Composite Binary Decomposition Networks

You Qiaoben[1], Zheng Wang[2], Jianguo Li[3], Yinpeng Dong[1], Yu-Gang Jiang[2], Jun Zhu[1]
[1]Dept. of Comp. Sci. & Tech., State Key Lab for Intell. Tech. & Sys., Institute for AI, Tsinghua University
[2]School of Computer Science, Fudan University
[3]Intel Labs China
qby_222@126.com, {zhengwang17,ygj}@fudan.edu.cn, jianguo.li@intel.com, {dyp17@mails., dcszj@}tsinghua.edu.cn

Abstract

Binary neural networks have great resource and computing efficiency, while suffer from long training procedure and non-negligible accuracy drops, when comparing to the full-precision counterparts. In this paper, we propose the composite binary decomposition networks (CBDNet), which first compose real-valued tensor of each layer with a limited number of binary tensors, and then decompose some conditioned binary tensors into two low-rank binary tensors, so that the number of parameters and operations are greatly reduced comparing to the original ones. Experiments demonstrate the effectiveness of the proposed method, as CBDNet can approximate image classification network ResNet-18 using 5.25 bits, VGG-16 using 5.47 bits, DenseNet-121 using 5.72 bits, object detection networks SSD300 using 4.38 bits, and semantic segmentation networks SegNet using 5.18 bits, all with minor accuracy drops. [1]

Introduction

With the remarkable improvements of Convolutional Neural Networks (CNNs), varied excellent performance has been achieved in a wide range of pattern recognition tasks, such as image classification (Krizhevsky et al. 2012; Szegedy et al. 2015; He et al. 2016; Huang et al. 2017), object detection (Girshick et al. 2014; Ren et al. 2015; Shen et al. 2017) and semantic segmentation (Long et al. 2015; Badrinarayanan et al. 2017), etc. A well-performed CNN based systems usually need considerable storage and computation power to store and calculate millions of parameters in tens or even hundreds of CNN layers. Therefore,the deployment of CNNs to some resource limited scenarios is hindered, especially low-power embedded devices in the emerging Internet-of-Things (IoT) domain.

Many efforts have been devoted to optimizing the inference resource requirement of CNNs, which can be roughly divided into three categories according to the life cycle of deep models. *First, design-time network optimization* considers designing efficient network structures from scratch in a handcraft way such as MobileNet (Howard et al. 2017), interlacing/shuffle networks (Zhang et al. 2017; 2018), or even automatic search way such as NASNet (Zoph and Le 2016), PNASNet (Liu et al. 2017a). *Second, training-time network optimization* tries to simplify the pre-defined network structures on neural connections (Han et al. 2015; 2016), filter structures (Wen et al. 2016; Li et al. 2017; Liu et al. 2017b), and even weight precisions (Chen et al. 2015; Courbariaux et al. 2016; Rastegari et al. 2016) through regularized retraining or fine-tuning or even knowledge distilling (Hinton et al. 2015). *Third, deploy-time network optimization* tries to replace heavy/redundant components/structures in pre-trained CNN models with efficient/lightweight ones in a training-free way. Typical works include low-rank decomposition (Denton et al. 2014), spatial decomposition (Jaderberg et al. 2014), channel decomposition (Zhang et al. 2016) and network decoupling (Guo et al. 2018).

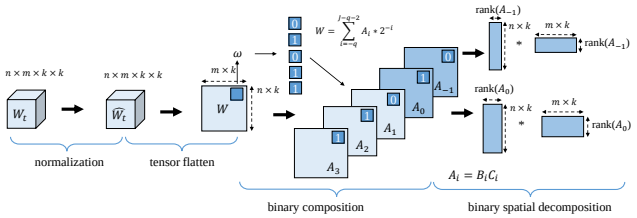


Figure 1: Overall framework illustration of CBDNet.

To produce desired outputs, it is obvious that the first two categories of methods require a time-consuming training procedure with full training-set available, while methods of the third category may not require training-set, or in some cases require a small dataset (e.g., 5000 images) to calibrate some parameters. The optimization process can be typically done within dozens of minutes. Therefore, in case that the customers can't provide training data due to privacy or confidential issues, it is of great value when software/hardware vendors help their customers optimize CNN based solutions. It also opens the possibility for on-device learning to compression, and online learning with new ingress data. In consequence, there is a strong demand for modern deep learning frameworks or hardware (GPU/ASIC/FPGA) vendors to provide deploy-time model optimizing tools.

However, current deploy-time optimization methods can only provide very limited optimization (2∼4× in compression/speedup) over original models. Meanwhile, binary neu-

[1]This work was done when You Qiaoben and Zheng Wang were interns at Intel Labs. Jianguo Li is the corresponding author.

ral networks (Courbariaux et al. 2015; 2016), which aim for training CNNs with binary weights or even binary activations, attract much more attention due to their high compression rate and computing efficiency. However, binary networks generally suffer much from a long training procedure and non-negligible accuracy drops, when comparing to the full-precision (FP32) counterparts. Many efforts have been spent to alleviate this problem in training-time optimization (Rastegari et al. 2016; Zhou et al. 2016). This paper considers the problem from a different perspective via raising the question: *is it possible to directly transfer full-precision networks into binary networks at deploy-time in a training-free way?* We study this problem, and give a positive answer by proposing a solution named composite binary decomposition networks (CBDNet). Figure 1 illustrates the overall framework of the proposed method. The main contributions of this paper are summarized as below:

- We show that full-precision CNN models can be directly transferred into highly parameter and computing efficient multi-bits binary network models in a training-free way by the proposed CBDNet.
- We propose an algorithm to first expand full-precision tensors of each conv-layer with a limited number of binary tensors, and then decompose some conditioned binary tensors into two low-rank binary tensors. To our best knowledge, we are the first to study the network sparsity and the low-rank decomposition in the binary space.
- We demonstrate the effectiveness of CBDNet on different classification networks including VGGNet, ResNet, DenseNet as well as detection network SSD300 and semantic segmentation network SegNet. This verifies that CBDNet is widely applicable.

