The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

From Hashing to CNNs: Training Binary Weight Networks via Hashing

Qinghao Hu, Peisong Wang, Jian Cheng
Institute of Automation, Chinese Academy of Sciences, Beijing, China
University of Chinese Academy of Sciences, Beijing, China
Center for Excellence in Brain Science and Intelligence Technology, CAS, Beijing, China
{qinghao.hu, peisong.wang, jcheng}@nlpr.ia.ac.cn

Abstract

Deep convolutional neural networks (CNNs) have shown appealing performance on various computer vision tasks in recent years. This motivates people to deploy CNNs to real-world applications. However, most of state-of-art CNNs require large memory and computational resources, which hinders the deployment on mobile devices. Recent studies show that low-bit weight representation can reduce much storage and memory demand, and also can achieve efficient network inference. To achieve this goal, we propose a novel approach named BWNH to train Binary Weight Networks via Hashing. In this paper, we first reveal the strong connection between inner-product preserving hashing and binary weight networks, and show that training binary weight networks can be intrinsically regarded as a hashing problem. Based on this perspective, we propose an alternating optimization method to learn the hash codes instead of directly learning binary weights. Extensive experiments on CIFAR10, CIFAR100 and ImageNet demonstrate that our proposed BWNH outperforms current state-of-art by a large margin.

Introduction

Since Alexnet (Krizhevsky, Sutskever, and Hinton 2012) made a success in ILSVRC2012 (Russakovsky et al. 2015), deep convolutional neural networks have become more and more popular. After that, various CNN models have been proposed such as VGGNet (Simonyan and Zisserman 2014), Inception (Szegedy et al. 2016), ResNet (He et al. 2016) and so on. Nowadays, these CNN models have been playing an important role in many computer vision areas (Krizhevsky, Sutskever, and Hinton 2012; Ren et al. 2015; Long, Shelhamer, and Darrell 2015).

Attracted by the great performance of CNN models, many people try to deploy CNNs to real world applications. Yet the huge computational complexity and large parameter size make CNN models hard to deploy on resource limited devices such as mobile phones and embedded devices. The huge computational complexity of CNN models makes the inference phase very slow, which is unacceptable for many real-time applications. The large parameter size brings three difficulties. First, the large parameter size means that deploying CNN models will consume huge disk storage. Second, much run-time memory is required, which is limited in many mobile devices. Third, large parameter size will cause heavy DRAM access, which consumes more energy. Since battery power is very limited in many mobile devices, this severely affects devices' battery life.

To alleviate these problems, a variety of methods have been proposed to reduce the parameter size or accelerate the inference phase. These methods can be divided into three main categories: low-rank decomposition based methods, pruning-based methods, and quantization based methods. Low-rank decomposition based methods (Denton et al. 2014; Jaderberg, Vedaldi, and Zisserman 2014; Zhang et al. 2015; Wang and Cheng 2016) decompose a weight matrix (tensor) into several small weight matrices (tensors). These methods achieve good speed-ups for large convolutional kernels, but usually perform poorly for small kernels. Besides, the compression ratio of parameters is kind of low by using low-rank based methods. Network pruning has a long history and is still a widely used technique for CNN compression and acceleration (Han et al. 2015; Liu et al. 2015). The main idea of these methods is to remove low-saliency parameters or small-weight connections (Han et al. 2015). In general, after the pruning step, k-means clustering and Huffman coding are also required to make a good compression ratio. The k-means clustering and Huffman coding bring inconvenience for inference phase since we have to decode the Huffman codes and use lookup table for k-means dictionary. As a result, the decoding and lookup table will bring extra memory and computational overhead. Quantization based methods include codebook based quantization methods and low-bit weight representation. Codebook based quantization methods mainly use vector quantization algorithms such as K-means, Product Quantization and so on (Gong et al. 2014; Wu et al. 2016) to quantize the weight kernels. These methods require lookup tables to store the dictionary, and is unfriendly to cache memory since accessing lookup tables is random and unordered. Low-bit weight representation methods (Lin, Talathi, and Annapureddy 2015; Gupta et al. 2015; Rastegari et al. 2016; Dong et al. 2017) represent weights as low-bit fixed point even binary values. Low-bit weight representation can reduce run-time memory and storage demand as no decoding or lookup tables are required. As a special case of low-bit weight representation, binary weight can achieve about 32$\times$ compression ratio. In addition, since weights are represented by binary values, multiplication operations can be replaced by addition and subtraction operations. Thus binary weight can also speed up the inference phase. Nevertheless,current weight binarization methods usually will bring significant network accuracy drop, especially for large CNN models.

In this paper, we propose a novel approach named BWNH to train binary weight networks via hashing. We first transform the binary weight learning problem into a hashing problem. Then an alternating optimization algorithm is proposed to solve the hashing problem. Finally, the whole binary weight network is fine-tuned to restore accuracy. Extensive experiments on three datasets demonstrate that our algorithm outperforms state-of-art algorithms. Our main contributions are:

(1) We uncovered the close connection between inner-product preserving hashing and binary weight neural networks. Based on this view, training binary weight networks can be transformed into a hashing problem. To the best of our knowledge, it is the first to train binary weight CNNs via hashing.

