Project Report
CSCE 633: Machine Learning
XNOR-Nets: Implementation and Analysis

Sameer Kumar Behera
Texas A&M University
College St., TX 77843
sameer-tamu@tamu.edu

Piyush Bhatt
Texas A&M University
College St., TX 77843
pbhatt@tamu.edu

Tanmay Verma
Texas A&M University
College St., TX 77843
tanmay.verma@tamu.edu

Sai Krishna Aditya
Texas A&M University
College St., TX 77843
pskaditya@tamu.edu

Abstract—**The excellent accuracy of Deep Neural Networks (DNN) has made them a popular tool for computer vision and artificial intelligence applications. However, training the models and inference is computation-intensive and have a large memory footprint, which makes them unsuitable to be deployed on resource-limited embedded systems. Moreover, recent research has shown that neural networks carry a lot of redundant information. The paradigm of Binary Neural Networks (BNN) attempts to compress the model and improve the computational efficiency by restraining the weights and inputs to either +1 or -1.**
In this project we implemented a well-known binarization technique that scales well to a large dataset and evaluated its performance on popular image and text datasets. We conclude that the technique gives minimal loss in accuracies across the datasets if we are careful in selecting the layers to binarize.

1. Introduction

The rise in computation power is attributed as one of the primal factors in the popularity of Dense Neural Networks (DNN). A neural network contains a large number of hyper-parameters which are determined by the repeated cycles of training and validations. Moreover for effective model convergence, the entire training data is fed to the network in multiple epochs. Hence, training a DNN is a time-taking exercise and the access to the right hardware is a determinant of the productivity of an individual as a Deep Learning practitioner.

Usually, a DNN is trained on multiple set of expensive and power-hungry Graphics Processing Units (GPUs) [2]. This is a huge limitation with respect to deploying the model on resource-constraint embedded systems. Substantial research is ongoing for optimizing the training and inference on general-purpose CPU or a specialized hardware accelerator.

Binarized Neural Networks (BNN) [3] is one paradigm which constrains the values of weights and activations to a single bit value. This restriction helps in replacing all Multiply and Accumulate(MAC) operations in a layer with two bitwise operations(XNOR and pocount). The inherent computational efficiency in BNN makes it suitable for the deployment on general embedded platforms.

1.1. General Deep Neural Network

The architecture of a Deep Neural Network is described in the Fig 1.

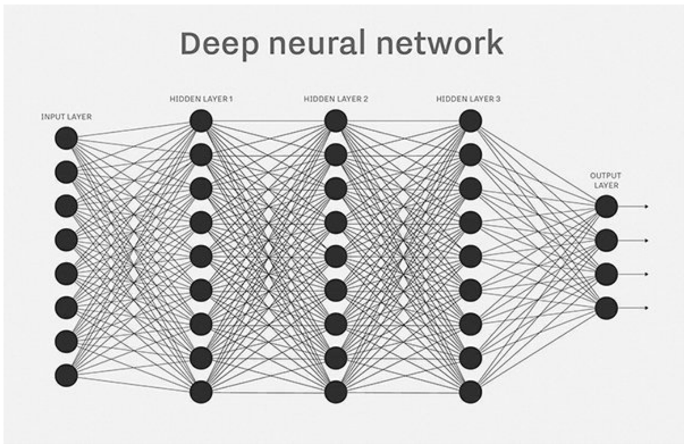


Figure 1. A Deep Neural Network with three hidden layers

There can be multiple hidden layers in the network. Each layer on the other hand can posses multiple nodes. The training of a DNN occurs in three steps:

- Forward Pass (Similar to Inference)
- Backpropagating Error
- Update the Weights

As the first step the training batch is fed to the network with the current set of parameters. Next the error in the ouptput of the network is backpropagated to the previous layers. Using the chain rule it can be shown that we can obtain the gradients in these previous layers. These gradients are then used to update the weights to minimize the loss.

Both the training a DNN and inference on the model performs the computation described in Figure 2 on each node.

