From: 林苏昕 1340172084@qq.com
Subject:
Date: October 19, 2018 at 7:43 PM
To:

林

1 XNORT_NET imagenet classification using Binary CNN

Saturday, October 13, 2018
9:20 AM

<<1603.05279.pdf>>

XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks

arXiv:1603.05279v4 [cs.CV] 2 Aug 2016

Mohammad Rastegari†, Vicente Ordonez†, Joseph Redmon*, Ali Farhadi†*

Allen Institute for AI†, University of Washington*
{mohammadr,vicenteor}@allenai.org
{pjreddie,ali}@cs.washington.edu

Abstract. We propose two efficient approximations to standard convolutional neural networks: Binary-Weight-Networks and XNOR-Networks. In Binary-Weight-Networks, the filters are approximated with binary values resulting in 32× memory saving. In XNOR-Networks, both the filters and the input to convolutional layers are binary. XNOR-Networks approximate convolutions using primarily binary operations. This results in 58× faster convolutional operations (in terms of number of the high precision operations) and 32× memory savings. XNOR-Nets offer the possibility of running state-of-the-art networks on CPUs (rather than GPUs) in real-time. Our binary networks are simple, accurate, efficient, and work on challenging visual tasks. We evaluate our approach on the ImageNet classification task. The classification accuracy with a Binary-Weight-Network version of AlexNet is the same as the full-precision AlexNet. We compare our method with recent network binarization methods, BinaryConnect and BinaryNets, and outperform these methods by large margins on ImageNet, more than 16% in top-1 accuracy. Our code is available at: http://allenai.org/plato/xnornet.

1 Introduction

Deep neural networks (DNN) have shown significant improvements in several application domains including computer vision and speech recognition. In computer vision, a particular type of DNN, known as Convolutional Neural Networks (CNN), have demonstrated state-of-the-art results in object recognition [1,2,3,4] and detection [5,6,7].

Convolutional neural networks show reliable results on object recognition and detection that are useful in real world applications. Concurrent to the recent progress in recognition, interesting advancements have been happening in virtual reality (VR by Oculus) [8], augmented reality (AR by HoloLens) [9], and smart wearable devices. Putting these two pieces together, we argue that it is the right time to equip smart portable devices with the power of state-of-the-art recognition systems. However, CNN-based recognition systems need large amounts of memory and computational power. While they perform well on expensive, GPU-based machines, they are often unsuitable for smaller devices like cell phones and embedded electronics.

For example, AlexNet[1] has 61M parameters (249MB of memory) and performs 1.5B high precision operations to classify one image. These numbers are even higher for deeper CNNs *e.g.*,VGG [2] (see section 4.1). These models quickly overtax the limited storage, battery power, and compute capabilities of smaller devices like cell phones.

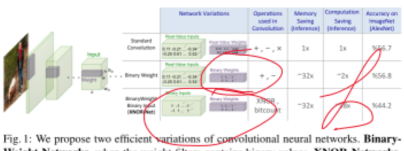


Fig. 1: We propose two efficient variations of convolutional neural networks. **Binary-Weight-Networks**, when the weight filters contains binary values. **XNOR-Networks**, when both weigh and input have binary values. These networks are very efficient in terms of memory and computation, while being very accurate in natural image classification. This offers the possibility of using accurate vision techniques in portable devices with limited resources.

In this paper, we introduce simple, efficient, and accurate approximations to CNNs by binarizing the weights and even the intermediate representations in convolutional neural networks. Our binarization method aims at finding the best approximations of the convolutions using binary operations. We demonstrate that our way of binarizing neural networks results in ImageNet classification accuracy numbers that are comparable to standard full precision networks while requiring a significantly less memory and fewer floating point operations.

We study two approximations: Neural networks with binary weights and XNOR-Networks. In **Binary-Weight-Networks** all the weight values are approximated with binary values. A convolutional neural network with binary weights is significantly smaller ($\sim 32\times$) than an equivalent network with single-precision weight values. In addition, when weight values are binary, convolutions can be estimated by only addition and subtraction (without multiplication), resulting in $\sim 2\times$ speed up. Binary-weight approximations of large CNNs can fit into the memory of even small, portable devices while maintaining the same level of accuracy (See Section 4.1 and 4.2).

To take this idea further, we introduce **XNOR-Networks** where both the weights and the inputs to the convolutional and fully connected layers are approximated with binary values[1]. Binary weights and binary inputs allow an efficient way of implementing convolutional operations. If all of the operands of the convolutions are binary, then the convolutions can be estimated by XNOR and bitcounting operations [11]. XNOR-Nets result in accurate approximation of CNNs while offering $\sim 58\times$ speed up in CPUs (in terms of number of the high precision operations). This means that XNOR-Nets can enable real-time inference in devices with small memory and no GPUs (Inference in XNOR-Nets can be done very efficiently on CPUs).

To the best of our knowledge this paper is the first attempt to present an evaluation of binary neural networks on large-scale datasets like ImageNet. Our experimental

[1] fully connected layers can be implemented by convolution, therefore, in the rest of the paper, we refer to them also as convolutional layers [10].

results show that our proposed method for binarizing convolutional neural networks outperforms the state-of-the-art network binarization method of [11] by a large margin (16.3%) on top-1 image classification in the ImageNet challenge ILSVRC2012. Our contribution is two-fold: First, we introduce a new way of binarizing the weight values in convolutional neural networks and show the advantage of our solution compared

Binary Weight value can be
Convoluted by +-(32smaller,2faster)

1Binary weight network
Filter is binary

Proposed 2 approximation of CNN

Contribution:

In Alexnet
binarization method:Over binary connect and binarynets
First On large dataset

2 Xnor network
both input and filter of convoluted layer are binary

All op binary ,XNOr and bitcounting
Op(58faster)

Challenge

1memory
2compute power
3 battery
4 DNN :over parameter and reluctant

Diff between others method to address efficient training and inference

Shallow network:

train a shallow network to mimic deep moldel
Vs
Use standard deep arc not the shallow estimation

Compress pre-train model:

pruning
Weight decay
Remove redundant connection,share same weight
Huffman code to compress weight
Hash to reduce size by randomly grouping
Vs
Not use pretrain network,train from scrach

Compact layer:

Replace full layer with global average pool
Bottleneck arc in Resnet
Vs
Not compress network but with binary weight