Related Work

Binary Neural Networks

Binary neural networks (Courbariaux et al. 2015; 2016; Rastegari et al. 2016) with high compression rate and great computing efficiency, have progressively attracted attentions owing to their great inference performance.

Particularly, BinaryConnect (BNN) (Courbariaux et al. 2015) binarizes weights to $+1$ and -1 and substitutes multiplications with additions and subtractions to speed up the computation. As well as binarizing weight values plus one scaling factor for each filter channel, Binary weighted networks (BWN) (Rastegari et al. 2016) extends it to XNOR-Net with both weights and activations binarized. DoReFaNet (Zhou et al. 2016) binarizes not merely weights and activations, but also gradients for the purpose of fast training. However, binary networks are facing the challenge that accuracy may drops non-negligibly, especially for very deep models (e.g., ResNet). In spite of the fact that (Hou et al. 2017) directly consider the loss to mitigate possible accuracy drops to mitigate during binarization, which gain more accurate results than BWN and XNOR-Net, it still has gap to the full-precision counterparts. A novel training procedure named stochastic quantization (Dong et al. 2017) was introduced to narrow down such gaps. All these works belongs to the training-time optimization category in summary.

Deploy-time Network Optimization

Deploy-time network optimization tries to replace some heavy CNN structures in pre-trained CNN models with efficient ones in a training-free way. Low-rank decomposition (Denton et al. 2014) exploits low-rank nature within CNN layers, and shows that fully-connected (FC) layers can be efficiently compressed and accelerated with low-rank approximations, while conv-layers can not. Spatial decomposition (Jaderberg et al. 2014) factorizes the $k_h \times k_w$ convolutional filters into a linear combination of a horizontal filter $1 \times k_w$ and a vertical filter $k_h \times 1$. Channel decomposition (Zhang et al. 2016) decomposes one conv-layer into two layers, while the first layer has the same filter-size but with less channels, and the second layer uses a 1×1 convolution to mix output of the first one. Network decoupling (Guo et al. 2018) decomposes the regular convolution into the successive combination of depthwise convolution and pointwise convolution.

Due to its simplicity, deploy-time optimization has many potential applications for software/hardware vendors as aforementioned. However, it suffers from relatively limited optimization gains ($2\sim4\times$ in compression/speedup) over original full-precision models.

Binary Network Decomposition

Few existing works like us consider transferring full-precision networks into multi-bits binary networks in a training-free way. Binary weighted decomposition (BWD) (Kamiya et al. 2017) takes each filter as a basic unit as BWN, and expands each filter into a linear combination of binary filters and a FP32 scalar. ABC-Net (Lin et al. 2017) approximates full-precision tensor with a linear combination of multiple binary tensors and FP32 scalar weights during training-procedure to obtain multi-bits binary networks. Our method is quite different to these two works. We further consider the redundance and sparsity in the expanded binary tensors, and try to decompose binary tensors. The decomposition is similar to spatial decomposition (Jaderberg et al. 2014) but in the binary space. Hence, our binary decomposition step can also be viewed as binary spatial decomposition.

Method

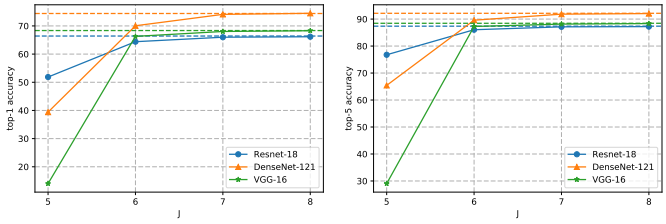
As is known, parameters of each conv-layer in CNNs could be represented as a 4-dimensional (4D) tensor. We take tensor as our study target. We first present how to expand full-precision tensors into a limited number of binary tensors. Then we show some binary tensors that fulfill certain conditions can be decomposed into two low-rank binary tensors, and propose an algorithm for that purpose.

Tensor Binary Composition

Suppose the weight parameters of a conv-layer are represented by a 4D tensor $\mathbf{W}_t \in \mathbb{R}^{n \times k \times k \times m}$, where n is the number of input channels, m is the number of output channels, and $k \times k$ is the convolution kernel size. For each element $w \in \mathbf{W}_t$, we first normalize it with

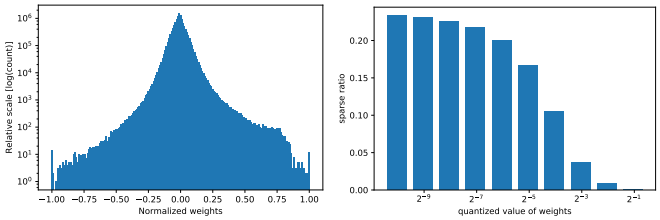
$$\tilde{w} = w / w_{max}, \tag{1}$$

where $w_{max} = \max_{w_i \in \mathbf{W}_t} \{|w_i|\}$. The normalized tensor is denoted as $\tilde{\mathbf{W}}_t$. The normalization makes every element $\tilde{w}$



(a) (b)

Figure 2: Performance on ImageNet for different networks with binary tensor expansion using different J bits. Dashed-line indicates FP32 accuracy. Left is for top-1, right is for top-5.



