# TRAINED TERNARY QUANTIZATION

**Chenzhuo Zhu***
Tsinghua University
zhucz13@mails.tsinghua.edu.cn

**Song Han**
Stanford University
songhan@stanford.edu

**Huizi Mao**
Stanford University
huizi@stanford.edu

**William J. Dally**
Stanford University
NVIDIA
dally@stanford.edu

## ABSTRACT

Deep neural networks are widely used in machine learning applications. However, the deployment of large neural networks models can be difficult to deploy on mobile devices with limited power budgets. To solve this problem, we propose Trained Ternary Quantization (TTQ), a method that can reduce the precision of weights in neural networks to ternary values. This method has very little accuracy degradation and can even improve the accuracy of some models (32, 44, 56-layer ResNet) on CIFAR-10 and AlexNet on ImageNet. And our AlexNet model is trained from scratch, which means it's as easy as to train normal full precision model. We highlight our trained quantization method that can learn both ternary values and ternary assignment. During inference, only ternary values (2-bit weights) and scaling factors are needed, therefore our models are nearly $16\times$ smaller than full-precision models. Our ternary models can also be viewed as sparse binary weight networks, which can potentially be accelerated with custom circuit. Experiments on CIFAR-10 show that the ternary models obtained by trained quantization method *outperform* full-precision models of ResNet-32,44,56 by 0.04%, 0.16%, 0.36%, respectively. On ImageNet, our model outperforms full-precision AlexNet model by 0.3% of Top-1 accuracy and outperforms previous ternary models by 3%.

## 1 INTRODUCTION

Deep neural networks are becoming the preferred approach for many machine learning applications. However, as networks get deeper, deploying a network with a large number of parameters on a small device becomes increasingly difficult. Much work has been done to reduce the size of networks. Half-precision networks (Amodei et al., 2015) cut sizes of neural networks in half. XNOR-Net (Rastegari et al., 2016), DoReFa-Net (Zhou et al., 2016) and network binarization (Courbariaux et al.; 2015; Lin et al., 2015) use aggressively quantized weights, activations and gradients to further reduce computation during training. While weight binarization benefits from $32\times$ smaller model size, the extreme compression rate comes with a loss of accuracy. Hubara et al. (2016) and Li & Liu (2016) propose ternary weight networks to trade off between model size and accuracy.

In this paper, we propose Trained Ternary Quantization which uses two full-precision scaling coefficients $W_l^p$, $W_l^n$ for each layer $l$, and quantize the weights to $\{-W_l^n, 0, +W_l^p\}$ instead of traditional {-1, 0, +1} or {-E, 0, +E} where E is the mean of the absolute weight value, which is not learned. Our positive and negative weights have different absolute values $W_l^p$ and $W_l^n$ that are trainable parameters. We also maintain latent full-precision weights at training time, and discard them at test time. We back propagate the gradient to both $W_l^p$, $W_l^n$ and to the latent full-precision weights. This makes it possible to adjust the ternary assignment (i.e. which of the three values a weight is assigned).

Our quantization method, achieves higher accuracy on the CIFAR-10 and ImageNet datasets. For AlexNet on ImageNet dataset, our method outperforms previously state-of-art ternary network(Li &

*Work done while at Stanford CVA lab.

Liu, 2016) by 3.0% of Top-1 accuracy and the full-precision model by 1.6%. By converting most of the parameters to 2-bit values, we also compress the network by about 16x. Moreover, the advantage of few multiplications still remains, because $W_l^p$ and $W_l^n$ are fixed for each layer during inference. On custom hardware, multiplications can be pre-computed on activations, so only two multiplications per activation are required.

## 2 MOTIVATIONS

The potential of deep neural networks, once deployed to mobile devices, has the advantage of lower latency, no reliance on the network, and better user privacy. However, energy efficiency becomes the bottleneck for deploying deep neural networks on mobile devices because mobile devices are battery constrained. Current deep neural network models consist of hundreds of millions of parameters. Reducing the size of a DNN model makes the deployment on edge devices easier.