Quantizing parameters:

Carry full precision during the test, quantize the para in back propagation
Activation 32,8,3-bits activation
Weight +1,0,-1
Vs
Weight -1,1

Network binarization:

Expectation BackPropagagtion EBP
Variational bayes approach
Binary connects and Binary Net
VS

Implementation of their model:

Model 1 :bianary weight network

(1)Estimate binary weight :
B *a estimate W(weight)
B = sign(W)
a = average of the ||weight value||
(2)Train
bBinarize weight during the forward and back propa

to state-of-the-art solutions. Second, we introduce XNOR-Nets, a deep neural network model with binary weights and binary inputs and show that XNOR-Nets can obtain similar classification accuracies compared to standard networks while being significantly more efficient. Our code is available at: http://allenai.org/plato/xnornet

2 Related Work

Deep neural networks often suffer from over-parametrization and large amounts of redundancy in their models. This typically results in inefficient computation and memory usage[12]. Several methods have been proposed to address efficient training and inference in deep neural networks.

Shallow networks: Estimating a deep neural network with a shallower model reduces the size of a network. Early theoretical work by Cybenko shows that a network with a large enough single hidden layer of sigmoid units can approximate any decision boundary [13]. In several areas (*e.g.*,vision and speech), however, shallow networks cannot compete with deep models [14]. [15] trains a shallow network on SIFT features to classify the ImageNet dataset. They show it is difficult to train shallow networks with large number of parameters. [16] provides empirical evidence on small datasets (*e.g.*,CIFAR-10) that shallow nets are capable of learning the same functions as deep nets. In order to get the similar accuracy, the number of parameters in the shallow network must be close to the number of parameters in the deep network. They do this by first training a state-of-the-art deep model, and then training a shallow model to mimic the deep model. These methods are different from our approach because we use the standard deep architectures not the shallow estimations.

Compressing pre-trained deep networks: Pruning redundant, non-informative weights in a previously trained network reduces the size of the network at inference time. Weight decay [17] was an early method for pruning a network. Optimal Brain Damage [18] and Optimal Brain Surgeon [19] use the Hessian of the loss function to prune a network by reducing the number of connections. Recently [20] reduced the number of parameters by an order of magnitude in several state-of-the-art neural networks by pruning. [21] proposed to reduce the number of activations for compression and acceleration. Deep compression [22] reduces the storage and energy required to run inference on large networks so they can be deployed on mobile devices. They remove the redundant connections and quantize weights so that multiple connections share the same weight, and then they use Huffman coding to compress the weights. HashedNets [23] uses a hash function to reduce model size by randomly grouping the weights, such that connections in a hash bucket use a single parameter value. Matrix factorization has been used by [24,25]. We are different from these approaches because we do not use a pretrained network. We train binary networks from scratch.

Compute gradient: real_value weight in gradient descend
because binarization ignore the changes
(3)procedure
A= average of the ||weight value||
B= sign(W)
W = AB
Y^ = forward (I,A,B) excepct conv are compute val equation(1)
Gradient= back(d cost/d Y^,W) excepct gradient are computed using W
Not the real value weight.
Updateparameter: any method (SGD,ADAM)
Update learning rate(any function)

Model 2 :bianary weight network

+binarize inputs,XNOR and bitcount to compute
Binary Dot product
CY estimate XW
C = sign(X) xnor sign(W)
Y = (average of the ||X||)(average of the ||W||)
Binary Convolution
Not quite clear
Train XNOR Network
Block:
1 conv
2 batch_normalization normalize the iuput by its meam and variance
3 activation element_wise non linear function
4 pooling max min average
K_bit Quantization:

Designing compact layers: Designing compact blocks at each layer of a deep network can help to save memory and computational costs. Replacing the fully connected layer with global average pooling was examined in the Network in Network architecture [26], GoogLenet[3] and Residual-Net[4], which achieved state-of-the-art results on several benchmarks. The bottleneck structure in Residual-Net [4] has been proposed to reduce the number of parameters and improve speed. Decomposing 3×3 convolutions with two 1×1 is used in [27] and resulted in state-of-the-art performance on object recognition. Replacing 3×3 convolutions with 1×1 convolutions is used in [28] to create a very compact neural network that can achieve $\sim 50\times$ reduction in the number of parameters while obtaining high accuracy. Our method is different from this line of work because we use the full network (not the compact version) but with binary parameters. Not arc diff

Quantizing parameters: High precision parameters are not very important in achieving high performance in deep networks. [29] proposed to quantize the weights of fully connected layers in a deep network by vector quantization techniques. They showed just thresholding the weight values at zero only decreases the top-1 accuracy on ILSVRC2012 by less than %10. [30] proposed a provably polynomial time algorithm for training a sparse networks with +1/0/-1 weights. A fixed-point implementation of 8-bit integer was compared with 32-bit floating point activations in [31]. Another fixed-point network with ternary weights and 3-bits activations was presented by [32]. Quantizing a network with L_2 error minimization achieved better accuracy on MNIST and CIFAR-10 datasets in [33]. [34] proposed a back-propagation process by quantizing the representations at each layer of the network. To convert some of the remaining multiplications into binary shifts the neurons get restricted values of power-of-two integers. In [34] they carry the full precision weights during the test phase, and only quantize the neurons during the back-propagation process, and not during the forward-propagation. Our work is similar to these methods since we are quantizing the parameters in the network. But our quantization is the extreme scenario +1,-1.