(a) (b)

Figure 3: Weight distribution of all conv-layers for ResNet-18. (a) normalized weights, (b) sparse-ratio of each binary quantized weight.

within range $[-1, 1]$. For simplicity, we denote the magnitude of $\tilde{w}$ as $\hat{w}$, i.e., $\tilde{w} = sign(\tilde{w})\hat{w}$, where $sign(\cdot)$ is a sign function which equals to -1 when $\tilde{w} < 0$, otherwise 1. And $\hat{w} \in [0, 1]$ can be expressed by the composite of a series of fixed-point basis as

$$\hat{w} \approx \sum_{i=0}^{J-2} a_i * 2^{-i}, \tag{2}$$

where $a_i \in \{0, 1\}$ is a binary coefficient indicating whether certain power-of-two term is activated or not, $i \in \{0, \cdots, J-2\}$ means totally $J-1$ bits is needed for the representation. When taking the sign bit into consideration, $\tilde{w}$ requires J bits to represent.

Denote the magnitude of the normalized tensor as $\hat{\mathbf{W}}_t$. Tensor binary composition is a kind of tensor expansion, when each element of the tensor is binary expanded and expressed by the same bit rate J as

$$\hat{\mathbf{W}}_t \approx \sum_{i=0}^{J-2} A_i * 2^{-i}, \tag{3}$$

where $A_i \in \{0, 1\}^{n \times k \times k \times m}$ is 4D binary tensor. J will impact the approximation accuracy, while larger J gives more accurate results. We empirically study three different ImageNet CNN models. Figure 2 shows that $J = 7$ is already sufficiently good to keep a balance between the accuracy and the efficiency of the expansion.

Different A_i may have different sparsity, which could be further utilized to compress the binary tensor. Figure 3a illustrates the distribution of normalized weights $\tilde{w}$ in all the layers of ResNet-18, which looks like a Laplacian distribution, where most weight values concentrate in the range $(-0.5, 0.5)$. This suggests that 1 is very rare in some binary tensor A_i with smaller i, since smaller i corresponds to bigger values in the power-of-two expansion. Figure 3b further shows the average sparsity of each binary tensor A_i, which also verifies that A_i with smaller i is much more sparse. Due to the sparsity of A_i, we next perform binary tensor decomposition to further reduce the computation complexity, as introduced in the next section.

Binary expansion with α scaling factor The non-saturation direct expansion from FP32 to low-bits will yield non-negligible accuracy loss as shown in (Migacz 2017; Krishnamoort 2018). A scaling factor is usually introduced and learnt to minimize the loss through an additional calibration procedure (Migacz 2017; Krishnamoort 2018). Similarly, we impose a scaling factor α to Eq.(1) as

$$\tilde{w} = \alpha * w / w_{max}, \tag{4}$$

where $\alpha \geq 1$ is a parameter to control the range of $\tilde{w} \in [-\alpha, \alpha]$. When the scaling factor α is allowed, the normalized weight $\hat{w} \in [0, \alpha]$ can be expressed with a composite of power-of-two terms as below:

$$\hat{w} \approx \sum_{i=-q}^{J-q-2} a_i * 2^{-i}, \tag{5}$$

where $q = \lceil \log_2 \alpha \rceil$ and J also denotes the number bits of the weight, including $J-1$ bits for magnitude and 1 sign bit. The corresponding tensor form can be written as

$$\hat{\mathbf{W}}_t^{\alpha} \approx \sum_{i=-q}^{J-q-2} A_i * 2^{-i}. \tag{6}$$

Note that the scaling factor α will shift the power-of-two bases from $\{2^0, 2^{-1}, \cdots, 2^{-J+2}\}$ for Eq.(3) to $\{2^q, \cdots, 2^0, \cdots, 2^{-J+q+2}\}$ for Eq.(6). When $\alpha = 1$, we have $q = \lceil \log_2 \alpha \rceil = 0$, which makes Eq.(6) reduce to the case without scaling factor as in Eq.(3).

Binary Tensor Decomposition

We have shown that some binary tensors A_i are sparse. As sparse operations require specific hardware/software accelerators, it is not preferred in many practical usages. In deploy-time network optimization, researches show that full-precision tensor could be factorized into two much smaller and more efficient tensors (Jaderberg et al. 2014; Zhang et al. 2016). Here, we attempt to extend the spatial-decomposition (Jaderberg et al. 2014) to our binary case.

For the simplicity of analysis, we flatten the 4D tensor $\hat{\mathbf{W}}_t \in \mathbb{R}^{n \times k \times k \times m}$ into the weight matrix $\mathbf{W} \in \mathbb{R}^{(n \times k) \times (k \times m)}$, so does for each A_i. Here the matrix height and width are $n \times k$ and $k \times m$ respectively. We then factorize a sparse matrix A into two smaller matrices as

$$A = B * C, \tag{7}$$

where matrix $B \in \{0, 1\}^{(n \times k) \times c}$ and matrix $C \in \{0, 1\}^{c \times (k \times m)}$. Note that our method is significantly different from the vector decomposition method (Kamiya et al.

2017), which keeps B binary, the other full-precision. On the contrary, we keep both B and C binary. This decomposition has the special meaning in conv-layers. It decomposes a conv-layer with $k \times k$ spatial filters into two layers—one layer with $k \times 1$ spatial filters and the other with $1 \times k$ spatial filters. Suppose the feature map size is $h \times w$, then the number of operations is $n \times m \times k^2 \times h \times w$ for matrix A, while the number of operations reduces to $(m+n) \times c \times k \times h \times w$ for $B * C$. We have the following lemma regarding to the difference before and after binary decomposition.

Lemma 1 *(1) The computing cost ratio for A over $B * C$ is $n \times m \times k / c \times (m+n)$. (2) The bit-rate compression ratio from A to $B * C$ is also $n \times m \times k / c \times (m+n)$. (3) $c < \frac{n \times m \times k}{(m+n)}$ can yield real parameter and computing operation reduction.*

Binary Matrix Decomposition We first review the property of matrix rank:

$$rank(B * C) \leq min\{rank(B), rank(C)\}. \tag{8}$$

Comparing with binary matrix factorization methods (Zhang et al. 2007; Miettinen 2010), which tend to minimize certain kind of loss like $|A - B * C|$ and find matrices B and C iteratively, we attempt to decompose A into matrices B and C without any loss when $c \geq rank(A)$ is satisfied.