(2) To alleviate the loss brought by hashing, the binary codes is multiplied by a scaling factor. To solve the binary codes and scaling factor, we propose an alternating optimization method to iteratively update binary codes and scaling factor.

(3) We conduct extensive experiments on CIFAR10, CIFAR100, and ImageNet. And the experimental results demonstrate that our proposed BWNH outperforms the state-of-art algorithms. Specifically, on ResNet-18, our BWNH achieves 3.0% higher accuracy than the best reported binary weight networks for ImageNet classification task.

Related Work

While deep convolutional neural networks have achieved quite good performance in many computer vision tasks, the large computational complexity and parameter size have hindered the deployment on mobile devices. A variety of methods have been proposed to alleviate these problems.

Pruning Optimal Brain Demage (LeCun et al. 1989) and Optimal Brain Surgeon (Hassibi and Stork 1993) are early works of pruning. Both these two algorithms used Hessian matrix of loss functions, which makes them hard to scale up to large scale models. (Han et al. 2015) proposed the deep compression framework and they introduced a three-stage pipline: pruning, trained quantization and Huffman coding. They demonstrated that such a three-stage method can reduce the parameter size of AlexNet up to 35 times. After pruning, the sparse connections of CNNs do not fit well on parallel computation. To cure this problem, a group-sparsity regularizer was proposed (Lebedev and Lempitsky 2015). By using the group-sparsity regularizer, they pruned the convolutional kernel tensor in a group-wise fashion. After such pruning, convolutions can be reduced to multiplications of thinned dense matrices, which can use the Basic Linear Algebra Subprograms (BLAS) to get higher speed-ups.

Low-rank Approximation Low-rank based methods assume that the featuremaps or weights of CNNs lie on a low-rank subspace. Based on this assumption, matrix or tensor decomposition methods are applied to the convolutional kernels or featuremaps (Denton et al. 2014; Jaderberg, Vedaldi, and Zisserman 2014; Denil et al. 2013). By using biclustering and Singule Value Decomposition (SVD), (Denton et al. 2014) achieved $1.6\times$ speed-up of single layer. (Zhang et al. 2015) took nonlinear layers into consideration and used an asymmetric reconstruction method to approximate the low rank matrix. (Lebedev et al. 2014) utilized CP-decomposition to approximate the 4D convolutional kernel tensor. For large models such as Alexnet, their methods only processed a single layer and could not work well for the whole network. (Kim et al. 2015) proposed to use Tucker decomposition to reduce the computational complexity of CNN models. By making a compromise between CP-decomposition and Tucker decomposition, Block Term Decomposition was used to accelerate the CNN models (Wang and Cheng 2016). (Novikov et al. 2015) used the Tensor-Train format to decompose the fully connected layer, which achieved up to $7\times$ compression ratio of the whole network.

Quantization-based Methods As mentioned above, quantization-based methods can be divided into two groups: codebook-based quantization and low-bit quantization. (Gong et al. 2014; Wu et al. 2016) are typical ones of codebook-based quantization methods. (Gong et al. 2014) used vector quantization to compress the fully-connected layers of CNNs. And (Wu et al. 2016) proposed an product quantization based algorithm to speed up and compress CNNs in the meantime. For low-bit weight quantization, early works focused on using fixed-point data format to represent the weights of CNNs. (Gupta et al. 2015) introduced a stochastic rounding scheme to quantize the weights to fixed-point format. They showed that neural networks can be trained using only 16-bit fixed-point format with little degradation of classification accuracy. Later, a dynamic-precision data quantization method was proposed by (Qiu et al. 2016), and 8/4-bit quantization was achieved with little loss. These methods assume that all layers share the same bit-width. (Lin, Talathi, and Annapureddy 2015) showed that it's better that different layers have different bit-width. Binary weight is a special case of low-bit quantization where weights are quantized into binary values. (Courbariaux, Bengio, and David 2015) proposed BinaryConnect to train CNNs with binary weights, and their method demonstrated well performance on small dataset such as MNIST, CIFAR10, and SVHN. Later, (Lin et al. 2015) proposed ternary connect to quantize the weights to ternary values, and they also quantized the back propagation. Experiments demonstrated that ternary connect achieved better result than binary connect. (Zhou et al. 2017) proposed an incremental network quantization method which consists of three operations: weight partition, group-wise quantization and re-training. These three operations are repeated on an iterative manner, and experiments on ImageNet demonstrated that CNN models can be quantized into 5 bits without accuracy drop.

(Rastegari et al. 2016) proposed Binary-Weight-Networks whose weights are binarized and multiplied with a scaling factor, and they also proposed XNOR-Net by binarizing both activations and weights. (Cai et al. 2017) proposed an Halfwave Gaussian quantizer (HWGQ) for forward approximation. (Dong et al. 2017) introduced a stochastic quantization scheme which quantizing weights with a stochastic probability inversely proportional to the quantization error.