三. Qustion list
Intermediate representation
EBP expectation bp
Variational Bayesian approach
Scaling factor
Xnor bitcounting operation

2的幂

SGD

Network binarization: These works are the most related to our approach. Several methods attempt to binarize the weights and the activations in neural networks.The performance of highly quantized networks (*e.g.*,binarized) were believed to be very poor due to the destructive property of binary quantization [35]. Expectation BackPropagation (EBP) in [36] showed high performance can be achieved by a network with binary weights and binary activations. This is done by a variational Bayesian approach, that infers networks with binary weights and neurons. A fully binary network at run time presented in [37] using a similar approach to EBP, showing significant improvement in energy efficiency. In EBP the binarized parameters were only used during inference. BinaryConnect [38] extended the probablistic idea behind EBP. Similar to our approach, BinaryConnect uses the real-valued version of the weights as a key reference for the binarization process. The real-valued weight updated using the back propagated error by simply ignoring the binarization in the update. BinaryConnect achieved state-of-the-art results on small datasets (*e.g.*,CIFAR-10, SVHN). Our experiments shows that this method is not very successful on large-scale datsets (*e.g.*,ImageNet). BinaryNet[11] propose an extention of BinaryConnect, where both weights and activations are binarized. Our method is different from them in the binarization method and the net-

3 network binarization
EBP expectation bp
Variational Bayesian approach
Binary connect
—-not well on large
Dataset
Theirmodle diff from binary net inbinarizatuon
method and network structure

work structure. We also compare our method with BinaryNet on ImageNet, and our method outperforms BinaryNet by a large margin.[39] argued that the noise introduced by weight binarization provides a form of regularization, which could help to improve test accuracy. This method binarizes weights while maintaining full precision activation. [40] proposed fully binary training and testing in an array of committee machines with randomized input. [41] retraine a previously trained neural network with binary weights and binary inputs.

3 Binary Convolutional Neural Network

We represent an L-layer CNN architecture with a triplet $\langle \mathcal{I}, \mathcal{W}, * \rangle$. $\mathcal{I}$ is a set of tensors, where each element $\mathbf{I} = \mathcal{I}_{l(l=1,\dots,L)}$ is the input tensor for the l^{th} layer of CNN (Green cubes in figure 1). $\mathcal{W}$ is a set of tensors, where each element in this set $\mathbf{W} = \mathcal{W}_{lk(k=1,\dots,K^l)}$ is the k^{th} weight filter in the l^{th} layer of the CNN. K^l is the number of weight filters in the l^{th} layer of the CNN. $*$ represents a convolutional operation with $\mathbf{I}$ and $\mathbf{W}$ as its operands[2]. $\mathbf{I} \in \mathbb{R}^{c \times w_{in} \times h_{in}}$, where (c, w_{in}, h_{in}) represents *channels*, *width* and *height* respectively.$\mathbf{W} \in \mathbb{R}^{c \times w \times h}$, where $w \leq w_{in}$, $h \leq h_{in}$. We propose two variations of binary CNN: **Binary-weights**, where the elements of $\mathcal{W}$ are binary tensors and **XNOR-Networks**, where elements of both $\mathcal{I}$ and $\mathcal{W}$ are binary tensors.

3.1 Binary-Weight-Networks

In order to constrain a convolutional neural network $\langle \mathcal{I}, \mathcal{W}, * \rangle$ to have binary weights, we estimate the real-value weight filter $\mathbf{W} \in \mathcal{W}$ using a binary filter $\mathbf{B} \in \{+1, -1\}^{c \times w \times h}$ and a scaling factor $\alpha \in \mathbb{R}^+$ such that $\mathbf{W} \approx \alpha \mathbf{B}$. A convolutional operation can be appriximated by:

$$\mathbf{I} * \mathbf{W} \approx (\mathbf{I} \oplus \mathbf{B})\alpha \tag{1}$$

where, $\oplus$ indicates a convolution without any multiplication. Since the weight values are binary, we can implement the convolution with additions and subtractions. The binary weight filters reduce memory usage by a factor of $\sim 32\times$ compared to single-precision filters. We represent a CNN with binary weights by $\langle \mathcal{I}, \mathcal{B}, \mathcal{A}, \oplus \rangle$, where $\mathcal{B}$ is a set of binary tensors and $\mathcal{A}$ is a set of positive real scalars, such that $\mathbf{B} = \mathcal{B}_{lk}$ is a binary filter and $\alpha = \mathcal{A}_{lk}$ is an scaling factor and $\mathcal{W}_{lk} \approx \mathcal{A}_{lk}\mathcal{B}_{lk}$

Estimating binary weights: Without loss of generality we assume $\mathbf{W}, \mathbf{B}$ are vectors in $\mathbb{R}^n$, where $n = c \times w \times h$. To find an optimal estimation for $\mathbf{W} \approx \alpha \mathbf{B}$, we solve the following optimization:

提出两种主要模型
1 BCN
Binary weight and scaling factor

$$J(\mathbf{B},\alpha) = \|\mathbf{W} - \alpha\mathbf{B}\|^2$$
$$\alpha^*, \mathbf{B}^* = \underset{\alpha,\mathbf{B}}{\operatorname{argmin}} J(\mathbf{B},\alpha)$$

minimise J function

[2] In this paper we assume convolutional filters do not have bias terms

by expanding equation 2, we have

$$J(\mathbf{B},\alpha) = \alpha^2\mathbf{B}^\intercal\mathbf{B} - 2\alpha\mathbf{W}^\intercal\mathbf{B} + \mathbf{W}^\intercal\mathbf{W} \tag{3}$$

since $\mathbf{B} \in \{+1,-1\}^n$, $\mathbf{B}^\intercal\mathbf{B} = n$ is a constant . $\mathbf{W}^\intercal\mathbf{W}$ is also a constant because $\mathbf{W}$ is a known variable. Lets define $\mathbf{c} = \mathbf{W}^\intercal\mathbf{W}$. Now, we can rewrite the equation 3 as follow: $J(\mathbf{B},\alpha) = \alpha^2 n - 2\alpha\mathbf{W}^\intercal\mathbf{B} + \mathbf{c}$. The optimal solution for $\mathbf{B}$ can be achieved by maximizing the following constrained optimization: (note that α is a positive value in equation 2, therefore it can be ignored in the maximization)

$$\mathbf{B}^* = \underset{\mathbf{B}}{\operatorname{argmax}}\{\mathbf{W}^\intercal\mathbf{B}\} \quad s.t. \;\; \mathbf{B} \in \{+1,-1\}^n \tag{4}$$

This optimization can be solved by assigning $\mathbf{B}_i = +1$ if $\mathbf{W}_i \geq 0$ and $\mathbf{B}_i = -1$ if $\mathbf{W}_i < 0$, therefore the optimal solution is $\mathbf{B}^* = \operatorname{sign}(\mathbf{W})$. In order to find the optimal value for the scaling factor α^*, we take the derivative of J with respect to α and set it to zero:

$$\alpha^* = \frac{\mathbf{W}^\intercal\mathbf{B}^*}{n} \tag{5}$$

By replacing $\mathbf{B}^*$ with $\operatorname{sign}(\mathbf{W})$

$$\alpha^* = \frac{\mathbf{W}^\intercal \operatorname{sign}(\mathbf{W})}{n} = \frac{\sum|\mathbf{W}_i|}{n} = \frac{1}{n}\|\mathbf{W}\|_{\ell 1} \tag{6}$$

therefore, the optimal estimation of a binary weight filter can be simply achieved by taking the sign of weight values. The optimal scaling factor is the average of absolute weight values.