Theorem 1 *If $c \geq rank(A)$, binary matrix $A \in \{0,1\}^{(n \times k) \times (k \times m)}$ can be losslessly factorized into binary matrices $B \in \{0,1\}^{(n \times k) \times c}$ and $C \in \{0,1\}^{c \times (k \times m)}$.*

Proof According to the Gaussian elimination method, matrix A can be converted to an upper triangular matrix D. Our intuition is to construct matrices B and C through the process of Gaussian elimination. Assume $n \leq m$, P_i is the transform matrix representing the i-th primary transformation, matrix D can be expressed as:

$$D = \prod_{i=0}^{k} P_{k-i} \otimes A, \tag{9}$$

where $\otimes$ is element-wise binary multiply operator so that

$$A \otimes B = (A * B) \mod 2.$$

For simplicity, we use $*$ instead of $\otimes$ here. As $P_i \in \{0,1\}^{(n \times k) \times (n \times k)}$ is the permutation transform matrix, the inverse matrix of P_i exists. Therefore, A can be decomposed into the following form:

$$A = \prod_{i=0}^{k} P_i^{-1} * D \tag{10}$$

Since D only contains value 1 in the first r rows where $r = rank(A)$, D can be decomposed into two matrices:

$$D = \begin{bmatrix} D_1 \\ 0 \end{bmatrix} = \begin{bmatrix} I \\ 0 \end{bmatrix} * \begin{bmatrix} D_1 \end{bmatrix} \tag{11}$$

where $D_1 \in \{0,1\}^{r \times (k \times m)}$ is the first r rows of matrix D, I is a $r \times r$ identity matrix. Then matrix A can be written as:

$$A = (\prod_{i=0}^{k} P_i^{-1} * \begin{bmatrix} I \\ 0 \end{bmatrix}) * \begin{bmatrix} D_1 \end{bmatrix}. \tag{12}$$

Algorithm 1 Binary matrix decomposition

Input: binary matrix A with size $h_A \times w_A$
Output: matrix rank r, matrix B, matrix C ($A = B * C$)

function BINARYMATDECOMPOSITION(A)
  if $h_A \leq w_A$ **then**
    $A \leftarrow A^T$
    $transpose = True$
  end if
  $r \leftarrow 0$;
  $B \leftarrow$ identity matrix $h_A \times w_A$
  for $c \leftarrow 1$ to w_A **do**
    $l \leftarrow$ first raw satisfy the constraints: $A[l,c] = 1$ and $l \geq r+1$
    Reverse row l & $r+1$, $P \leftarrow$ corresponding transition matrix
    $r \leftarrow r+1$;
    $B \leftarrow B * P^{-1}$
    for $row \leftarrow r+1$ to h_A **do**
      if $A[row, c] > 0$ **then**
        $A[row, :] \leftarrow (A[r, :] + A[row, :])$ mod 2
        $P \leftarrow$ corresponding transition matrix
        $B \leftarrow B * P^{-1}$
      end if
    end for
  end for
  $C \leftarrow$ first r rows of A
  $P \leftarrow$ first r rows are identity matrix, other $h_A - r$ rows are zeros.
  $B \leftarrow B * P$
  if $transpose$ **then**
    Return r, C^T, B^T
  else
    Return r, B, C
  end if
end function

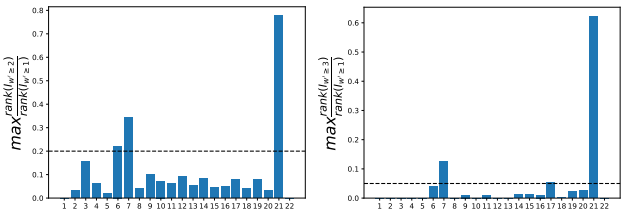
We then obtain the size $(n \times k) \times r$ matrix B as $B = \prod_{i=0}^{k} P_i^{-1} * \begin{bmatrix} I \\ 0 \end{bmatrix}$, and the size $r \times (k \times m)$ matrix C as $C = D_1$ exactly without any loss. This procedure also indicates that the minimum bottleneck parameter c is $rank(A)$, i.e., $c \geq rank(A)$. This ends the proof. □

Based on this proof, we outline the binary matrix decomposition procedure in Algorithm 1. From the proof, we should also point out that *the proposed binary decomposition is suitable for both conv-layers and FC-layers*. Note that for binary permutation matrix P, its inverse matrix P^{-1} equals to itself, i.e., $P^{-1} = P$. Suppose h_A and w_A are height and width of matrix A, the computing complexity of $B * P$ is just $O(h_A)$ as P is a permutation matrix. The computing complexity of Algorithm 1 is $O(w_A \times h_A^2)$.

Losslessly Compressible of A_i? Theorem 1 shows that only when $c \geq rank(A)$, our method could produce lossless binary decomposition, while Lemma 1 shows that only $c < \frac{n \times m \times k}{(m+n)}$ could yield practical parameter and computing operations reduction. We have the following corollary:

Corollary 1 *Binary matrix A is **losslessly compressible** based on Theorem 1 when $rank(A) \leq c < \frac{n \times m \times k}{(m+n)}$.*

However, it is unknown which A_i in Eq.(3) was *losslessly compressible* before decomposition. The brute-force way is trying to decompose each A_i as in Eq.(7), and then keeping those satisfying Definition 1. This is obviously inefficient and impracticable. Alternatively, we may use some



(a) (b)

Figure 4: Maximum rank ratio per-layer of ResNet-18.

heuristic cue to estimate which subset of $\{A_i\}$ could be *losslessly compressible*. According to Figure 3, A_i with smaller i is more sparse than that with bigger i. Empirically, more sparsity corresponds to smaller $rank(A_i)$. Based on Theorem 1, this pushes us to seek the watershed value j so that those A_i where $i \leq j$ are *losslessly compressible*, while other A_i ($i > j$) are not. This requires introducing a variable into the definition of A_i, so that we choose the A_i defined by Eq.(6) rather than Eq.(3). As is known, the A_i sequence defined by Eq.(6) is $\{A_{-q}, \cdots, A_0, \cdots, A_{J-q-2}\}$ where $q = \lceil \log_2 \alpha \rceil$. Here, we seek for the optimal α, so that $j = 0$ is the watershed, i.e., $\{A_{-q}, \cdots, A_0\}$ are *losslessly compressible*.