Hashing and Neural Networks (Chen et al. 2015) first introduced hashing methods to compress CNNs, they used the hashing trick to map the high dimensional features to a low-dimensional dictionary. The weights in the dictionary are still float-numbers, which has a large difference with our method. (Spring and Shrivastava 2017) proposed a scalable and sustainable deep learning framework via randomized hashing. By using hash codes lookup table, they collected a small portion of neural nodes called active set. Since only these neural nodes required forward and backward propagation, computational cost was reduced. Our method is different with (Spring and Shrivastava 2017) in several ways. First, the hashing method is used for different goals. Our algorithm aims to learn binary weights while (Spring and Shrivastava 2017) use hashing algorithm to select the active set. Second, our method is a learning-based (data-dependent) method while (Spring and Shrivastava 2017) used a data-independent Locality-Sensitive Hashing (LSH) method. Third, our method is different with (Spring and Shrivastava 2017) in the inference phase. The multiplication operation is replaced with add operation in our method while (Spring and Shrivastava 2017) choose a subset of neural nodes to do forward propagation.

Our Method

In this section, we first introduce the inner-product preserving hashing, and uncover the close connection between inner-product preserving hashing and learning binary convolutional kernels. Then we give details about how the objective is transformed from learning binary weights to learning hashing codes. A new objective function is proposed to compensate the accuracy loss brought by hashing codes, then an alternating optimization method is introduced to solve the new objective function. Finally, we present our whole training scheme.

Inner-product preserving hashing

Given two sets of points $\mathbf{X} \in \mathbb{R}^{S\times M}$ and $\mathbf{W} \in \mathbb{R}^{S\times N}$ where $\mathbf{X_i} \in \mathbb{R}^{S\times 1}$ and $\mathbf{W_i} \in \mathbb{R}^{S\times 1}$ represents i^{th} point of $\mathbf{X}$ and $\mathbf{W}$ respectively, we denote the inner-product similarity of $\mathbf{X}$ and $\mathbf{W}$ as $\mathbf{S} \in \mathbb{R}^{M\times N}$. (Shen et al. 2015a) proposed the inner-product preserving hashing by solving the following objective function:

$$\min \|\mathbf{S} - h(\mathbf{X})^{\mathrm{T}} g(\mathbf{W})\|_F^2 \tag{1}$$

where $h(\cdot)$ and $g(\cdot)$ are hash functions for $\mathbf{X}$ and $\mathbf{W}$ respectively.

Connection between hashing and binary weights

Suppose we have an L-layer pre-trained CNN model e.g. Alexnet, and $\mathbf{X} \in \mathbb{R}^{S\times M}$ is the input featuremap for l^{th} layer of the CNN model. We denote the real-value weights of l^{th} layer as $\mathbf{W} \in \mathbb{R}^{S\times N}$, and our goal is to get binary weight $\mathbf{B} \in \{-1,+1\}^{S\times N}$ for l^{th} layer of CNN model. A naive method is to optimize the following objective function:

$$\begin{aligned} \min\ & L(\mathbf{B}) = \|\mathbf{W} - \mathbf{B}\|_F^2 \\ s.t.\quad & \mathbf{B} \in \{+1,-1\}^{S\times N} \end{aligned} \tag{2}$$

where the solution is $\mathbf{B} = sign(\mathbf{W})$. Directly binarizing $\mathbf{W}$ would cause severe accuracy drops, another choice is to minimize the quantization error of inner-product similarity:

$$\begin{aligned} \min\ & L(\mathbf{B}) = \|\mathbf{X}^{\mathrm{T}}\mathbf{W} - \mathbf{X}^{\mathrm{T}}\mathbf{B}\|_F^2 \\ s.t.\quad & \mathbf{B} \in \{+1,-1\}^{S\times N} \end{aligned} \tag{3}$$

Note Equation (3) has a close connection with Equation (1), let $\mathbf{S} = \mathbf{X}^{\mathrm{T}}\mathbf{W}$, $\mathbf{B} = g(\mathbf{W})$ and $h(\mathbf{X}) = \mathbf{X}$, then Equation (3) is equal to Equation (1). In other words, training binary weight networks can be intrinsically transformed into a hashing problem. We notice that $h(\cdot)$ is an identity function, which means that we don't learn the hash codes for $\mathbf{X}$. This is commonly used for asymmetric distances calculation (ADC) in the hashing area. Now we have connected binary weight networks with hashing together, thus we can solve binary weight $\mathbf{B}$ by borrowing methods from hashing.