First, a smaller model means less overhead when exporting models to clients. Take autonomous driving for example; Tesla periodically copies new models from their servers to customers' cars. Smaller models require less communication in such over-the-air updates, making frequent updates more feasible. Another example is on Apple Store; apps above 100 MB will not download until you connect to Wi-Fi. It's infeasible to put a large DNN model in an app. The second issue is energy consumption. Deep learning is energy consuming, which is problematic for battery-constrained mobile devices. As a result, iOS 10 requires iPhone to be plugged with charger while performing photo analysis. Fetching DNN models from memory takes more than two orders of magnitude more energy than arithmetic operations. Smaller neural networks require less memory bandwidth to fetch the model, saving the energy and extending battery life. The third issue is area cost. When deploying DNNs on Application-Specific Integrated Circuits (ASICs), a sufficiently small model can be stored directly on-chip, and smaller models enable a smaller ASIC die.

Several previous works aimed to improve energy and spatial efficiency of deep networks. One common strategy proven useful is to quantize 32-bit weights to one or two bits, which greatly reduces model size and saves memory reference. However, experimental results show that compressed weights usually come with degraded performance, which is a great loss for some performance-sensitive applications. The contradiction between compression and performance motivates us to work on trained ternary quantization, minimizing performance degradation of deep neural networks while saving as much energy and space as possible.

## 3 RELATED WORK

### 3.1 BINARY NEURAL NETWORK (BNN)

Lin et al. (2015) proposed binary and ternary connections to compress neural networks and speed up computation during inference. They used similar probabilistic methods to convert 32-bit weights into binary values or ternary values, defined as:

$$\begin{aligned} w^b &\sim \text{Bernoulli}(\frac{\tilde{w}+1}{2}) \times 2 - 1 \\ w^t &\sim \text{Bernoulli}(|\tilde{w}|) \times \text{sign}(\tilde{w}) \end{aligned} \tag{1}$$

Here $w^b$ and $w^t$ denote binary and ternary weights after quantization. $\tilde{w}$ denotes the latent full precision weight.

During back-propagation, as the above quantization equations are not differentiable, derivatives of expectations of the Bernoulli distribution are computed instead, yielding the identity function:

$$\frac{\partial L}{\partial \tilde{w}} = \frac{\partial L}{\partial w^b} = \frac{\partial L}{\partial w^t} \tag{2}$$

Here $L$ is the loss to optimize.

For BNN with binary connections, only quantized binary values are needed for inference. Therefore a $32\times$ smaller model can be deployed into applications.

### 3.2 DoReFa-Net

Zhou et al. (2016) proposed DoReFa-Net which quantizes weights, activations and gradients of neural networks using different widths of bits. Therefore with specifically designed low-bit multiplication algorithm or hardware, both training and inference stages can be accelerated.

They also introduced a much simpler method to quantize 32-bit weights to binary values, defined as:

$$w^b = \boldsymbol{E}(|\tilde{w}|) \times \text{sign}(\tilde{w}) \tag{3}$$

Here $\boldsymbol{E}(|\tilde{w}|)$ calculates the mean of absolute values of full precision weights $\tilde{w}$ as layer-wise scaling factors. During back-propagation, Equation 2 still applies.

### 3.3 Ternary Weight Networks

Li & Liu (2016) proposed TWN (Ternary weight networks), which reduce accuracy loss of binary networks by introducing zero as a third quantized value. They use two symmetric thresholds $\pm\Delta_l$ and a scaling factor $W_l$ for each layer $l$ to quantize weighs into $\{-W_l, 0, +W_l\}$:

$$w_l^t = \begin{cases} W_l : \tilde{w}_l > \Delta_l \\ 0 : |\tilde{w}_l| \leq \Delta_l \\ -W_l : \tilde{w}_l < -\Delta_l \end{cases} \tag{4}$$

They then solve an optimization problem of minimizing L2 distance between full precision and ternary weights to obtain layer-wise values of $W_l$ and $\Delta_l$:

$$\begin{aligned} \Delta_l &= 0.7 \times \boldsymbol{E}(|\tilde{w}_l|) \\ W_l &= \underset{i \in \{i | \tilde{w}_l(i) | > \Delta\}}{\boldsymbol{E}} (|\tilde{w}_l(i)|) \end{aligned} \tag{5}$$

And again Equation 2 is used to calculate gradients. While an additional bit is required for ternary weights, TWN achieves a validation accuracy that is very close to full precision networks according to their paper.

