.4% *vs.* 74.3%). Moreover, compared with other 1-bit SOTAs, our RBONN’s performance stands out by a sizeable margin. For example, RBONN surpasses BiDet by 3.4% with the same structure and compression.

**COCO.** Because of its size and diversity, the COCO dataset presents a greater challenge than PASCAL VOC. On COCO, our RBONN is compared against state-of-the-art BNNs such as XNOR-Net [34], Bi-Real Net [29], and BiDet [41]. We present the performance of the 4-bit DoReFa-Net [48] for comparison. Tab. 4 does not indicate memory use or FLOPs due to page width constraints. With

Framework	Input Resolution	Backbone	Method	W/A	Memory Usage (MB)	OPs ( $\times 10^9$ )	mAP
Faster-RCNN	$1000 \times 600$	ResNet-18	Real-valued	32/32	47.48	434.39	74.6
			DoReFa-Net	4/4	6.73	55.90	71.0
			XNOR-Net	1/1	2.39	8.58	48.4
			Bi-Real Net				58.2
			BiDet				59.5
			<b>RBONN</b>				<b>65.4</b>
SSD	$300 \times 300$	VGG-16	Real-valued	32/32	105.16	31.44	74.3
			DoReFa-Net	4/4	29.58	6.67	69.2
			XNOR-Net	1/1	21.88	2.13	50.2
			Bi-Real Net				58.2
			BiDet				66.0
			<b>RBONN</b>				<b>69.4</b>

**Table 3.** Comparison of memory usage, OPs, and mAP (%) with state-of-the-art BNNs in SOTA binarized detection frameworks on VOC *test2007*.

Framework	Input Resolution	Backbone	Method	mAP @ [.5, .95]	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>s</sub>	AP <sub>m</sub>	AP <sub>l</sub>
Faster R-CNN	$1000 \times 600$	ResNet-18	Real-valued	26.0	44.8	27.2	10.0	28.9	39.7
			DeRoFa-Net	22.9	38.6	23.7	8.0	24.9	36.3
			Xnor-Net	10.4	21.6	8.8	2.7	11.8	15.9
			Bi-Real Net	14.4	29.0	13.4	3.7	15.4	24.1
			BiDet	15.7	31.0	14.4	4.9	16.7	25.4
			<b>RBONN</b>	<b>20.6</b>	<b>37.3</b>	<b>19.9</b>	<b>7.4</b>	<b>21.3</b>	<b>32.8</b>
SSD	$300 \times 300$	VGG-16	Real-valued	23.2	41.2	23.4	5.3	23.2	39.6
			DoReFa-Net	19.5	35.0	19.6	5.1	20.5	32.8
			XNOR-Net	8.1	19.5	5.6	2.6	8.3	13.3
			Bi-Real Net	11.2	26.0	8.3	3.1	12.0	18.3
			BiDet	13.2	28.3	10.5	5.1	14.3	20.5
			<b>RBONN</b>	<b>17.4</b>	<b>33.2</b>	<b>16.4</b>	<b>5.3</b>	<b>17.1</b>	<b>26.7</b>

**Table 4.** Comparison of mAP@ [.5, .95] (%), AP (%) with different IoU threshold and AP for objects in various sizes with SOTA 1-bit detectors on COCO *minival*.

just different fully-connected layers, the COCO dataset’s practical memory utilization and FLOPs are similar to those on VOC.

Compared to state-of-the-art XNOR-Net, Bi-Real Net, and BiDet, our method enhances the mAP@ [.5, .95] by 10.2%, 6.2%, and 4.9% using the Faster-RCNN framework with the ResNet-18 backbone. Moreover, our RBONN clearly outperforms competitors on other APs with various IoU thresholds. Our RBONN achieves only 2.3% lower mAP than DeRoFa-Net, a quantized neural network with 4-bit weights and activations. Our method yields a 1-bit detector with a performance of only 5.4% mAP lower than the best-performing real-valued counterpart (20.6% *vs.* 26.0%). Similarly, using the SSD300 framework with the VGG-16 backbone, our method achieves 17.4% mAP@ [.5, .95], outperforming XNOR-Net, Bi-Real Net, and BiDet by 9.3%, 6.2%, and 4.2% mAP, respectively.

Network	Method	W/A	Size (MB)	Memory Saving	Latency (ms)	Acceleration
ResNet-18	Real-valued	32/32	46.8	-	1060.2	-
	RBONN	1/1	4.2	11.1×	67.1	15.8×
SSD-VGG16	Real-valued	32/32	105.16	-	2788.7	-
	RBONN	1/1	21.88	4.8×	200.5	13.9×

**Table 5.** Comparing RBONN with real-valued models on hardware (single thread).

In conclusion, our method outperforms previous BNN algorithms in the AP with various IoU thresholds and AP for objects of various sizes on COCO, demonstrating the method’s superiority and applicability in a wide range of applications settings.