Training Binary-Weights-Networks: Each iteration of training a CNN involves three steps; forward pass, backward pass and parameters update. To train a CNN with binary weights (in convolutional layers), we only binarize the weights during the forward pass and backward propagation. To compute the gradient for sign function $\operatorname{sign}(r)$, we follow the same approach as [11], where $\frac{\partial \operatorname{sign}}{\partial r} = r1_{|r|\leq 1}$. The gradient in backward after the scaled sign function is $\frac{\partial C}{\partial W_i} = \frac{\partial C}{\widetilde{W_i}}(\frac{1}{n} + \frac{\partial \operatorname{sign}}{\partial W_i}\alpha)$. For updating the parameters, we use the high precision (real-value) weights. Because, in gradient descend the parameter changes are tiny, binarization after updating the parameters ignores these changes and the training objective can not be improved. [11,38] also employed this strategy to train a binary network.

Algorithm 1 demonstrates our procedure for training a CNN with binary weights. First, we binarize the weight filters at each layer by computing $\mathcal{B}$ and $\mathcal{A}$. Then we call forward propagation using binary weights and its corresponding scaling factors, where all the convolutional operations are carried out by equation 1. Then, we call backward propagation, where the gradients are computed with respect to the estimated weight filters $\widetilde{\mathcal{W}}$. Lastly, the parameters and the learning rate gets updated by an update rule e.g.,SGD update with momentum or ADAM [42]. 优化方法

Once the training finished, there is no need to keep the real-value weights. Because, at inference we only perform forward propagation with the binarized weights.

Algorithm 1 Training an L-layers CNN with binary weights:

Input: A minibatch of inputs and targets $(\mathbf{I}, \mathbf{Y})$, cost function $C(\mathbf{Y}, \hat{\mathbf{Y}})$, current weight $\mathcal{W}^t$ and current learning rate η^t.

Output: updated weight $\mathcal{W}^{t+1}$ and updated learning rate η^{t+1}.

Binarizing weight filters:
for $l = 1$ to L **do**
  for k^{th} filter in l^{th} layer **do**
    $\mathcal{A}_{lk} = \frac{1}{n}\|\mathcal{W}^t_{lk}\|_{\ell 1}$
    $\mathcal{B}_{lk} = \operatorname{sign}(\mathcal{W}^t_{lk})$
    $\widetilde{\mathcal{W}}_{lk} = \mathcal{A}_{lk}\mathcal{B}_{lk}$
$\hat{\mathbf{Y}} =$ **BinaryForward**$(\mathbf{I}, \mathcal{B}, \mathcal{A})$ // standard forward propagation except that convolutions are computed using equation 1 or 11
$\frac{\partial C}{\partial \widetilde{\mathcal{W}}} =$ **BinaryBackward**$(\frac{\partial C}{\partial \hat{\mathbf{Y}}}, \widetilde{\mathcal{W}})$ // standard backward propagation except that gradients are computed using $\widetilde{\mathcal{W}}$ instead of $\mathcal{W}^t$
$\mathcal{W}^{t+1} =$ **UpdateParameters**$(\mathcal{W}^t, \frac{\partial C}{\partial \widetilde{\mathcal{W}}}, \eta_t)$ // Any update rules (e.g.,SGD or ADAM)
$\eta^{t+1} =$ **UpdateLearningrate**(η^t, t) // Any learning rate scheduling function

3.2 XNOR-Networks

XNOR-Networks:
Binarize inputs
Xnor And bitcounting op

So far, we managed to find binary weights and a scaling factor to estimate the real-value weights. The inputs to the convolutional layers are still real-value tensors. Now, we explain how to binarize both weigths and inputs, so convolutions can be implemented efficiently using XNOR and bitcounting operations. This is the key element of our XNOR-Networks. In order to constrain a convolutional neural network $\langle \mathcal{I}, \mathcal{W}, * \rangle$ to have binary weights and binary inputs, we need to enforce binary operands at each step of the convolutional operation. A convolution consist of repeating a shift operation and a dot product. Shift operation moves the weight filter over the input and the dot product performs element-wise multiplications between the values of the weight filter and the corresponding part of the input. If we express dot product in terms of binary operations, convolution can be approximated using binary operations. Dot product between two binary vectors can be implemented by XNOR-Bitcounting operations [11]. In this section, we explain how to approximate the dot product between two vectors in $\mathbb{R}^n$ by a dot product between two vectors in $\{+1,-1\}^n$. Next, we demonstrate how to use this approximation for estimating a convolutional operation between two tensors.

Binary Dot Product: To approximate the dot product between $\mathbf{X}, \mathbf{W} \in \mathbb{R}^n$ such that $\mathbf{X}^\intercal\mathbf{W} \approx \beta\mathbf{H}^\intercal\alpha\mathbf{B}$, where $\mathbf{H}, \mathbf{B} \in \{+1,-1\}^n$ and $\beta, \alpha \in \mathbb{R}^+$, we solve the following optimization:

$$\alpha^*, \mathbf{B}^*, \beta^*, \mathbf{H}* = \underset{\alpha,\mathbf{B},\beta,\mathbf{H}}{\operatorname{argmin}}\|\mathbf{X} \odot \mathbf{W} - \beta\alpha\mathbf{H} \odot \mathbf{B}\| \tag{7}$$

where $\odot$ indicates element-wise product. We define $\mathbf{Y} \in \mathbb{R}^n$ such that $\mathbf{Y}_i = \mathbf{X}_i\mathbf{W}_i$, $\mathbf{C} \in \{+1,-1\}^n$ such that $\mathbf{C}_i = \mathbf{H}_i\mathbf{B}_i$ and $\gamma \in \mathbb{R}^+$ such that $\gamma = \beta\alpha$. The equation 7 can be written as:

$$\gamma^*, \mathbf{C}^* = \underset{\gamma,\mathbf{C}}{\operatorname{argmin}}\|\mathbf{Y} - \gamma\mathbf{C}\| \tag{8}$$