For simplicity, we still denote the flatten matrix of the tensor $\hat{\mathbf{W}}_t^{\alpha}$ in Eq.(6) as $\mathbf{W} \in \mathbb{R}^{(n\times k)\times(k\times m)}$. We propose to use the indicator matrix described below for easy analysis.

Definition 1 *The **indicator matrix** of* $\mathbf{W}$ *is defined as* $I_{w>\beta} \in \{0,1\}^{(n\times k)\times(k\times m)}$*, in which the value at position* (x, y) *is* $I_{w>\beta}[x, y] = I_f(\mathbf{W}[x, y] > \beta)$*, where* $\beta \in [0, \alpha]$ *is a parameter,* $I_f(\cdot)$ *is an element-wise indication function, which equals to 1 when the prediction is true, otherwise 0.*

Based on this definition, A_i in Eq.(6) can be written as

$$A_i[x, y] = I_f(\lfloor \frac{\mathbf{W}[x, y] + \Delta w}{2^{-i}} \rfloor \mod 2 = 1), \quad (13)$$

where $\Delta w = 2^{-J+q+1}$ is the largest throwing-away power-of-two terms in Eq.(6).

Define $w' = w + \Delta w$, according to Eq.(6), the rank of matrix A_0 can be expressed as:

$$\begin{aligned} rank(A_0) &= rank(\sum_{i=0}^{\lceil n_I/2 \rceil - 1} I_{(2i+1)\leq w' < (2i+2)}) \\ &= rank(\sum_{i=1}^{n_I} I_{w'\geq i}), \end{aligned}$$

where $n_I = \lceil \alpha \rceil$. Based on the matrix rank property

$$rank(A + B) \leq rank(A) + rank(B), \quad (14)$$

we derive the upper bound of the $rank(A_i)$ as:

$$rank(A_0) \leq \sum_{i=1}^{n_I} rank(I_{w'\geq i}). \quad (15)$$

The empirical results show that when $rank(I_{w'\geq 1}) \leq 0.5 * min\{n \times k, m \times k\}$, and the rank of the indicator matrix satisfies the following constraints:

$$\begin{aligned} &(1)\ \frac{rank(I_{w'\geq 2})}{rank(I_{w'\geq 1})} \leq C_0, \\ &(2)\ max\{\frac{rank(I_{w'\geq i})}{rank(I_{w'\geq 1})}\} \leq C_1, 3 \leq i \leq n_I. \end{aligned} \quad (16)$$

Algorithm 2 Binary search α to satisfy rank condition.

Input: weight matrix $\mathbf{W}$, expected rank c
Output: scalar value α

function SCALARVALUESEARCH(W, c)
  $min \leftarrow 0$, $max \leftarrow$ max number of 1 value in a full rank matrix
  sort $\mathbf{W}$ in a descending order to a vector $\mathbf{v}$
  while $min \leq max$ **do**
    $center \leftarrow (min + max)/2$
    $\alpha \leftarrow 1/\mathbf{v}[center]$
    Compute indicator matrix $I_{w\geq 1}$
    Compute rank r of $I_{w\geq 1}$
    if $r > c$ **then**
      $max \leftarrow center - 1$
    else if $r < c$ **then**
      $min \leftarrow center + 1$
    else
      Return α
    end if
  end while
  Return $\alpha = 1/\mathbf{v}[max]$
end function

With the bound (15), we get the bound of the rank by the indicator matrix $I_{w'\geq 1}$:

$$rank(A_0) \leq ((\alpha - 2)_+ * C_1 + C_0 + 1) * rank(I_{w'\geq 1}), \quad (17)$$

where $(x)_+$ equals to x if $x > 0$, otherwise 0. We make some empirical study on ResNet-18, and find that $C_0 \leq 0.2$ in most of the layers as in Figure 4a, $C_1 \leq 0.05$ in most of the layers as in Figure 4b, and $\alpha \leq 8$ in all the layers. Hence, we have a simpler method to estimate $rank(A_0)$ by:

$$\begin{aligned} rank(A_0) &\leq (4 * 0.05 + 0.2 + 1) * rank(I_{w'\geq 1}) \\ &= 1.4 * rank(I_{w'\geq 1}). \end{aligned}$$

Therefore, we transfer the optimization of α to the optimization of $rank(I_{w'\geq 1})$.

A binary search algorithm for scaling factor a

In practice, we may give an expected upper bound c for $rank(I_{w\geq 1})$, and search the optimal α to satisfy

$$rank(I_{w\geq 1}) \leq c. \quad (18)$$

As $w \in [0, \alpha]$ and $\alpha > 1$, $w \in [0, 1]$ only corresponds to $1/\alpha$ portion of the whole range of w.

Generally, weight matrix $\mathbf{W}$ should be sorted and traversed to compute the $rank(I_{w\geq 1})$. Instead of using this time-consuming method, we propose an efficient solution based on the binary search algorithm.