### 3.4 Deep Compression

Han et al. (2015) proposed deep compression to prune away trivial connections and reduce precision of weights. Unlike above models using zero or symmetric thresholds to quantize high precision weights, Deep Compression used clusters to categorize weights into groups. In Deep Compression, low precision weights are fine-tuned from a pre-trained full precision network, and the assignment of each weight is established at the beginning and stay unchanged, while representative value of each cluster is updated throughout fine-tuning.

## 4 Method

Our method is illustrated in Figure 1. First, we normalize the full-precision weights to the range [-1, +1] by dividing each weight by the maximum weight. Next, we quantize the intermediate full-resolution weights to {-1, 0, +1} by thresholding. The threshold factor $t$ is a hyper-parameter that is the same across all the layers in order to reduce the search space. Finally, we perform trained quantization by back propagating two gradients, as shown in the dashed lines in Figure 1. We back-propagate $gradient_1$ to the full-resolution weights and $gradient_2$ to the scaling coefficients. The former enables learning the ternary **assignments**, and the latter enables learning the ternary **values**.

At inference time, we throw away the full-resolution weights and only use ternary weights.

### 4.1 Learning both Ternary Values and Ternary Assignments

During gradient descent we learn both the quantized ternary weights (the codebook), and choose which of these values is assigned to each weight (choosing the codebook index).

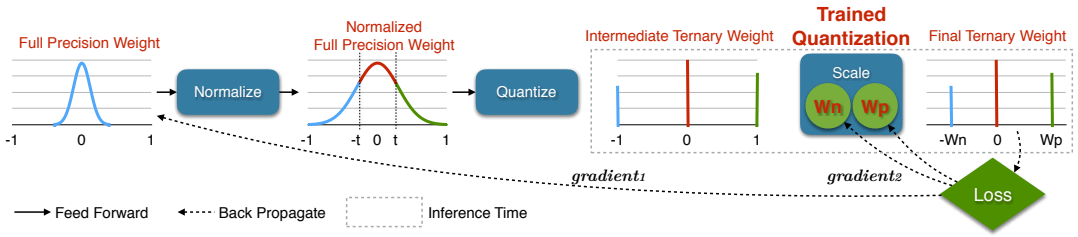


Figure 1: Overview of the trained ternary quantization procedure.

To learn the ternary value (codebook), we introduce two quantization factors $W_l^p$ and $W_l^n$ for positive and negative weights in each layer $l$. During feed-forward, quantized ternary weights $w_l^t$ are calculated as:

$$w_l^t = \begin{cases} W_l^p : \tilde{w}_l > \Delta_l \\ 0 : |\tilde{w}_l| \leq \Delta_l \\ -W_l^n : \tilde{w}_l < -\Delta_l \end{cases} \tag{6}$$

Unlike previous work where quantized weights are calculated from 32-bit weights, the scaling coefficients $W_l^p$ and $W_l^n$ are two independent parameters and are trained together with other parameters. Following the rule of gradient descent, derivatives of $W_l^p$ and $W_l^n$ are calculated as:

$$\frac{\partial L}{\partial W_l^p} = \sum_{i \in I_l^p} \frac{\partial L}{\partial w_l^t(i)}, \frac{\partial L}{\partial W_l^n} = \sum_{i \in I_l^n} \frac{\partial L}{\partial w_l^t(i)} \tag{7}$$

Here $I_l^p = \{i|\tilde{w}_l(i) > \Delta_l\}$ and $I_l^n = \{i|(i)\tilde{w}_l < -\Delta_l\}$. Furthermore, because of the existence of two scaling factors, gradients of latent full precision weights can no longer be calculated by Equation 2. We use scaled gradients for 32-bit weights:

$$\frac{\partial L}{\partial \tilde{w}_l} = \begin{cases} W_l^p \times \dfrac{\partial L}{\partial w_l^t} : \tilde{w}_l > \Delta_l \\ 1 \times \dfrac{\partial L}{\partial w_l^t} : |\tilde{w}_l| \leq \Delta_l \\ W_l^n \times \dfrac{\partial L}{\partial w_l^t} : \tilde{w}_l < -\Delta_l \end{cases} \tag{8}$$