#### 4.5 Deployment Efficiency

We implement the 1-bit models achieved by our RBONN on ODROID C4, which has a 2.016 GHz 64-bit quad-core ARM Cortex-A55. With evaluating its real speed in practice, the efficiency of our RBONN is proved when deployed into real-world mobile devices. We leverage the SIMD instruction SSSL on ARM NEON to make the inference framework BOLT [7] compatible with RBONN. We compare RBONN to the real-valued backbones in Tab. 5. We can see that RBONN’s inference speed is substantially faster with the highly efficient BOLT framework. For example, the acceleration rate achieves about 15.8× on ResNet-18, which is slightly lower than the theoretical acceleration rate discussed in Sec. 4.3. Furthermore, RBONN achieves 13.9× acceleration with SSD. All deployment results are significant for the computer vision on real-world edge devices.

## 5 Conclusion

This paper proposed a new learning algorithm, termed recurrent bilinear optimization, to efficiently calculate BNNs, which is the first attempt to optimize BNNs from the bilinear perspective. Our method specifically introduces recurrent optimization to sequentially backtrack the sparse real-valued weight filters, which can be sufficiently trained and reach their performance limit based on a controllable learning process. RBONNs show strong generalization to gain impressive performance on both image classification and object detection tasks, demonstrating the superiority of the proposed method over state-of-the-art BNNs.

**Acknowledgement.** This work was supported in part by the National Natural Science Foundation of China under Grant 62076016, 92067204, 62141604 and the Shanghai Committee of Science and Technology under Grant No. 21DZ1100100.

## References

1. Chen, S., Wang, W., Pan, S.J.: Metaquant: Learning to quantize by learning to penetrate non-differentiable quantization. *Proc. of NeurIPS* **32**, 3916–3926 (2019)
2. Courbariaux, M., Bengio, Y., David, J.P.: Binaryconnect: Training deep neural networks with binary weights during propagations. In: *Proc. of NeurIPS*. pp. 3123–3131 (2015)
3. 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: *Proc. of NeurIPS*. pp. 1–9 (2016)
4. Del Bue, A., Xavier, J., Agapito, L., Paladini, M.: Bilinear modeling via augmented lagrange multipliers (balm). *IEEE transactions on pattern analysis and machine intelligence* **34**(8), 1496–1508 (2011)
5. Denil, M., Shakibi, B., Dinh, L., Ranzato, M., De Freitas, N.: Predicting parameters in deep learning. In: *Proc. of NeurIPS*. pp. 2148–2156 (2013)
6. Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The pascal visual object classes (voc) challenge. *International Journal of Computer Vision* **88**(2), 303–338 (2010)
7. Feng, J.: Bolt. <https://github.com/huawei-noah/bolt> (2021)
8. Gao, P., Ma, T., Li, H., Dai, J., Qiao, Y.: Convmae: Masked convolution meets masked autoencoders. *arXiv preprint arXiv:2205.03892* (2022)
9. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: *Proc. of CVPR*. pp. 580–587 (2014)
10. Gu, J., Li, C., Zhang, B., Han, J., Cao, X., Liu, J., Doermann, D.: Projection convolutional neural networks for 1-bit cnns via discrete back propagation. In: *Proc. of AAAI*. pp. 8344–8351 (2019)
11. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *Proc. of ICCV*. pp. 1026–1034 (2015)
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: *Proc. of CVPR*. pp. 770–778 (2016)
13. He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y.: Soft filter pruning for accelerating deep convolutional neural networks. In: *Proc. of IJCAI*. pp. 2234–2240 (2018)
14. Huang, Z., Wang, N.: Data-driven sparse structure selection for deep neural networks. In: *Proc. of ECCV*. pp. 304–320 (2018)
15. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: *Proc. of NeurIPS*. pp. 598–605 (1990)
16. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. In: *Proc. of ICLR*. pp. 1–13 (2016)
17. Li, Y., Wang, N., Liu, J., Hou, X.: Factorized bilinear models for image recognition. In: *Proc. of ICCV*. pp. 2079–2087 (2017)
18. Lin, M., Ji, R., Xu, Z., Zhang, B., Chao, F., Xu, M., Lin, C.W., Shao, L.: Siman: Sign-to-magnitude network binarization. *arXiv preprint arXiv:2102.07981* (2021)
19. Lin, M., Ji, R., Xu, Z., Zhang, B., Wang, Y., Wu, Y., Huang, F., Lin, C.W.: Rotated binary neural network. In: *Proc. of NeurIPS*. pp. 1–9 (2020)
20. Lin, S., Ji, R., Chen, C., Huang, F.: Espace: Accelerating convolutional neural networks via eliminating spatial and channel redundancy. In: *Proc. of AAAI*. pp. 1424–1430 (2017)