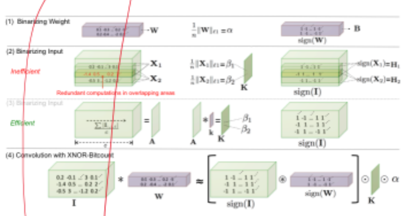


Fig. 2: This figure illustrates the procedure explained in section 3.2 for approximating a convolution using binary operations.

the optimal solutions can be achieved from equation 2 as follow

$$\mathbf{C}^* = \operatorname{sign}(\mathbf{Y}) = \operatorname{sign}(\mathbf{X}) \odot \operatorname{sign}(\mathbf{W}) = \mathbf{H}^* \odot \mathbf{B}^* \tag{9}$$

Since $|\mathbf{X}_i|, |\mathbf{W}_i|$ are independent, knowing that $\mathbf{Y}_i = \mathbf{X}_i\mathbf{W}_i$ then, $\mathbf{E}[|\mathbf{Y}_i|] = \mathbf{E}[|\mathbf{X}_i||\mathbf{W}_i|] = \mathbf{E}[|\mathbf{X}_i|]\,\mathbf{E}[|\mathbf{W}_i|]$ therefore,

$$\gamma^* = \frac{\sum_i |\mathbf{Y}_i|}{n} = \frac{\sum_i |\mathbf{X}_i||\mathbf{W}_i|}{n} \approx \left(\frac{1}{n}\|\mathbf{X}\|_{\ell 1}\right)\left(\frac{1}{n}\|\mathbf{W}\|_{\ell 1}\right) = \beta^*\alpha^* \tag{10}$$

Binary Convolution: Convolving weight filter $\mathbf{W} \in \mathbb{R}^{c \times w \times h}$ (where $w_{in} \gg w$, $h_{in} \gg h$) with the input tensor $\mathbf{I} \in \mathbb{R}^{c \times w_{in} \times h_{in}}$ requires computing the scaling factor β for all possible sub-tensors in $\mathbf{I}$ with same size as $\mathbf{W}$. Two of these sub-tensors are illustrated in figure 2 (second row) by $\mathbf{X}_1$ and $\mathbf{X}_2$. Due to overlaps between subtensors, computing β for all possible sub-tensors leads to a large number of redundant computations. To overcome this redundancy, first, we compute a matrix $\mathbf{A} = \frac{\sum |\mathbf{I}_{:,:,i}|}{c}$, which is the average over absolute values of the elements in the input $\mathbf{I}$ across the channel. Then we convolve $\mathbf{A}$ with a 2D filter $\mathbf{k} \in \mathbb{R}^{w \times h}$, $\mathbf{K} = \mathbf{A} * \mathbf{k}$, where $\forall ij \;\; \mathbf{k}_{ij} = \frac{1}{w \times h}$. $\mathbf{K}$ contains scaling factors β for all sub-tensors in the input $\mathbf{I}$. $\mathbf{K}_{ij}$ corresponds to β for a sub-tensor centered at the location ij (across width and height). This procedure is shown in the third row of figure 2. Once we obtained the scaling factor α for the weight and β for all sub-tensors in $\mathbf{I}$ (denoted by $\mathbf{K}$), we can approximate the convolution between input $\mathbf{I}$ and weight filter $\mathbf{W}$ mainly using binary operations:

$$\mathbf{I} * \mathbf{W} \approx (\operatorname{sign}(\mathbf{I}) \circledast \operatorname{sign}(\mathbf{W})) \odot \mathbf{K}\alpha \tag{11}$$

where $\circledast$ indicates a convolutional operation using XNOR and bitcount operations. This is illustrated in the last row in figure 2. Note that the number of non-binary operations is very small compared to binary operations.

Fig. 3: This figure contrasts the block structure in our XNOR-Network (right) with a typical CNN (left).

Training XNOR-Networks: A typical block in CNN contains several different layers. Figure 3 (left) illustrates a typical block in a CNN. This block has four layers in the following order: 1-Convolutional, 2-Batch Normalization, 3-Activation and 4-Pooling. Batch Normalization layer[43] normalizes the input batch by its mean and variance. The activation is an element-wise non-linear function (*e.g.*,Sigmoid, ReLU). The pooling layer applies any type of pooling (*e.g.*,max,min or average) on the input batch. Applying pooling on binary input results in significant loss of information. For example, max-pooling on binary input returns a tensor that most of its elements are equal to $+1$. Therefore, we put the pooling layer after the convolution. To further decrease the information loss due to binarization, we normalize the input before binarization. This ensures the data to hold zero mean, therefore, thresholding at zero leads to less quantization error. The order of layers in a block of binary CNN is shown in Figure 3(right).

The binary activation layer(BinActiv) computes $\mathbf{K}$ and $\operatorname{sign}(\mathbf{I})$ as explained in section 3.2. In the next layer (BinConv), given $\mathbf{K}$ and $\operatorname{sign}(\mathbf{I})$, we compute binary convolution by equation 11. Then at the last layer (Pool), we apply the pooling operations. We can insert a non-binary activation(*e.g.*,ReLU) after binary convolution. This helps when we use state-of-the-art networks (*e.g.*,AlexNet or VGG).

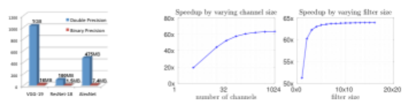
Once we have the binary CNN structure, the training algorithm would be the same as algorithm 1.

Binary Gradient: The computational bottleneck in the backward pass at each layer is computing a convolution between weight filters(w) and the gradients with respect of the inputs (g^{in}). Similar to binarization in the forward pass, we can binarize g^{in} in the backward pass. This leads to a very efficient training procedure using binary operations. Note that if we use equation 6 to compute the scaling factor for g^{in}, the direction of maximum change for SGD would be diminished. To preserve the maximum change in all dimensions, we use $\max_i(|g_i^{in}|)$ as the scaling factor.

k-bit Quantization: So far, we showed 1-bit quantization of weights and inputs using $\operatorname{sign}(x)$ function. One can easily extend the quantization level to k-bits by using $q_k(x) = 2(\frac{[(2^k-1)(\frac{x+1}{2})]}{2^k-1} - \frac{1}{2})$ instead of the sign function. Where $[.]$ indicates rounding operation and $x \in [-1, 1]$.