Note we use scalar number 1 as factor of gradients of zero weights. The overall quantization process is illustrated as Figure 1. The evolution of the ternary weights from different layers during training is shown in Figure 2. We observe that as training proceeds, different layers behave differently: for the first quantized conv layer, the absolute values of $W_l^p$ and $W_l^n$ get smaller and sparsity gets lower, while for the last conv layer and fully connected layer, the absolute values of $W_l^p$ and $W_l^n$ get larger and sparsity gets higher.

We learn the ternary assignments (index to the codebook) by updating the latent full-resolution weights during training. This may cause the assignments to change between iterations. Note that the thresholds are not constants as the maximal absolute values change over time. Once an updated weight crosses the threshold, the ternary assignment is changed.

The benefits of using trained quantization factors are: i) The asymmetry of $W_l^p \neq W_l^n$ enables neural networks to have more model capacity. ii) Quantized weights play the role of "learning rate multipliers" during back propagation.

### 4.2 Quantization Heuristic

In previous work on ternary weight networks, Li & Liu (2016) proposed Ternary Weight Networks (TWN) using $\pm\Delta_l$ as thresholds to reduce 32-bit weights to ternary values, where $\pm\Delta_l$ is defined as Equation 5. They optimized value of $\pm\Delta_l$ by minimizing expectation of L2 distance between full precision weights and ternary weights. Instead of using a strictly optimized threshold, we adopt

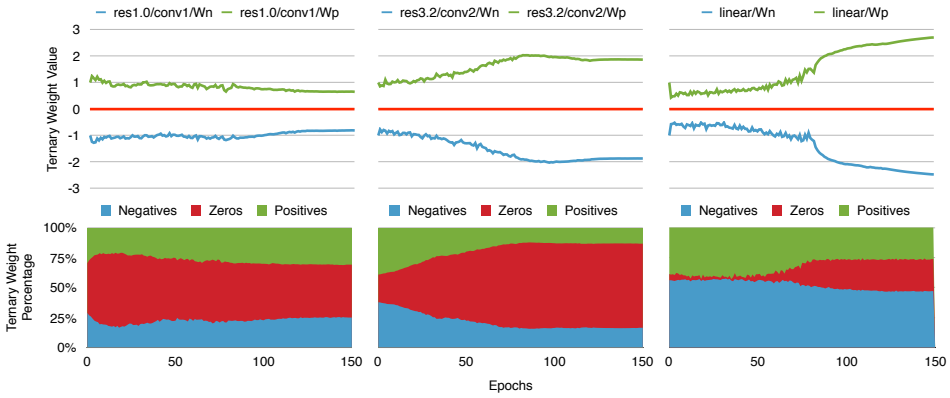


Figure 2: Ternary weights value (above) and distribution (below) with iterations for different layers of ResNet-20 on CIFAR-10.

different heuristics: 1) use the maximum absolute value of the weights as a reference to the layer's threshold and maintain a constant factor $t$ for all layers:

$$\Delta_l = t \times \max(|\tilde{w}|) \tag{9}$$

and 2) maintain a constant sparsity $r$ for all layers throughout training. By adjusting the hyper-parameter $r$ we are able to obtain ternary weight networks with various sparsities. We use the first method and set $t$ to 0.05 in experiments on CIFAR-10 and ImageNet dataset and use the second one to explore a wider range of sparsities in section 5.1.1.

We perform our experiments on CIFAR-10 (Krizhevsky & Hinton, 2009) and ImageNet (Russakovsky et al., 2015). Our network is implemented on both TensorFlow (Abadi & et. al o, 2015) and Caffe (Jia et al., 2014) frameworks.

### 4.3 CIFAR-10

## 5 Experiments

CIFAR-10 is an image classification benchmark containing images of size 32×32RGB pixels in a training set of 50000 and a test set of 10000. ResNet (He et al., 2015) structure is used for our experiments.