Suppose that every element in weight matrix $\mathbf{W}$ has a unique value, we sort $\mathbf{W}$ to be a vector $\mathbf{v}$ with length $N = n \times m \times k^2$ in descending order, so that $\mathbf{v}[1]$ is the largest element. Assume index i satisfy the constraint: $\mathbf{v}[i] \geq 1 > \mathbf{v}[i + 1]$, then the indicator matrix can be expressed as:

$$I_{w\geq 1} = I_{p(w)<i+1} \quad (19)$$

where $p(w)$ is the index position of the element w in array $\mathbf{v}$, i.e., $\mathbf{v}[p(w)] = w$. Hence, there are i ones in the indicator matrix while others are zeros. Comparing matrix $I_{p(w)<i+1}$ with matrix $I_{p(w)<i}$, we have the property:

$$I_{p(w)<i+1}[x, y] = \begin{cases} I_{p(w)<i}[x, y] & p(w[x, y]) \neq i \\ I_{p(w)<i}[x, y] + 1 & p(w[x, y]) = i \end{cases} \quad (20)$$

Model	FP32			CBDNet			Bitrate
	top-1(%)	top-5(%)	size-MB	top-1(%)	top-5(%)	size-MB	
ResNet-18	66.41	87.37	44.57	65.27(-1.14)	86.62(-0.75)	7.31	5.25
VGG-16	68.36	88.44	527.7	67.36(-1.00)	87.81(-0.63)	90.3	5.47
DenseNet-121	74.41	92.14	30.11	73.13(-1.28)	91.29(-0.85)	5.38	5.72

Table 1: Performance of different CNNs on ImageNet and CBDNet. Numbers in bracket indicates accuracy drop relative to FP32 models. Last-column lists bitrate of CBDNet.

With the property of Eq.(14), we get the relation between the indicator matrices with adjacent index:

$$|rank(I_{p(w)<i+1}) - rank(I_{p(w)<i})| \leq 1. \tag{21}$$

This indicates that $rank(I_{p(w)<i})$ are continuous integers. Hence, $i \in [i_1, i_2]$ always exists to satisfy below constraints:

$$\begin{aligned} &rank(I_{p(w)<i}) = c, \\ &(rank(I_{p(w)<i_1}) - c) * (rank(I_{p(w)<i_2}) - c) < 0. \end{aligned} \tag{22}$$

Eq.(22) provides the property to design a binary search algorithm as outlined in Algorithm 2, which could find the local optimal of α when matrix $\mathbf{W}$ has some identical elements. The computing complexity of this algorithm is $O(N \log(N))$, where $N = n \times k^2 \times m$.

Here, c is a tunable hyper-parameter. In practice, we did not directly give c since different layers may require different c. Instead, we assume $m \geq n$ and define $b = {}^{c}/_{n \times k}$ as bottleneck ratio, since we reduce the neuron number from $n \times k$ to c by the binary matrix B. Then we tune b and make it constant over all the layers, while $b < 0.5$ could provide compression effect. After getting α, we obtain $q = \lceil \log_2 \alpha \rceil$, and only do binary decomposition for the subset $\{A_{-q}, \cdots, A_0\}$ as they are *losslessly compressible*.

Experiments

This section conducts experiments to verify the effectiveness of CBDNet on various ImageNet classification networks (Russakovsky et al. 2015), as well as object detection network SSD300 (Liu et al. 2016) and semantic segmentation network SegNet (Badrinarayanan et al. 2017).

Classification Networks on ImageNet

The ImageNet dataset contains 1.2 million training images, 100k test images, and 50k validation images. Each image is classified into one of 1000 object categories. The images are cropped to 224×224 before fed into the networks. We use the validation set for evaluation and report the classification performance via top-1 and top-5 accuracies. Table 1 gives an overall performance of different evaluated networks. Note we decomposed all the conv-layers and FC-layers with the proposed method in this study. The resulted bit-rate is defined by $32 \times size(\text{CBDNet})/size(\text{FP32})$, where $size(x)$ gives the model size of model x.

Model 1: ResNet-18 (He et al. 2016) consists of 21 conv-layers and 1 FC-layer. The top-1/top-5 accuracy of the FP32 model is $66.41/87.37$ in our single center crop evaluation. Figure 5 shows the accuracy and the bit-rate of CBDNet at different bottleneck ratios. The effective bit-rate of CBDNet is 5.25 with b=0.3 and J=7, while the top-1/top-5 accuracy drops by 1.14% and 0.75% respectively.

Model	FP32	BWD		CBDNet	
	top-5(%)	top-5(%)	Bitrate	top-5(%)	Bitrate
ResNet-152	92.11	90.25(-1.86)	6	91.61(-0.5)	5.37
VGG-16	88.44	86.28(-2.16)	6	87.81(-0.63)	5.47

Table 2: CBDNet vs BWD on ResNet and VGGNet.

Model 2: VGG-16 (Simonyan and Zisserman 2015) consists of 13 conv-layers and 3 FC-layers. The top-1/top-5 accuracy of the FP32 model is $68.36/88.44$ in our single center crop evaluation. Figure 6 shows the accuracy and the bit-rate of CBDNet at different bottleneck ratios. The effective bit-rate of CBDNet is 5.47 with b=0.3 and J=7, while the top-1/top-5 accuracy drops by 1% and 0.63% respectively.

Model 3: DenseNet-121 (Huang et al. 2017) consists of 121 layers. The top-1/top-5 accuracy of the FP32 model is $74.41/92.14$ in our single center crop evaluation. Figure 7 shows the accuracy and the bit-rate of CBDNet at different bottleneck ratios. The effective bit-rate is 5.72 with $b = 0.4$ and $J = 7$, while the top-1/top-5 accuracy drops by 1.28% and 0.85% respectively.

Comparison to State-of-the-art We further compare the accuracy and bit-rate to the prior art binary weighted decomposition (BWD) (Kamiya et al. 2017). Table 2 lists the comparison results on ResNet-152 and VGG-16. Note that, BWD keeps the first conv-layer un-decomposed to ensure no significant accuracy drops, while our CBDNet decomposes all the layers. It shows that our CBDNet achieves much less accuracy drops even with smaller bit-rate.