4 Experiments

We evaluate our method by analyzing its efficiency and accuracy. We measure the efficiency by computing the computational speedup (in terms of number of high precision operation) achieved by our binary convolution vs. standard convolution. To mea-



(a) (b) (c)

Fig. 4: This figure shows the efficiency of binary convolutions in terms of memory(a) and computation(b-c). (a) is contrasting the required memory for binary and double precision weights in three different architectures(AlexNet, ResNet-18 and VGG-19). (b,c) Show speedup gained by binary convolution under (b)-different number of channels and (c)-different filter size

sure accuracy, we perform image classification on the large-scale ImageNet dataset. This paper is the first work that evaluates binary neural networks on the ImageNet dataset. Our binarization technique is general, we can use any CNN architecture. We evaluate AlexNet [1] and two deeper architectures in our experiments. We compare our method with two recent works on binarizing neural networks; BinaryConnect [38] and BinaryNet [11]. The classification accuracy of our binary-weight-network version of AlexNet is as accurate as the full precision version of AlexNet. This classification accuracy outperforms competitors on binary neural networks by a large margin. We also present an ablation study, where we evaluate the key elements of our proposed method; computing scaling factors and our block structure for binary CNN. We shows that our method of computing the scaling factors is important to reach high accuracy.

4.1 Efficiency Analysis

In an standard convolution, the total number of operations is $cN_{\mathbf{W}}N_{\mathbf{I}}$, where c is the number of channels, $N_{\mathbf{W}} = wh$ and $N_{\mathbf{I}} = w_{in}h_{in}$. Note that some modern CPUs can fuse the multiplication and addition as a single cycle operation. On those CPUs, Binary-Weight-Networks does not deliver speed up. Our binary approximation of convolution (equation 11) has $cN_{\mathbf{W}}N_{\mathbf{I}}$ binary operations and $N_{\mathbf{I}}$ non-binary operations. With the current generation of CPUs, we can perform 64 binary operations in one clock of CPU, therefore the speedup can be computed by $S = \frac{cN_{\mathbf{W}}N_{\mathbf{I}}}{\frac{1}{64}cN_{\mathbf{W}}N_{\mathbf{I}} + N_{\mathbf{I}}} = \frac{64cN_{\mathbf{W}}}{cN_{\mathbf{W}} + 64}$.

The speedup depends on the channel size and filter size but not the input size. In figure 4-(b-c) we illustrate the speedup achieved by changing the number of channels and filter size. While changing one parameter, we fix other parameters as follows: $c = 256$, $n_{\mathbf{I}} = 14^2$ and $n_{\mathbf{W}} = 3^2$ (majority of convolutions in ResNet[4] architecture have this structure). Using our approximation of convolution we gain $62.27\times$ theoretical speed up, but in our CPU implementation with all of the overheads, we achieve 58x speed up in one convolution (Excluding the process for memory allocation and memory access). With the small channel size ($c = 3$) and filter size ($N_{\mathbf{W}} = 1 \times 1$) the speedup is not considerably high. This motivates us to avoid binarization at the first and last

layer of a CNN. In the first layer the chanel size is 3 and in the last layer the filter size is 1×1. A similar strategy was used in [11]. Figure 4-a shows the required memory for three different CNN architectures(AlexNet, VGG-19, ResNet-18) with binary and double precision weights. Binary-weight-networks are so small that can be easily fitted into portable devices. BinaryNet [11] is in the same order of memory and computation efficiency as our method. In Figure 4, we show an analysis of computation and memory cost for a binary convolution. The same analysis is valid for BinaryNet and BinaryConnect. The key difference of our method is using a scaling-factor, which does not change the order of efficiency while providing a significant improvement in accuracy.

4.2 Image Classification

We evaluate the performance of our proposed approach on the task of natural image classification. So far, in the literature, binary neural network methods have presented their evaluations on either limited domain or simplified datasets *e.g.,*CIFAR-10, MNIST, SVHN. To compare with state-of-the-art vision, we evaluate our method on ImageNet (ILSVRC2012). ImageNet has ∼1.2M train images from 1K categories and 50K validation images. The images in this dataset are natural images with reasonably high resolution compared to the CIFAR and MNIST dataset, which have relatively small images. We report our classification performance using Top-1 and Top-5 accuracies. We adopt three different CNN architectures as our base architectures for binarization: AlexNet [1], Residual Networks (known as ResNet) [4], and a variant of GoogLenet [3].We compare our Binary-weight-network (**BWN**) with BinaryConnect(**BC**) [38] and our XNOR-Networks(**XNOR-Net**) with BinaryNeuralNet(**BNN**) [11]. BinaryConnect(BC) is a method for training a deep neural network with binary weights during forward and backward propagations. Similar to our approach, they keep the real-value weights during the updating parameters step. Our binarization is different from BC. The binarization in BC can be either deterministic or stochastic. We use the deterministic binarization for BC in our comparisons because the stochastic binarization is not efficient. The same evaluation settings have been used and discussed in [11]. BinaryNeuralNet(BNN) [11] is a neural network with binary weights and activations during inference and gradient computation in training. In concept, this is a similar approach to our XNOR-Network but the binarization method and the network structure in BNN is different from ours. Their training algorithm is similar to BC and they used deterministic binarization in their evaluations.

CIFAR-10 : BC and BNN showed near state-of-the-art performance on CIFAR-10, MNIST, and SVHN dataset. BWN and XNOR-Net on CIFAR-10 using the same network architecture as BC and BNN achieve the error rate of 9.88% and 10.17% respectively. In this paper we explore the possibility of obtaining near state-of-the-art results on a much larger and more challenging dataset (ImageNet).

AlexNet: [1] is a CNN architecture with 5 convolutional layers and two fully-connected layers. This architecture was the first CNN architecture that showed to be successful on ImageNet classification task. This network has 61M parameters. We use AlexNet coupled with batch normalization layers [43].

Train: In each iteration of training, images are resized to have 256 pixel at their smaller dimension and then a random crop of 224×224 is selected for training. We run

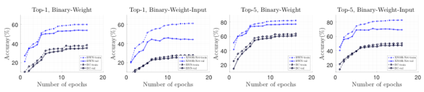


Fig. 5: This figure compares the imagenet classification accuracy on Top-1 and Top-5 across training epochs. Our approaches BWN and XNOR-Net outperform BinaryConnect(BC) and BinaryNet(BNN) in all the epochs by large margin(∼17%).