We use parameters pre-trained from a full precision ResNet to initialize our model. Learning rate is set to 0.1 at beginning and scaled by 0.1 at epoch 80, 120 and 300. A L2-normalized weight decay

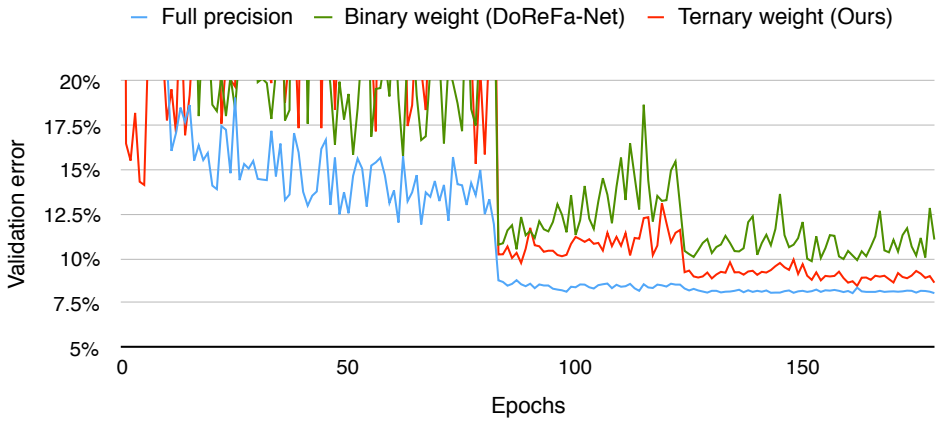


Figure 3: ResNet-20 on CIFAR-10 with different weight precision.

of 0.0002 is used as regularizer. Most of our models converge after 160 epochs. We take a moving average on errors of all epochs to filter off fluctuations when reporting error rate.

We compare our model with the full-precision model and a binary-weight model. We train a a full precision ResNet (He et al., 2016) on CIFAR-10 as the baseline (blue line in Figure 3). We fine-tune the trained baseline network as a 1-32-32 DoReFa-Net where weights are 1 bit and both activations and gradients are 32 bits giving a significant loss of accuracy (green line) . Finally, we fine-tuning the baseline with trained ternary weights (red line). Our model has substantial accuracy improvement over the binary weight model, and our loss of accuracy over the full precision model is small. We also compare our model to Tenary Weight Network (TWN) on ResNet-20. Result shows our model improves the accuracy by $\sim 0.25\%$ on CIFAR-10.

We expand our experiments to ternarize ResNet with 32, 44 and 56 layers. All ternary models are fine-tuned from full precision models. Our results show that we improve the accuracy of ResNet-32, ResNet-44 and ResNet-56 by 0.04%, 0.16% and 0.36% . The deeper the model, the larger the improvement. We conjecture that this is due to ternary weights providing the right model capacity and preventing overfitting for deeper networks.

| Model | Full resolution | Ternary (Ours) | Improvement |
|---|---|---|---|
| ResNet-20 | 8.23 | **8.87** | **-0.64** |
| ResNet-32 | 7.67 | **7.63** | **0.04** |
| ResNet-44 | 7.18 | **7.02** | **0.16** |
| ResNet-56 | 6.80 | **6.44** | **0.36** |

Table 1: Error rates of full-precision and ternary ResNets on Cifar-10

### 5.1 IMAGENET

We further train and evaluate our model on ILSVRC12(Russakovsky et al. (2015)). ILSVRC12 is a 1000-category dataset with over 1.2 million images in training set and 50 thousand images in validation set. Images from ILSVRC12 also have various resolutions. We used a variant of AlexNet(Krizhevsky et al. (2012)) structure by removing dropout layers and add batch normalization(Ioffe & Szegedy, 2015) for all models in our experiments. The same variant is also used in experiments described in the paper of DoReFa-Net.

Our ternary model of AlexNet uses full precision weights for the first convolution layer and the last fully-connected layer. Other layer parameters are all quantized to ternary values. We train our model on ImageNet from scratch using an Adam optimizer (Kingma & Ba (2014)). Minibatch size is set to 128. Learning rate starts at $10^{-4}$ and is scaled by 0.2 at epoch 56 and 64. A L2-normalized weight decay of $5 \times 10^{-6}$ is used as a regularizer. Images are first resized to $256 \times 256$ then randomly cropped to $224 \times 224$ before input. We report both top 1 and top 5 error rate on validation set.