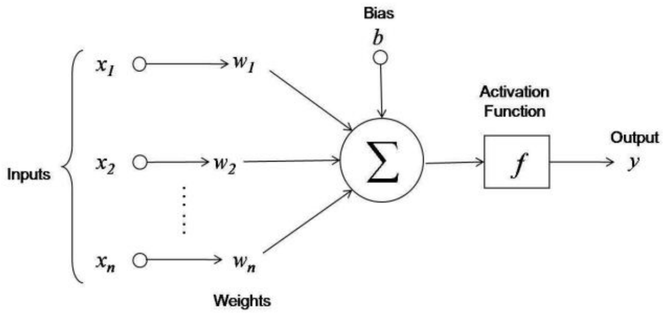


Figure 2. Computations at each node

The total number of multiply-accumulate operations for a single inference will be:

$$N = \sum_{i=1}^{Layers} \sum_{j=1}^{Nodes_i} (Fan\_in)_{ij} \tag{1}$$

where $Layers$ is the total number of layers in the network, $Nodes_i$ is the number of nodes in the i^{th}, and Fan_in_{ij} is the fan_in of j^{th} node in i^{th} Layer.

The Table 1 provides the number of MAC operations in popular Convolution Neural Networks (CNN) during a single pass.

Model	Le--Net-5	Alex--Net	VGG-16	Goog--LeNet v1	Res--Net-50
Weights	60 K	61 M	138 M	7 M	25.5 M
MACs	341 K	724 M	15.5 G	1.43 G	3.9 G

TABLE 1. WEIGHT AND MAC NUMBER OF POPULAR CNNS [17]

General-purpose processors execute these MAC operations mostly sequentially, which leads to a low efficiency. On the other hand, GPUs exploit the parallelism in the network computation to give higher throughput. However, GPUs also suffer from a high energy cost which makes them unsuitable to be used on a device with limited power budget.

1.2. Binarized Neural Networks

Binarized Neural Networks restrains the weights and inputs of layers to {-1,1}. This restraint helps in representing 32 distinct weights in an otherwise single weight in 32 bits. There have been multiple efforts on training and inferencing on a Binary Neural Network in the past with very few successes reported. The next sections describes the progress in this paradigm in a more detailed manner.

1.3. Advantages

The Figure 3 describe how Nine Arithmetic Multiplies and Six Arithmetic Additions] can be replaced with Three bitwise XNOR and Three popcounts in a 3 $\times$ 3 matrix multiplication with a 3 $\times$ 1 vector producing essentially the same exact results.

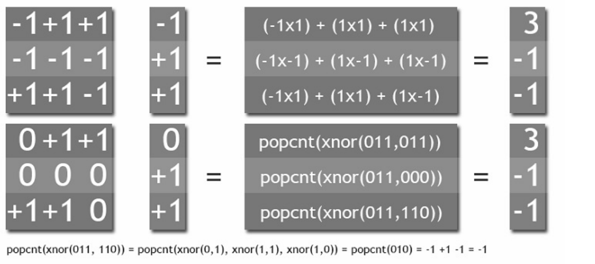


Figure 3. Arithmetic Multiplication Versus Bitwise Multiplication

This demonstrates following key benefits of using Binarized Neural Networks against full-precision neural network.

- Smaller-Sized Model (1-bit required instead of 32-bits)
- Faster computations (Compute expensive arithmetic operations replaced with quick bitwise operations)
- Energy-efficient computations

1.4. Related Work

Efficient computational structures have been studied since a long time. It was shown that power consumption can be reduced by using 10 bits and 12 bits to represent gradients and store weights for state-of-art neural networks [4] as Arithmetic operations consume significantly less power than floating-point operations.

As a natural progression, there had been attempts to use a neural network with a single bit to represent weights. Previous attempts destroyed the Neural Network until Kim et al [9] successfully trained a BNN on MNIST dataset with negligible loss in accuracy but significant gain in power efficiency. The weights, inputs, activation, biases and output of the model were represented as binary values {-1,1}.

At the same time Courbariaux et al [4] used weight binarization as regularizer to obtaining accuracy comparable to state-of-the-art models on MNIST, CIFAR-10 and SHVN [5]. However, as the inputs and activations were real-valued, the model completely to derive the complete efficiency.

Courbariaux et al. in [3] constrained the activations and weigths to [-1,1]. An online training method is proposed which can run efficiently on a resource constrained system. Although, the model performed competitively on MNIST, CIFAR-10 and SHVN, the method performed poorly with large data set like ImageNet.

Rastegari et al. argues that the training method described in BinConnect and BinaryNet can not scale to large data set [16]. XNOR-NET with binary weights and inputs are proposed. The proposed model is shown to significantly perform better than BinaryNet [3].