Classification Accuracy(%)									
Binary-Weight				Binary-Input-Binary-Weight				Full-Precision	
BWN		BC[11]		XNOR-Net		BNN[11]		AlexNet[1]	
Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
56.8	**79.4**	35.4	61.0	**44.2**	**69.2**	27.9	50.42	56.6	80.2

Table 1: This table compares the final accuracies (Top1 - Top5) of the full precision network with our binary precision networks; Binary-Weight-Networks(BWN) and XNOR-Networks(XNOR-Net) and the competitor methods; BinaryConnect(BC) and BinaryNet(BNN).

the training algorithm for 16 epochs with batche size equal to 512. We use negative-log-likelihood over the soft-max of the outputs as our classification loss function. In our implementation of AlexNet we do not use the Local-Response-Normalization(LRN) layer[3]. We use SGD with momentum=0.9 for updating parameters in BWN and BC. For XNOR-Net and BNN we used ADAM [42]. ADAM converges faster and usually achieves better accuracy for binary inputs [11]. The learning rate starts at 0.1 and we apply a learning-rate-decay=0.01 every 4 epochs.

Test: At inference time, we use the 224×224 center crop for forward propagation.

Figure 5 demonstrates the classification accuracy for training and inference along the training epochs for top-1 and top-5 scores. The dashed lines represent training accuracy and solid lines shows the validation accuracy. In all of the epochs our method outperforms BC and BNN by large margin (∼17%). Table 1 compares our final accuracy with BC and BNN. We found that the scaling factors for the weights (α) is much more effective than the scaling factors for the inputs (β). Removing β reduces the accuracy by a small margin (less than 1% top-1 alexnet).

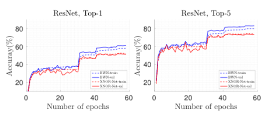
Binary Gradient: Using XNOR-Net with binary gradient the accuracy of top-1 will drop only by 1.4%.

Residual Net : We use the ResNet-18 proposed in [4] with short-cut type B.[4]

Train: In each training iteration, images are resized randomly between 256 and 480 pixel on the smaller dimension and then a random crop of 224×224 is selected for training. We run the training algorithm for 58 epochs with batch size equal to 256

[3] Our implementation is followed by https://gist.github.com/szagoruyko/dd032c529048492630fc

[4] We used the Torch implementation in https://github.com/facebook/fb.resnet.torch



(a) (b)

Fig. 6: This figure shows the classification accuracy; (a)Top-1 and (b)Top-5 measures across the training epochs on ImageNet dataset by Binary-Weight-Network and XNOR-Network using ResNet-18.

	ResNet-18		GoogLenet	
Network Variations	top-1	top-5	top-1	top-5
Binary-Weight-Network	60.8	83.0	65.5	86.1
XNOR-Network	51.2	73.2	N/A	N/A
Full-Precision-Network	69.3	89.2	71.3	90.0

Table 2: This table compares the final classification accuracy achieved by our binary precision networks with the full precision network in ResNet-18 and GoogLenet architectures.

images. The learning rate starts at 0.1 and we use the learning-rate-decay equal to 0.01 at epochs number 30 and 40.

Test: At inference time, we use the 224×224 center crop for forward propagation.

Figure 6 demonstrates the classification accuracy (Top-1 and Top-5) along the epochs for training and inference. The dashed lines represent training and the solid lines represent inference. Table 2 shows our final accuracy by BWN and XNOR-Net.

GoogLenet Variant : We experiment with a variant of GoogLenet [3] that uses a similar number of parameters and connections but only straightforward convolutions, no branching[5]. It has 21 convolutional layers with filter sizes alternating between 1×1 and 3×3.

Train: Images are resized randomly between 256 and 320 pixel on the smaller dimension and then a random crop of 224×224 is selected for training. We run the training algorithm for 80 epochs with batch size of 128. The learning rate starts at 0.1 and we use polynomial rate decay, $\beta = 4$.

Test: At inference time, we use a center crop of 224×224.

4.3 Ablation Studies

There are two key differences between our method and the previous network binariaza-tion methods; the binararization technique and the block structure in our binary CNN.

[5] We used the Darknet [44] implementation: http://pjreddie.com/darknet/imagenet/#extraction

Binary-Weight-Network		
Strategy for computing α	top-1	top-5
Using equation 6	56.8	79.4
Using a separate layer	46.2	69.5

(a)

XNOR-Network		
Block Structure	top-1	top-5
C-B-A-P	30.3	57.5
B-A-C-P	44.2	69.2

(b)

Table 3: In this table, we evaluate two key elements of our approach; computing the optimal scaling factors and specifying the right order for layers in a block of CNN with binary input. (a) demonstrates the importance of the scaling factor in training binary-weight-networks and (b) shows that our way of ordering the layers in a block of CNN is crucial for training XNOR-Networks. C,B,A,P stands for Convolutional, BatchNormalization, Acive function (here binary activation), and Pooling respectively.

For binarization, we find the optimal scaling factors at each iteration of training. For the block structure, we order the layers in a block in a way that decreases the quantization loss for training XNOR-Net. Here, we evaluate the effect of each of these elements in the performance of the binary networks. Instead of computing the scaling factor α using equation 6, one can consider α as a network parameter. In other words, a layer after binary convolution multiplies the output of convolution by an scalar parameter for each filter. This is similar to computing the affine parameters in batch normalization. Table 3-a compares the performance of a binary network with two ways of computing the scaling factors. As we mentioned in section 3.2 the typical block structure in CNN is not suitable for binarization. Table 3-b compares the standard block structure C-B-A-P (Convolution, Batch Normalization, Activation, Pooling) with our structure B-A-C-P. (A, is binary activation).

5 Conclusion

We introduce simple, efficient, and accurate binary approximations for neural networks. We train a neural network that learns to find binary values for weights, which reduces the size of network by $\sim 32\times$ and provide the possibility of loading very deep neural networks into portable devices with limited memory. We also propose an architecture, XNOR-Net, that uses mostly bitwise operations to approximate convolutions. This provides $\sim 58\times$ speed up and enables the possibility of running the inference of state of the art deep neural network on CPU (rather than GPU) in real-time.