We compare our model to a full precision baseline, 1-32-32 DoReFa-Net and TWN. After around 64 epochs, validation error of our model dropped significantly compared to other low-bit networks as well as the full precision baseline. Finally our model reaches top 1 error rate of 42.5%, while DoReFa-Net gets 46.1% and TWN gets 45.5%. Furthermore, our model still outperforms full precision AlexNet (the batch normalization version, 44.1% according to paper of DoReFa-Net) by 1.6%, and is even better than the best AlexNet results reported (42.8%[1]). The complete results are listed in Table 2.

| Error | Full precision | 1-bit (DoReFa) | 2-bit (TWN) | 2-bit (Ours) |
|---|---|---|---|---|
| Top1 | 42.8% | 46.1% | 45.5% | **42.5%** |
| Top5 | 19.7% | 23.7% | 23.2% | **20.3%** |

Table 2: Top1 and Top5 error rate of AlexNet on ImageNet

[1] https://github.com/BVLC/caffe/wiki/Models-accuracy-on-ImageNet-2012-val

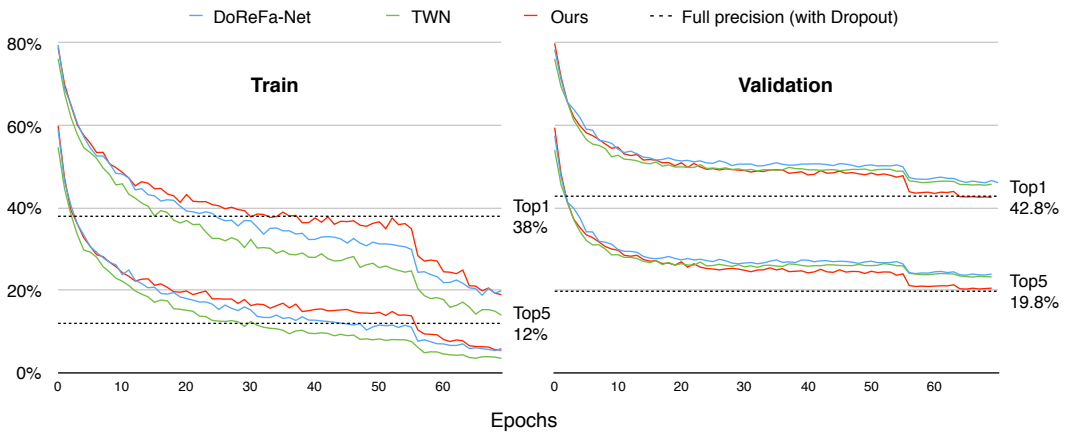


Figure 4: Training and validation accuracy of AlexNet on ImageNet

We draw the process of training in Figure 4, the baseline results of AlexNet are marked with dashed lines. Our ternary model effectively reduces the gap between training and validation performance, which appears to be quite great for DoReFa-Net and TWN. This indicates that adopting trainable $W_l^p$ and $W_l^n$ helps prevent models from overfitting to the training set.

We also report the results of our methods on ResNet-18B in Table 3. The full-precision error rates are obtained from Facebook's implementation. Here we cite Binarized Weight Network(BWN)Rastegari et al. (2016) results with all layers quantized and TWN finetuned based on a full precision network, while we train our TTQ model from scratch. Compared with BWN and TWN, our method obtains a substantial improvement.

| Error | Full precision | 1-bit (BWN) | 2-bit (TWN) | 2-bit (Ours) |
|---|---|---|---|---|
| Top1 | 30.4% | 39.2% | 34.7% | **33.4%** |
| Top5 | 10.8% | 17.0% | 13.8% | **12.8%** |

Table 3: Top1 and Top5 error rate of ResNet-18 on ImageNet

## 6 Discussion

In this section we analyze performance of our model with regard to weight compression and inference speeding up. These two goals are achieved through reducing bit precision and introducing sparsity. We also visualize convolution kernels in quantized convolution layers to find that basic patterns of edge/corner detectors are also well learned from scratch even precision is low.