However, solving Eq. (3) still can cause somewhat accuracy drops. Inspired by (Rastegari et al. 2016), we multiply a scaling factor to each hashing codes $\mathbf{B_i}$:

$$g(\mathbf{W}) = \mathbf{BA} \tag{4}$$

where $\mathbf{A}$ is a diagonal matrix and $\alpha_i = \mathbf{A_{ii}}$ is the scaling factor for $\mathbf{B_i}$. Finally, our objective function is:

$$\begin{aligned} \min\ L(\mathbf{A},\mathbf{B}) &= \|\mathbf{S} - \mathbf{X}^{\mathrm{T}}\mathbf{BA}\|_F^2 \\ &= \sum_i^N \|\mathbf{S_i} - \alpha_i \cdot \mathbf{X}^{\mathrm{T}}\mathbf{B_i}\|_F^2 \end{aligned} \tag{5}$$

where $\mathbf{S} = \mathbf{X}^{\mathrm{T}}\mathbf{W}$ and $\mathbf{S_i} \in \mathbb{R}^{M\times 1}$ is i^{th} column vector of $\mathbf{S}$. The Eq. (5) can be easily divided into N independent sub-problems:

$$\begin{aligned} \min\ & L_i(a_i, \mathbf{B_i}) = \|\mathbf{S_i} - \alpha_i \cdot \mathbf{X}^{\mathrm{T}}\mathbf{B_i}\|_F^2 \\ s.t.\quad & \mathbf{B_i} \in \{+1,-1\}^{S\times 1} \end{aligned} \tag{6}$$

Here we propose to use an alternating optimization method to solve Eq.(6), i.e. update binary codes $\mathbf{B_i}$ with scaling factor α_i fixed, and vice versa.

Initialization of $\mathbf{B_i}$ and α_i At the beginning of alternating optimization method, we initialize $\mathbf{B_i}$ with $sign(\mathbf{W_i})$. For α_i, we take the average $L1$ norm of $\mathbf{W_i}$ as initialization.

Update α_i with $\mathbf{B_i}$ fixed By expanding Eq.(6), we have

$$\min L_i(\alpha_{\mathbf{i}}) = const + \alpha_i^2 \|\mathbf{X}^{\mathrm{T}}\mathbf{B_i}\|_F^2 - 2\alpha_i \mathbf{S_i}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{B_i}. \tag{7}$$

Then the derivative of $L_i(\alpha_{\mathbf{i}}, \mathbf{B_i})$ w.r.t α_i is:

$$\frac{\partial L_i(\alpha_\mathbf{i})}{\partial \alpha_i} = 2\alpha_i \|\mathbf{X}^\mathrm{T}\mathbf{B_i}\|_F^2 - 2\mathbf{S_i}^\mathrm{T}\mathbf{X}^\mathrm{T}\mathbf{B_i} \tag{8}$$

By setting it to zero, we get the solution of α_i:

$$\alpha_i = \frac{\mathbf{S_i}^\mathrm{T}\mathbf{X}^\mathrm{T}\mathbf{B_i}}{\|\mathbf{X}^\mathrm{T}\mathbf{B_i}\|_F^2} \tag{9}$$

Solving $\mathbf{B_i}$ with α_i fixed By expanding Equation (6), we can get:

$$\begin{aligned} \min \quad & L_i(\mathbf{B_i}) = const + \|\mathbf{Z}^\mathrm{T}\mathbf{B_i}\|_F^2 - 2\,\mathrm{Tr}\,(\mathbf{B_i}^\mathrm{T}\mathbf{q}) \\ s.t. \quad & \mathbf{B_i} \in \{+1,-1\}^{S\times 1} \end{aligned} \tag{10}$$

where $\mathbf{Z} = \alpha \cdot \mathbf{X}$,Tr() is the trace norm, and $\mathbf{q} = \alpha \cdot \mathbf{XS_i}$. Eq. (10) can be solved by *discrete cyclic coordinate descent* (DCC) method which is proposed in (Shen et al. 2015b) for solving hashing codes. Let b be the j^{th} element of $\mathbf{B_i}$, and $\mathbf{B_i}'$ the column vector of $\mathbf{B_i}$ excluding b. Similarly we denote the j^{th} element of $\mathbf{q}$ as $\mathbf{q_j}$, and let $\mathbf{q}'$ as the $\mathbf{q}$ excluding $\mathbf{q_j}$. Let $\mathbf{v}^\mathrm{T}$ be the j^{th} row of matrix $\mathbf{Z}$ and $\mathbf{Z}'$ be matrix $\mathbf{Z}$ excluding $\mathbf{v}^\mathrm{T}$. Then problem (10) can be written as:

$$\begin{aligned} \min \quad & (\mathbf{B_i}'^\mathrm{T}\mathbf{Z}'\mathbf{v} - \mathbf{q_j})b \\ s.t. \quad & b \in \{+1,-1\} \end{aligned} \tag{11}$$

Then we can get the solution for the j^{th} element of $\mathbf{B_i}$:

$$b = sign(\mathbf{q_j} - \mathbf{B_i}'^\mathrm{T}\mathbf{Z}'\mathbf{v}) \tag{12}$$

By using this method, each element of $\mathbf{B_i}$ can be iteratively updated with other $S-1$ elements of $\mathbf{B_i}$ fixed.

The convergence of our proposed alternating optimization method is guaranteed theoretically. And It can be easily proven since every update step decreases the objective function value and the objective function has a lower bound. Empirical results demonstrate that the algorithm takes a few iterations to converge. Figure 1 shows the convergence curves on different convolutional neural networks via the proposed alternating optimization method. It's clear that our algorithm get converged in a few iterations.