21. Lin, S., Ji, R., Yan, C., Zhang, B., Cao, L., Ye, Q., Huang, F., Doermann, D.: Towards optimal structured cnn pruning via generative adversarial learning. In: Proc. of CVPR. pp. 2790–2799 (2019)
22. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: Proc. of ECCV (2014)
23. Lin, T.Y., RoyChowdhury, A., Maji, S.: Bilinear cnn models for fine-grained visual recognition. In: Proc. of ICCV. pp. 1449–1457 (2015)
24. Liu, C., Ding, W., Xia, X., Hu, Y., Zhang, B., Liu, J., Zhuang, B., Guo, G.: Rbcn: Rectified binary convolutional networks for enhancing the performance of 1-bit dcnn. In: Proc. of IJCAI. pp. 854–860 (2019)
25. Liu, C., Ding, W., Xia, X., Zhang, B., Gu, J., Liu, J., Ji, R., Doermann, D.: Circulant binary convolutional networks: Enhancing the performance of 1-bit dcnn with circulant back propagation. In: Proc. of CVPR. pp. 2691–2699 (2019)
26. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. In: Proc. of ICLR (2019)
27. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: Ssd: Single shot multibox detector. In: Proc. of ECCV. pp. 21–37 (2016)
28. Liu, Z., Shen, Z., Savvides, M., Cheng, K.T.: Reactnet: Towards precise binary neural network with generalized activation functions. In: Proc. of ECCV. pp. 143–159 (2020)
29. Liu, Z., Wu, B., Luo, W., Yang, X., Liu, W., Cheng, K.T.: Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In: Proc. of ECCV (2018)
30. Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., Zhang, C.: Learning efficient convolutional networks through network slimming. In: Proc. of ICCV. pp. 2736–2744 (2017)
31. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch. In: NeurIPS Workshops (2017)
32. Petersen, K., Pedersen, M., et al.: The matrix cookbook. Technical University of Denmark **15** (2008)
33. Qin, H., Gong, R., Liu, X., Shen, M., Wei, Z., Yu, F., Song, J.: Forward and backward information retention for accurate binary neural networks. In: Proc. of CVPR. pp. 2250–2259 (2020)
34. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. In: Proc. of ECCV. pp. 525–542 (2016)
35. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **39**(6), 1137–1149 (2016)
36. Romero, A., Ballas, N., Kahou, S.E., Chassang, A., Gatta, C., Bengio, Y.: Fitnets: Hints for thin deep nets. In: Proc. of ICLR. pp. 1–13 (2015)
37. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* **115**(3), 211–252 (2015)
38. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: Proc. of ICLR. pp. 1–13 (2015)
39. Suh, Y., Wang, J., Tang, S., Mei, T., Mu Lee, K.: Part-aligned bilinear representations for person re-identification. In: Proc. of ECCV. pp. 1449–1457 (2018)



40. Wan, D., Shen, F., Liu, L., Zhu, F., Qin, J., Shao, L., Tao Shen, H.: Tbn: Convolutional neural network with ternary inputs and binary weights. In: Proc. of ECCV. pp. 315–332 (2018)
41. Wang, Z., Wu, Z., Lu, J., Zhou, J.: Bidet: An efficient binarized object detector. In: Proc. of CVPR. pp. 2049–2058 (2020)
42. Xu, S., Li, Y., Zhao, J., Zhang, B., Guo, G.: Poem: 1-bit point-wise operations based on expectation-maximization for efficient point cloud processing. In: Proc. of BMVC. pp. 1–10 (2021)
43. Xu, S., Zhao, J., Lu, J., Zhang, B., Han, S., Doermann, D.: Layer-wise searching for 1-bit detectors. In: Proc. of CVPR. pp. 5682–5691 (2021)
44. Xu, Z., Lin, M., Liu, J., Chen, J., Shao, L., Gao, Y., Tian, Y., Ji, R.: Recu: Reviving the dead weights in binary neural networks. In: Proc. of ICCV. pp. 5198–5208 (2021)
45. Yang, Z., Wang, Y., Han, K., Xu, C., Xu, C., Tao, D., Xu, C.: Searching for low-bit weights in quantized neural networks. In: Proc. of NeurIPS. pp. 1–11 (2020)
46. Yu, Z., Yu, J., Fan, J., Tao, D.: Multi-modal factorized bilinear pooling with co-attention learning for visual question answering. In: Proc. of ICCV. pp. 1821–1830 (2017)
47. Zhao, J., Xu, S., Zhang, B., Gu, J., Doermann, D., Guo, G.: Towards compact 1-bit cnns via bayesian learning. *International Journal of Computer Vision* **130**(2), 201–225 (2022)
48. Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y.: Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160 (2016)