### 6.1 Spatial and energy efficiency

We save storage for models by $16\times$ by using ternary weights. Although switching from a binary-weight network to a ternary-weight network increases bits per weight, it brings sparsity to the weights, which gives potential to skip the computation on zero weights and achieve higher energy efficiency.

#### 6.1.1 Trade-off between sparsity and accuracy

Figure 5 shows the relationship between sparsity and accuracy. As the sparsity of weights grows from 0 (a pure binary-weight network) to 0.5 (a ternary network with 50% zeros), both the training and validation error decrease. Increasing sparsity beyond 50% reduces the model capacity too far, increasing error. Minimum error occurs with sparsity between 30% and 50%.

We introduce only one hyper-parameter to reduce search space. This hyper-parameter can be either sparsity, or the threshold $t$ w.r.t the max value in Equation 6. We find that using threshold produces better results. This is because fixing the threshold allows the sparsity of each layer to vary (Figure reffig:weights).

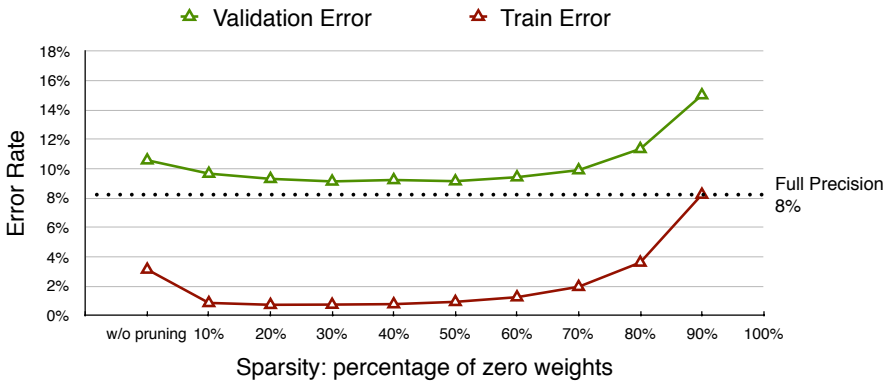


Figure 5: Accuracy v.s. Sparsity on ResNet-20

#### 6.1.2 SPARSITY AND EFFICIENCY OF ALEXNET

We further analyze parameters from our AlexNet model. We calculate layer-wise density (complement of sparsity) as shown in Table 4. Despite we use different $W_l^p$ and $W_l^n$ for each layer, ternary weights can be pre-computed when fetched from memory, thus multiplications during convolution and inner product process are still saved. Compared to Deep Compression, we accelerate inference speed using ternary values and more importantly, we reduce energy consumption of inference by saving memory references and multiplications, while achieving higher accuracy.

We notice that without all quantized layers sharing the same $t$ for Equation 9, our model achieves considerable sparsity in convolution layers where the majority of computations takes place. Therefore we are able to squeeze forward time to less than 30% of full precision networks.

As for spatial compression, by substituting 32-bit weights with 2-bit ternary weights, our model is approximately $16\times$ smaller than original 32-bit AlexNet.

### 6.2 KERNEL VISUALIZATION

We visualize quantized convolution kernels in Figure 6. The left matrix is kernels from the second convolution layer ($5 \times 5$) and the right one is from the third ($3 \times 3$). We pick first 10 input channels and first 10 output channels to display for each layer. Grey, black and white color represent zero, negative and positive weights respectively.

We observe similar filter patterns as full precision AlexNet. Edge and corner detectors of various directions can be found among listed kernels. While these patterns are important for convolution neural networks, the precision of each weight is not. Ternary value filters are capable enough extracting key features after a full precision first convolution layer while saving unnecessary storage.