Layer-wise optimization

By using the proposed alternating optimization method, we optimize the binary weight layer by layer. One concern is the quantization error will be accumulated across multiple layers. More specifically, quantizing the weights of l^{th} layer will cause quantization error of output featuremaps which are the input featuremaps of $(l+1)^{th}$ layer, consequently affects the optimization procedure of $(l+1)^{th}$ layer. To solve this problem, the hashing codes are supposed to adapt to the input featuremaps which are affected by binary weights of the previous layer.

Here we adopt the similar training scheme as (Wu et al. 2016). Suppose we have a pre-trained L-layer CNN model and a binarized CNN model whose first l^{th} layers have been binarized, we denote the input featuremaps of $(l+1)^{th}$ layer for pre-trained CNN model and binarized CNN model as $\mathbf{X}^{l+1}$ and $\tilde{\mathbf{X}}^{l+1}$. The objective function would be:

$$\begin{aligned} \min L(\mathbf{A},\mathbf{B}) &= \|(\mathbf{X}^{l+1})^\mathrm{T}\mathbf{W}^{l+1} - (\tilde{\mathbf{X}}^{l+1})^\mathrm{T}\mathbf{B}^{l+1}\mathbf{A}^{l+1}\|_F^2 \\ &= \|\mathbf{S}^{l+1} - (\tilde{\mathbf{X}}^{l+1})^\mathrm{T}\mathbf{B}^{l+1}\mathbf{A}^{l+1}\|_F^2 \end{aligned} \tag{13}$$

On one hand, the target similarity matrix $\mathbf{S}^{l+1}$(in a hashing view) is calculated between $\mathbf{W}^{l+1}$ and input featuremaps $\mathbf{X}^{l+1}$. On the other hand, the realistic similarity is calculated between $\tilde{\mathbf{X}}^{l+1}$ and binary codes $\mathbf{B}^{l+1}$. Thus the binary codes is trained to adapt to quantization error of the input featuremaps. By such a layer by layer training scheme, the quantization error explosion is avoided.

The Whole Training Scheme

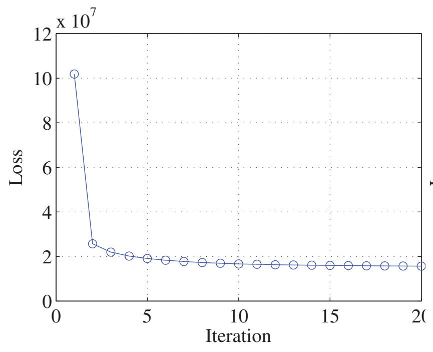
For a given pre-trained CNN model, we first use the proposed method to binarize weights of CNN model layer by layer. Then we fine-tune the binarized CNN model to get a better result. For the fine-tuning procedure, the weights of convolution layer is initialized by the learned binary codes. The scaling factor is used to initialize the weights of scale layer which is added right after the convolutional layer. We summarize the overall training algorithm in Algorithm 1.

Algorithm 1: Training Binary weight Convolutional Neural Networks via Hashing

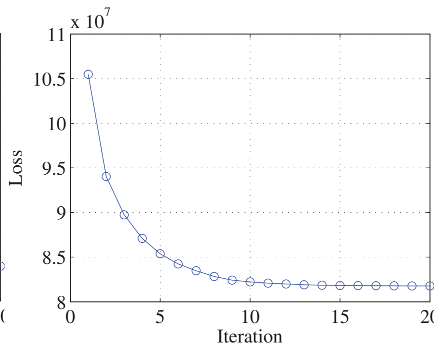
```
Input: Pre-trained convolutional neural networks
    weights $\{\mathbf{W}^l\}_{l=1}^L$ and Max_Iter
Output: Learned binary weights $\{\mathbf{B}^l\}_{l=1}^L$ and
    scaling factors $\{\mathbf{A}^l\}_{l=1}^L$
for $l = 1; l \le L$ do
    Sampling a mini-batch images from database
    Forward propagation to get $\tilde{\mathbf{X}}^l$ and $\mathbf{X}^l$
    Calculate $S$ with $\tilde{\mathbf{X}}^l$
    for $i = 1; i \le N$ do
        Initialize $\mathbf{B_i}$ with $sign(\mathbf{W_i})$
        Initialize $\alpha_i$ with mean $L1$ norm of $\mathbf{W_i}$
        while $iter \le Max\_Iters$ do
            Update $\alpha_i$ with Eq.(9)
            for $j = 1; j \le S$ do
                Update $j^{th}$ element of $\mathbf{B_i}$ with
                  Eq.(12)
            end
        end
    end
end
for $l = 1; l \le L$ do
    Initialize $l^{th}$ layer of binarized CNN model with
      $\mathbf{B}^l$
    Add a scale layer right after the $l^{th}$ layer
    Initialize weights of the scale layer with $\mathbf{A}^l$
end
Fine-tune the binarized CNN model
return $\{\mathbf{B}^l\}_{l=1}^L$ and $\{\mathbf{A}^l\}_{l=1}^L$;
```