Detection Networks

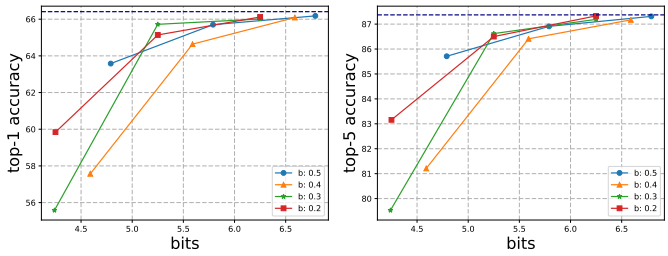
We apply CBDNet to the object detection network SSD300 (Liu et al. 2016). The evaluation is performed on the VOC0712 dataset. SSD300 is trained with the combined training set from VOC 2007 `trainval` and VOC 2012 `trainval` ("07+12"), and tested on the VOC 2007 testset. We compare the performance between original SSD300 and our CBDNet. Figure 8a shows the comparison results, in which we also include the composite-only (without binary decomposition) results.

The effective bit-rate of CBDNet is 4.38, with 1.34% drop of the mean Average Precision (mAP). That verifies the effectiveness of our CBDNet on object detection networks.

Semantic Segmentation Networks

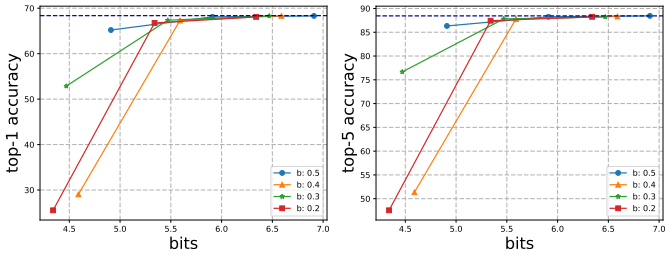
We also perform an evaluation on the semantic segmentation network SegNet (Badrinarayanan et al. 2017). The experiment is conducted on the Cityscapes dataset (Cordts et al. 2016) of the 11 class version, for fair comparison to results by (Kamiya et al. 2017). We use the public available SegNet model, and test on the Cityscapes validation set. Figure 8b shows the comparison results, in which the composite-only results are also included.

The effective bit-rate of CBDNet on SegNet is 5.18, with only 0.8% drop of the mean of intersection over union (mIOU). In comparison, under the same setting, the BWD (Kamiya et al. 2017) requires 6 bits but yields more than 2.75% accuracy loss. Figure 9 further illustrates the segmentation results on two tested images, which compares results



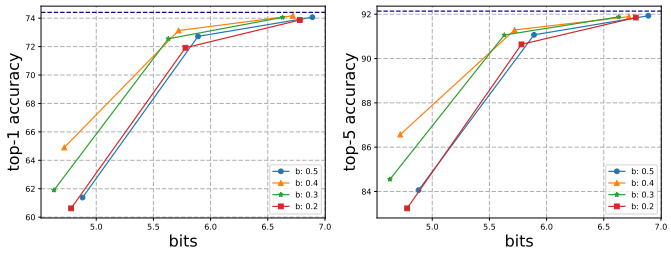
(a) (b)

Figure 5: CBDNet accuracy on ImageNet for ResNet-18. Dashed-line indicates FP32 accuracy. 'b' for bottleneck ratio.



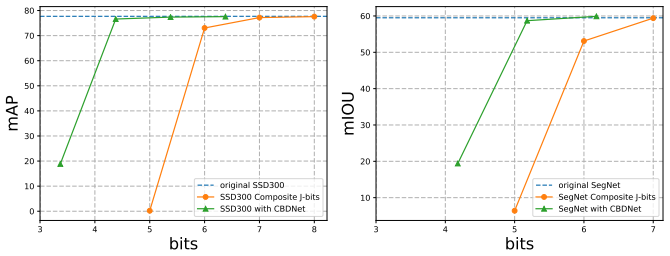
(a) (b)

Figure 6: CBDNet accuracy on ImageNet for VGG-16. Dashed-line indicates FP32 accuracy. 'b' for bottleneck ratio.



(a) (b)

Figure 7: CBDNet accuracy on ImageNet for DenseNet-121. Dashed-line indicates FP32 accuracy. 'b' for bottleneck ratio.



(a) (b)

Figure 8: Accuracy comparison on (a) SSD300 and (b) SegNet.

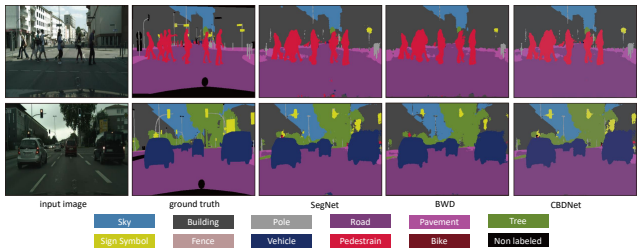


Figure 9: Semantic segmentation results on CityScapes. CBDNet is better than BWD on "sign" and "pole".

from ground truth, SegNet, BWD, and our CBDNet. It is obvious that our CBDNet gives better results than BWD. That verifies the effectiveness of our CBDNet on semantic segmentation networks.

Inference Speed with CBDNet

We make the inference speed comparison between CBDNet and the FP32 version on Intel Core i7-6700 CPU with 32GB RAM. We optimize one conv-layer of VGG-16 using SSE4.2 instructions (128-bit SIMD) for both the FP32 version and our CBDNet version. For CBDNet, we quantize the input into 8-bits using the TensorFlow quantizer (Krishnamoort 2018), and then perform bitwise operations. We project results in this layer to the whole network based on operation numbers per-layer, and show that our CBDNet is about 4.02× faster than the FP32 counterpart. In comparison, BWD only gives 2.07× speedup over the FP32 version with SSE4.2 optimization, under 6-bits data/weight quantization. This clearly demonstrates our advantage over BWD (Kamiya et al. 2017). We believe specific designed hardware with dedicated bitwise accelerators could realize ultra efficient inference for our CBDNet.