Furthermore, we find that there are a number of empty filters (all zeros) or filters with single non-zero value in convolution layers. More aggressive pruning can be applied to prune away these redundant kernels to further compress and speed up our model.

| Layer | Full precision | | Pruning (NIPS'15) | | Ours | |
|---|---|---|---|---|---|---|
| | Density | Width | Density | Width | Density | Width |
| conv1 | 100% | 32 bit | 84% | 8 bit | 100% | 32 bit |
| conv2 | 100% | 32 bit | 38% | 8 bit | 23% | 2 bit |
| conv3 | 100% | 32 bit | 35% | 8 bit | 24% | 2 bit |
| conv4 | 100% | 32 bit | 37% | 8 bit | 40% | 2 bit |
| conv5 | 100% | 32 bit | 37% | 8 bit | 43% | 2 bit |
| conv total | 100% | - | 37% | - | 33% | - |
| fc1 | 100% | 32 bit | 9% | 5 bit | 30% | 2 bit |
| fc2 | 100% | 32 bit | 9% | 5 bit | 36% | 2 bit |
| fc3 | 100% | 32 bit | 25% | 5 bit | 100% | 32 bit |
| fc total | 100% | - | 10% | - | 37% | - |
| All total | 100% | - | 11% | - | 37% | - |

Table 4: Alexnet layer-wise sparsity

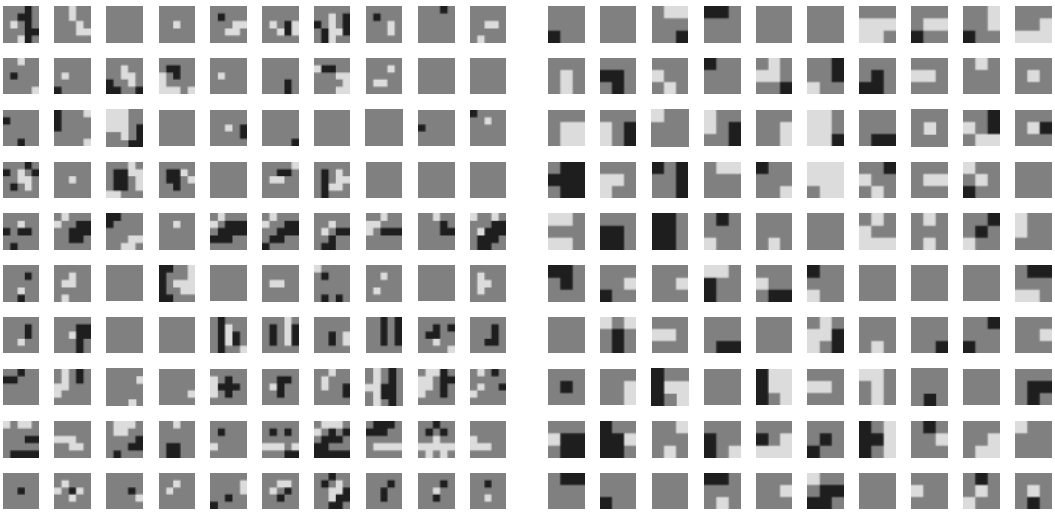

Figure 6: Visualization of kernels from Ternary AlexNet trained from Imagenet.

## 7 CONCLUSION

We introduce a novel neural network quantization method that compresses network weights to ternary values. We introduce two trained scaling coefficients $W_p^l$ and $W_n^l$ for each layer and train these coefficients using back-propagation. During training, the gradients are back-propagated both to the latent full-resolution weights and to the scaling coefficients. We use layer-wise thresholds that are proportional to the maximum absolute values to quantize the weights. When deploying the ternary network, only the ternary weights and scaling coefficients are needed, which reducing parameter size by at least $16\times$. Experiments show that our model reaches or even surpasses the accuracy of full precision models on both CIFAR-10 and ImageNet dataset. On ImageNet we exceed the accuracy of prior ternary networks (TWN) by $3\%$.

## REFERENCES


Martín Abadi and et. al o. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. *arXiv preprint arXiv:1512.02595*, 2015.

Matthieu Courbariaux, Itay Hubara, COM Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training neural networks with weights and activations constrained to+ 1 or-.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pp. 3123–3131, 2015.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR, abs/1510.00149*, 2, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *arXiv preprint arXiv:1603.05027*, 2016.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *arXiv preprint arXiv:1609.07061*, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

Fengfu Li and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. Neural networks with few multiplications. *arXiv preprint arXiv:1510.03009*, 2015.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *arXiv preprint arXiv:1603.05279*, 2016.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.