Experiments

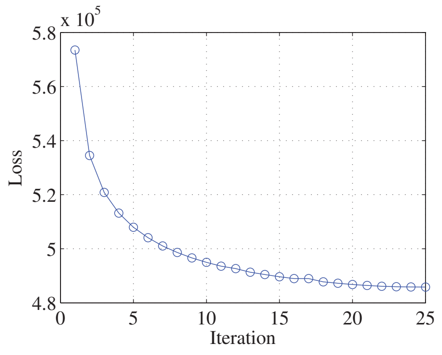
In this section, we first give details of experiment settings including datasets, network architectures, training settings and so on. Then experimental results on three datasets are analysed, showing that our proposed method outperforms



(a) optimization loss of VGG9 on CIFAR10



(b) optimization loss of AlexNet



(c) optimization loss of ResNet-18

Figure 1: The optimization loss vs Iteration using the proposed alternating optimization method on different networks

the state-of-art algorithms. Finally, we analyse the effect of scaling factor $\mathbf{A}$.

Datasets

To evaluate our proposed method, we conduct extensive experiments on three public benchmark datasets including CIFAR10, CIFAR100, and ImageNet.

- **CIFAR10** dataset consists of 60,000 colour images in 10 classes. Each class contains 6000 images in size 32×32. There are 5000 training images and 1000 testing images per class.
- **CIFAR100** dataset is like CIFAR10 except it has 60,000 colour images in 100 classes. There are 500 training images and 100 testing images per class.
- **ImageNet** dataset (ILSVRC2012) has about 1.2M training images from 1000 classes and 50,000 validation images. Compared to CIFAR10 and CIFAR100, the images in ImageNet have higher resolution and complex background.

Network Architectures

For ImageNet (ILSVRC2012), two state-of-art networks i.e. AlexNet and ResNet-18 are adopted to evaluate the proposed method. For CIFAR10 and CIFAR100, we adopt the VGG-9 network following (Dong et al. 2017).

AlexNet AlexNet consists of 5 convolutional layers and three fully-connected layers. Following (Rastegari et al. 2016; Dong et al. 2017), we use AlexNet coupled with batch normalization layers.

ResNet-18 Recently (He et al. 2016) proposed ResNet architecture which is more efficient and powerful. Base on this architecture, very deep convolutional neural networks can be trained efficiently. The ResNet architecture is built with residual blocks, which is quite different with AlexNet. Here we adopt the ResNet-18 architecture.

VGG-9 The architecture of VGG-9 is denoted as "($2\times$64C3)-MP2-($2\times$128C3)-MP2-($2\times$256C3)-MP2-($2\times$512C3)-10FC-Softmax". Here '64C3' denotes convolutional layer with 64 kernels of size 3×3. MP denotes max-pooling layer and FC denotes the fully-connected layer. Batch Normalization layer is added after each convolutional or fully-connected layer.

Training Settings

We implement our proposed method based on the Caffe framework, and the proposed alternating optimization algorithm is implemented using CUDA. All experiments are conducted on a GPU Server which has 8 Nvidia Titan Xp GPUs. During layer-wise optimization, we set maximum iterations of the proposed alternating optimization method to 20 which is enough for training according to Figure 1. We adopt different fine-tuning settings for different network architecture.

AlexNet We fine-tune AlexNet using a SGD solver with momentum=0.9, weight decay=0.0005. The learning rate starts at 0.001, and is divided by 10 after 100k, 150k, and 180k iterations. The network is fine-tuned for 200k iterations with batch-size equals to 256. Before training, images are resized to have 256 pixels at their smaller side. Random cropping and mirroring are adopted in the training stage and center cropping is used in the testing stage.

ResNet-18 We fine-tune the ResNet-18 using a SGD solver with momentum=0.9, weight decay=0.0005. The learning rate starts at 0.0005, and is divided by 10 every 200k iterations. We run the training algorithm for 650k iterations with batch size equal to 128. We use random cropping and mirroring for data augmentation. Like AlexNet, images are resized to have 256 pixels at their smaller side.

VGG-9 We use a SGD solver with momentum=0.9, weight decay=0.0001 for fine-tuning the VGG-9 network. The learning rate starts at 0.1, and is divided by 10 every 15K iterations. Following (Dong et al. 2017), the network is fine-tuned for 100K iterations with batch-size equals to 100.

Experimental Results

To evaluate our proposed method, we compare our method with BC (Courbariaux, Bengio, and David 2015), BWN (Rastegari et al. 2016), SQ-BWN (Dong et al. 2017), and HWGQ-BWN (Cai et al. 2017). Table. 1 shows the classification accuracy of VGG9 network on CIFAR10 and CIFAR100 dataset via different methods. From Table. 1, it's clear that our proposed method outperforms other state-of-art algorithms.