Acknowledgements

This work is in part supported by ONR N00014-13-1-0720, NSF IIS- 1338054, Allen Distinguished Investigator Award, and the Allen Institute for Artificial Intelligence.

References

1. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. (2012) 1097–1105 1, 10, 11, 12
2. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014) 1
3. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2015) 1–9 1, 4, 11, 13
4. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR (2015) 1, 4, 10, 11, 12
5. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2014) 580–587 1
6. Girshick, R.: Fast r-cnn. In: Proceedings of the IEEE International Conference on Computer Vision. (2015) 1440–1448 1
7. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: Advances in Neural Information Processing Systems. (2015) 91–99 1
8. Oculus, V.: Oculus rift-virtual reality headset for 3d gaming. URL: http://www. oculusvr.com (2012) 1
9. Gottmer, M.: Merging reality and virtuality with microsoft hololens. (2015) 1
10. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2015) 3431–3440 2
11. Courbariaux, M., Bengio, Y.: Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. CoRR (2016) 2, 3, 4, 6, 7, 10, 11, 12
12. Denil, M., Shakibi, B., Dinh, L., de Freitas, N., et al.: Predicting parameters in deep learning. In: Advances in Neural Information Processing Systems. (2013) 2148–2156 3
13. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems **2**(4) (1989) 303–314 3
14. Seide, F., Li, G., Yu, D.: Conversational speech transcription using context-dependent deep neural networks. In: Interspeech. (2011) 437–440 3
15. Dauphin, Y.N., Bengio, Y.: Big neural networks waste capacity. arXiv preprint arXiv:1301.3583 (2013) 3
16. Ba, J., Caruana, R.: Do deep nets really need to be deep? In: Advances in neural information processing systems. (2014) 2654–2662 3
17. Hanson, S.J., Pratt, L.Y.: Comparing biases for minimal network construction with back-propagation. In: Advances in neural information processing systems. (1989) 177–185 3
18. LeCun, Y., Denker, J.S., Solla, S.A., Howard, R.E., Jackel, L.D.: Optimal brain damage. In: NIPs. Volume 89. (1989) 3
19. Hassibi, B., Stork, D.G.: Second order derivatives for network pruning: Optimal brain surgeon. Morgan Kaufmann (1993) 3
20. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information Processing Systems. (2015) 1135–1143 3
21. Van Nguyen, H., Zhou, K., Vemulapalli, R.: Cross-domain synthesis of medical images using efficient location-sensitive deep network. In: Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015. Springer (2015) 677–684 3

22. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149 (2015) 3
23. Chen, W., Wilson, J.T., Tyree, S., Weinberger, K.Q., Chen, Y.: Compressing neural networks with the hashing trick. arXiv preprint arXiv:1504.04788 (2015) 3
24. Denton, E.L., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R.: Exploiting linear structure within convolutional networks for efficient evaluation. In: Advances in Neural Information Processing Systems. (2014) 1269–1277 3
25. Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up convolutional neural networks with low rank expansions. arXiv preprint arXiv:1405.3866 (2014) 3
26. Lin, M., Chen, Q., Yan, S.: Network in network. arXiv preprint arXiv:1312.4400 (2013) 4
27. Szegedy, C., Ioffe, S., Vanhoucke, V.: Inception-v4, inception-resnet and the impact of residual connections on learning. CoRR (2016) 4
28. Iandola, F.N., Moskewicz, M.W., Ashraf, K., Han, S., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 1mb model size. arXiv preprint arXiv:1602.07360 (2016) 4
29. Gong, Y., Liu, L., Yang, M., Bourdev, L.: Compressing deep convolutional networks using vector quantization. arXiv preprint arXiv:1412.6115 (2014) 4
30. Arora, S., Bhaskara, A., Ge, R., Ma, T.: Provable bounds for learning some deep representations. arXiv preprint arXiv:1310.6343 (2013) 4
31. Vanhoucke, V., Senior, A., Mao, M.Z.: Improving the speed of neural networks on cpus. In: Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop. Volume 1. (2011) 4
32. Hwang, K., Sung, W.: Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1. In: Signal Processing Systems (SiPS), 2014 IEEE Workshop on, IEEE (2014) 1–6 4
33. Anwar, S., Hwang, K., Sung, W.: Fixed point optimization of deep convolutional neural networks for object recognition. In: Acoustics, Speech and Signal Processing (ICASSP)

networks for object recognition. In: Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on, IEEE (2015) 1131–1135 4
34. Lin, Z., Courbariaux, M., Memisevic, R., Bengio, Y.: Neural networks with few multiplications. arXiv preprint arXiv:1510.03009 (2015) 4
35. Courbariaux, M., Bengio, Y., David, J.P.: Training deep neural networks with low precision multiplications. arXiv preprint arXiv:1412.7024 (2014) 4
36. Soudry, D., Hubara, I., Meir, R.: Expectation backpropagation: parameter-free training of multilayer neural networks with continuous or discrete weights. In: Advances in Neural Information Processing Systems. (2014) 963–971 4
37. Esser, S.K., Appuswamy, R., Merolla, P., Arthur, J.V., Modha, D.S.: Backpropagation for energy-efficient neuromorphic computing. In: Advances in Neural Information Processing Systems. (2015) 1117–1125 4
38. Courbariaux, M., Bengio, Y., David, J.P.: Binaryconnect: Training deep neural networks with binary weights during propagations. In: Advances in Neural Information Processing Systems. (2015) 3105–3113 4, 6, 10, 11
39. Wan, L., Zeiler, M., Zhang, S., Cun, Y.L., Fergus, R.: Regularization of neural networks using dropconnect. In: Proceedings of the 30th International Conference on Machine Learning (ICML-13). (2013) 1058–1066 5
40. Baldassi, C., Ingrosso, A., Lucibello, C., Saglietti, L., Zecchina, R.: Subdominant dense clusters allow for simple learning and high computational performance in neural networks with discrete synapses. Physical review letters **115**(12) (2015) 128101 5
41. Kim, M., Smaragdis, P.: Bitwise neural networks. arXiv preprint arXiv:1601.06071 (2016) 5

42. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014) 6, 12
43. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015) 9, 11
44. Redmon, J.: Darknet: Open source neural networks in c. http://pjreddie.com/darknet/ (2013–2016) 13