Conclusion

This paper proposes composite binary decomposition networks (CBDNet), which can directly transfer pre-trained full-precision CNN models into multi-bits binary network models in a training-free way, while remain model accuracy and computing efficiency. The method contains two steps, composite real-valued tensors into a limited number of binary tensors, and decomposing certain conditioned binary tensors into two low-rank tensors for parameter and computing efficiency. Experiments demonstrate the effectiveness of the proposed method on various classification networks like ResNet, DenseNet, VGGNet, as well as object detection network SSD300, and semantic segmentation network SegNet.

Acknowledgement: Y. Qiaoben and J. Zhu were supported by the National Key Research and Development Program of China (2017YFA0700904), NSFC projects (61620106010, 61621136008, 61332007) and the MIIT Grant of Int. Man. Comp. Stan (2016ZXFB00001). Yu-Gang Jiang was supported partly by NSFC projects (61622204, U1509206).

References

Badrinarayanan, V.; Kendall, A.; Cipolla, R.; et al. 2017. SegNet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Trans PAMI* 39.

Chen, W.; Wilson, J.; S, T.; et al. 2015. Compressing neural networks with the hashing trick. In *ICML*.

Cordts, M.; Omran, M.; Ramos, S.; et al. 2016. The cityscapes dataset for semantic urban scene understanding. In *CVPR*.

Courbariaux, M.; Bengio, Y.; David, J.-P.; et al. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*.

Courbariaux, M.; Bengio, Y.; David, J.-P.; et al. 2016. Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1. In *ICLR*.

Denton, E.; Zaremba; Lecun, Y.; et al. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*.

Dong, Y.; Ni, R.; Li, J.; et al. 2017. Learning accurate low-bit deep neural networks with stochastic quantization. In *BMVC*.

Girshick, R.; Donahue, J.; Darrell, T.; et al. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*.

Guo, J.; Li, Y.; Lin, W.; et al. 2018. Network decoupling: From regular to depthwise separable convolutions. In *BMVC*.

Han, S.; Pool, J.; Tran, J.; Dally, W.; et al. 2015. Learning both weights and connections for efficient neural network. In *NIPS*.

Han, S.; Mao, H.; Dally, W. J.; et al. 2016. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*.

Hinton, G.; Vinyals, O.; Dean, J.; et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hou, L.; Yao, Q.; Kwok, J. T.; et al. 2017. Loss-aware binarization of deep networks. In *ICLR*.

Howard, A. G.; Zhu, M.; Chen, B.; et al. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Huang, G.; Liu, Z.; Weinberger, K. Q.; et al. 2017. Densely connected convolutional networks. In *CVPR*.

Jaderberg, M.; Vedaldi, A.; Zisserman, A.; et al. 2014. Speeding up convolutional neural networks with low rank expansions. In *BMVC*.

Kamiya, R.; Yamashita, T.; Ambai, M.; et al. 2017. Binary-decomposed dcnn for accelerating computation and compressing model without retraining. In *ICCV Workshop*.

Krishnamoort, R. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1805.07941*.

Krizhevsky, A.; Sutskever, I.; Hinton, G. E.; et al. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*.

Li, H.; Kadav, A.; I, D.; et al. 2017. Pruning filters for efficient convnets. In *ICLR*.

Lin, X.; Zhao, C.; Pan, W.; et al. 2017. Towards accurate binary convolutional neural network. In *NIPS*.

Liu, W.; Anguelov, D.; Erhan, D.; et al. 2016. SSD: Single shot multibox detector. In *ECCV*.

Liu, C.; Zoph, B.; Neumann, M.; et al. 2017a. Progressive neural architecture search. *arXiv preprint arXiv:1712.00559*.

Liu, Z.; Li, J.; Shen, Z.; et al. 2017b. Learning efficient convolutional networks through network slimming. In *ICCV*.

Long, J.; Shelhamer, E.; Darrell, T.; et al. 2015. Fully convolutional networks for semantic segmentation. In *CVPR*.

Miettinen, P. 2010. Sparse boolean matrix factorizations. In *ICDM*.

Migacz, S. 2017. 8-bit inference with TensorRT. Technical report, NVidia.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. XNOR-Net: Imagenet classification using binary convolutional neural networks. In *ECCV*.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; et al. 2015. ImageNet Large Scale Visual Recognition Challenge. *IJCV* 115(3).

Shen, Z.; Liu, Z.; Li, J.; et al. 2017. DSOD: Learning deeply supervised object detectors from scratch. In *ICCV*.

Simonyan, K., and Zisserman, A. 2015. Very deep convolutional networks for large-scale image recognition. In *ICLR*.

Szegedy, C.; Liu, W.; Jia, Y.; et al. 2015. Going deeper with convolutions. In *CVPR*.

Wen, W.; Wu, C.; Wang, Y.; et al. 2016. Learning structured sparsity in deep neural networks. In *NIPS*.

Zhang, Z.; Li, T.; Ding, C.; and Zhang, X. 2007. Binary matrix factorization with applications. In *ICDM*.

Zhang, X.; Zou, J.; Sun, J.; et al. 2016. Accelerating very deep convolutional networks for classification and detection. *IEEE Trans PAMI* 38(10).

Zhang, T.; Qi, G.; Wang, J.; et al. 2017. Interleaved group convolutions. In *ICCV*.

Zhang, X.; Zhou, X.; Sun, J.; et al. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*.

Zhou, S.; Wu, Y.; Ni, Z.; et al. 2016. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*.

Zoph, B., and Le, Q. V. 2016. Neural architecture search with reinforcement learning. In *ICLR*.