Table 1: Test error rate of VGG9 on CIFAR10 and CIFAR100

Method	Test error rate	
	CIFAR10	CIFAR100
Full-Precision	9.01	30.45
BinaryConnect	11.15	37.70
BWN	10.67	37.68
SQ-BWN	9.40	35.25
BWNH (Ours)	**9.21**	**34.35**

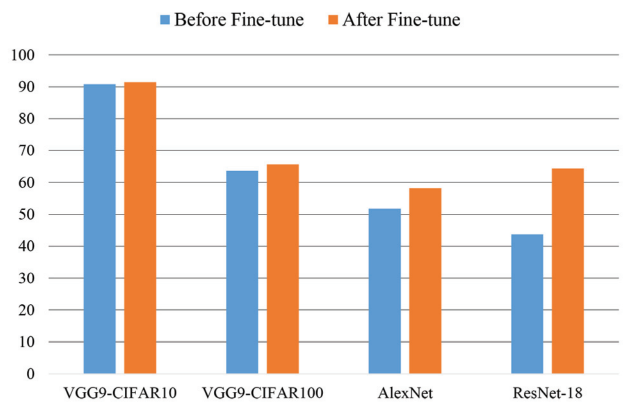


Figure 2: Top1 Accuracy of VGG9, AlexNet, and ResNet-18 with or without fine-tuning

Since CIFAR10 and CIFAR100 both are small dataset, we mainly verify our proposed method and tune the hyper-parameters on these two datasets. Here we focus on the experiment results on ImageNet. Table. 2 demonstrates the Top1 and Top5 classification accuracy of AlexNet on ImageNet dataset for different methods. And our proposed method outperforms the state-of-art methods in both Top1 and Top5 accuracy.

Table. 3 compares our proposed BWNH with other methods on ResNet-18 network. We can find that our proposed method outperforms the state-of-art method by a large margin (3% in top1 accuracy).

Table 2: Classification Accuracy of AlexNet for different methods

Method	Classification Accuracy	
	Top1	Top5
BinaryConnect	35.4	61.0
BWN	56.8	79.4
SQ-BWN	51.2	75.1
HWGQ-BWN	52.4	75.9
BWNH (Ours)	**58.5**	**80.9**

Figure. 2 shows the Top1 accuracy of different networks by using proposed BWNH with or without using a fine-tuning step. From Figure. 2, we notice that our proposed BWNH has achieved a relatively high accuracy without the fine-tuning step, which demonstrates the usefulness of learning binary weights via hashing. The binary weights learned by hashing can be used as initialization of CNNs, and a fine-tuning step will improve the accuracy of networks. Other training binary weights algorithms can also be combined with our methods by simply using our learned binary weights as initialization, which can get higher accuracy.

Table 3: Classification Accuracy of ResNet-18 for different methods

Method	Classification Accuracy	
	Top1	Top5
Full-Precision	69.3	89.2
BWN	60.8	83.0
SQ-BWN	58.3	81.6
HWGQ-BWN	61.3	83.9
BWNH (Ours)	**64.3**	**85.9**

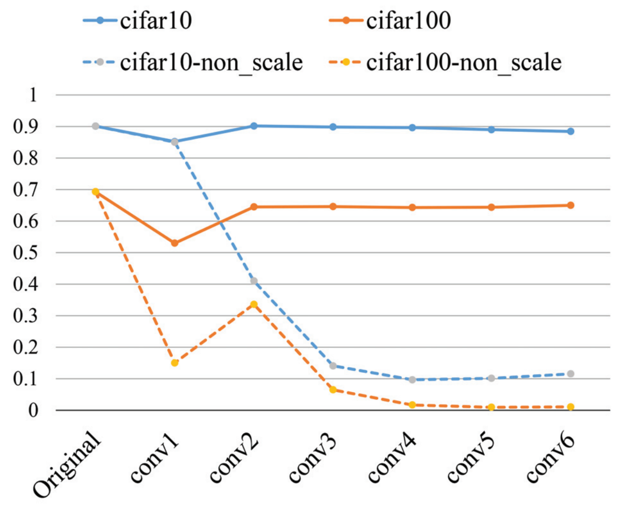


Figure 3: Accuracy of VGG9 Network on CIFAR10 and CIFAR100 after layer-wise optimization from *conv1* to *conv6*

The Effect of Scaling Factor

In this subsection, we explore the effect of scaling factor. Figure. 3 shows the accuracy of VGG9 network on CIFAR10 and CIFAR100 by using proposed BWNH with or without scaling factor. We denote the result without using scaling factor as *non_scale*. From Figure. 3, it's clear that the scaling factor is very important for the proposed method. Without the scaling factor, the accuracy of network degrades very quickly and reaches the *random* accuracy (0.1 for CIFAR10 and 0.01 for CIFAR100) after optimizing several layers. Besides, the scaling factor can be merged into the Batch Normalization layer in the inference phase, thus it won't add extra memory or storage overhead. Another interesting phenomenon in Figure. 3 is that the accuracy after optimizing *conv1* and *conv2* is higher than the accuracy after optimizing *conv1*. This is because the binary weights in *conv2* compensates the accuracy drop by adapting to the input featuremaps generated by binary weights in *conv1*.

Conclusion and Future Work

In this paper, we first uncovered the close connection between inner-product preserving hashing and binary weight networks and showed that training binary weight networks can be transformed into a hashing problem. A scaling factor is multiplied to the binary codes to improve the accuracy, then we propose an alternating optimization method to solve the problem. Experiments on CIFAR10, CIFAR100, and ImageNet show that our proposed method outperforms the state-of-art algorithms.

At present, our method aims to quantize the weights of CNNs to 1 bit (-1 or +1). In fact, our method can also be used to train a binary neural network (BNN) where the featuremaps and weights of CNN are all quantized to 1 bit. In the future, we will try to train binary neural networks using our proposed method.

Acknowledgements

This work was supported in part by National Natural Science Foundation of China (No.61332016), the Scientific Research Key Program of Beijing Municipal Commission of Education (KZ201610005012), Jiangsu Key Laboratory of Big Data Analysis Technology, and Hubei Key Laboratory of Transportation Internet of Things.

References

Cai, Z.; He, X.; Sun, J.; and Vasconcelos, N. 2017. Deep learning with low precision by half-wave gaussian quantization. *arXiv preprint arXiv:1702.00953*.

Chen, W.; Wilson, J.; Tyree, S.; Weinberger, K.; and Chen, Y. 2015. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, 2285–2294.

Courbariaux, M.; Bengio, Y.; and David, J.-P. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, 3123–3131.

Denil, M.; Shakibi, B.; Dinh, L.; de Freitas, N.; et al. 2013. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, 2148–2156.

Denton, E. L.; Zaremba, W.; Bruna, J.; LeCun, Y.; and Fergus, R. 2014. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, 1269–1277.

Dong, Y.; Ni, R.; Li, J.; Chen, Y.; Zhu, J.; and Su, H. 2017. Learning accurate low-bit deep neural networks with stochastic quantization. *arXiv preprint arXiv:1708.01001*.

Gong, Y.; Liu, L.; Yang, M.; and Bourdev, L. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*.

Gupta, S.; Agrawal, A.; Gopalakrishnan, K.; and Narayanan, P. 2015. Deep learning with limited numerical precision. *CoRR, abs/1502.02551* 392.

Han, S.; Pool, J.; Tran, J.; and Dally, W. 2015. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, 1135–1143.

Hassibi, B., and Stork, D. G. 1993. *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

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

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

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Lebedev, V., and Lempitsky, V. 2015. Fast convnets using group-wise brain damage. *arXiv preprint arXiv:1506.02515*.

Lebedev, V.; Ganin, Y.; Rakhuba, M.; Oseledets, I.; and Lempitsky, V. 2014. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*.

LeCun, Y.; Denker, J. S.; Solla, S. A.; Howard, R. E.; and Jackel, L. D. 1989. Optimal brain damage. In *Advances in Neural Information Processing Systems*, volume 89.

Lin, Z.; Courbariaux, M.; Memisevic, R.; and Bengio, Y. 2015. Neural networks with few multiplications. *arXiv preprint arXiv:1510.03009*.

Lin, D. D.; Talathi, S. S.; and Annapureddy, V. S. 2015. Fixed point quantization of deep convolutional networks. *arXiv preprint arXiv:1511.06393*.

Liu, B.; Wang, M.; Foroosh, H.; Tappen, M.; and Pensky, M. 2015. Sparse convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 806–814.

Long, J.; Shelhamer, E.; and Darrell, T. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 3431–3440.

Novikov, A.; Podoprikhin, D.; Osokin, A.; and Vetrov, D. P. 2015. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, 442–450.

Qiu, J.; Wang, J.; Yao, S.; Guo, K.; Li, B.; Zhou, E.; Yu, J.; Tang, T.; Xu, N.; Song, S.; et al. 2016. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 26–35. ACM.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, 525–542. Springer.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, 91–99.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* 115(3):211–252.

Shen, F.; Liu, W.; Zhang, S.; Yang, Y.; and Tao Shen, H. 2015a. Learning binary codes for maximum inner product search. In *Proceedings of the IEEE International Conference on Computer Vision*, 4148–4156.

Shen, F.; Shen, C.; Liu, W.; and Tao Shen, H. 2015b. Supervised discrete hashing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 37–45.

Simonyan, K., and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Spring, R., and Shrivastava, A. 2017. Scalable and sustainable deep learning via randomized hashing. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 445–454. ACM.

Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; and Wojna, Z. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2818–2826.

Wang, P., and Cheng, J. 2016. Accelerating convolutional neural networks for mobile applications. In *Proceedings of the 2016 ACM on Multimedia Conference*, 541–545. ACM.

Wu, J.; Leng, C.; Wang, Y.; Hu, Q.; and Cheng, J. 2016. Quantized convolutional neural networks for mobile devices. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhang, X.; Zou, J.; He, K.; and Sun, J. 2015. Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*.

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