Outside network binarization, the ideas like deep compression [7] and specialized computing platforms have also been proposed for efficient computation on DNN.

For this project we selected the binarization technique introduced in [16] to binarize classifiers on MNIST, CIFAR-10 and IMDB dataset.

2. XNOR-Net Mechanism

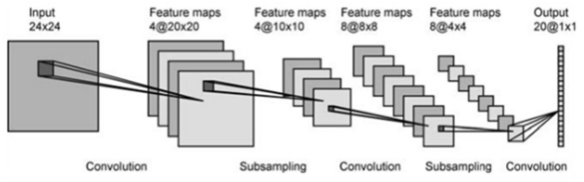


Figure 4. A Typical CNN Architecture

Let's represent the input to a convolutional layer with **I** $\in \mathrm{R}^{c \times w_i \times h_{in}}$, where ($c$, w_i, h_{in}) represents channels, width and height respectively. The real-valued weight kernel be presented by **W** $\in \mathrm{R}^{c \times w \times h}$, where $w \leq \mathrm{w}_{in}$ and $h \leq \mathrm{h}_{in}$.

The weight matrix **W** is strides across the layer input. At each step the matrix product between weight filter and underlying inputs is calculated. Assuming at one such step, **X** $\in \mathrm{R}^{c \times w \times h}$ represent the underlying receptive field.

Under this assumption, we need to represent **W** and **X** matrices with binary filter **B** and α, **H** and β respectively, where **B** and **H** $\in \{+1,-1\}^{c \times w \times h}$, and α and β are scaling factors $\in \mathrm{R}^{+}$ such that the following constraint is satisfied

$$X^T W \approx \beta H^T \alpha B \tag{2}$$

The optimization problem becomes:

$$\alpha^\star, \mathbf{B}^\star, \beta^\star, \mathbf{B}^\star = \underset{\alpha, \mathbf{B}, \beta, \mathbf{B}}{\operatorname{argmin}} f(x) \tag{3}$$

where

$$f(x) = ||\mathbf{X} \odot \mathbf{W} - \beta \alpha H \odot B|| \tag{4}$$

where $\odot$ represents an element-wise product.

Solving this optimization problem we will get [16] the following results:

$$\alpha^\star = \frac{1}{n} ||\mathbf{W}||_{l1} \tag{5}$$

$$\mathbf{B}^\star = sign(\mathbf{W}) \tag{6}$$

$$\beta^\star = \frac{1}{n} ||\mathbf{X}||_{l1} \tag{7}$$

$$\mathbf{H}^\star = sign(\mathbf{X}) \tag{8}$$

The Figure 5 describes the binarized convolution operation in detail. It also explains a trick, using which we can reduce the redundant operations involved in calculating βs.

3. XNOR-Net Implementation

In this section we will discuss the implementation details, decisions and the reasoning behind our decisions. First we will go through the tools used and why they were preferred over other available alternatives. Then, we will discuss the files and directory structure of the git repository. Finally we will discuss different functional units in our project and a brief introduction of their implementation details.

3.1. Tools Used

Pytorch [14] is an open source machine learning framework, which makes the implementation of neural networks very readable and easy. Moreover, we chose it over Tensorflow because of Pytorch's advantage when it comes to the ease of experimentation and readability of model specifications. For visualizing the changes in our model as the training progresses with the increasing number of epochs, we have used a library TensorboardX [8].

3.2. Files and Directory Structure

Directory structure for experiment with each dataset is exactly same and the functional units are also implemented in the similar fashion to maintain consistensy across multiple models. Within each dataset directory, we decided the file and directory structure based on the functionality served by each file. The models directory contains python file in which the neural network is defined. It contains all the functionality of how the model structure is, what are the hyper-parameters, definition of the new binary convolution and how the new activation function should work. The trained model is also saved in the models directory so it can be reused whenever needed. One directory up from models directory, the main.py and util.py files are placed. Utility file contains functionality which can directly be used by the CNN model, functionality like binarization, clamping convolution parameters, saving the parameters, how to update the binary weights, etc. Finally the main.py file contains the code which prepares the input which it then feds into the neural network defined/saved inside the models directory. Within this file, we are also evaluating our model by recording the accuracy and visualizing the process using TensorboardX.

3.3. Functional units

Following are the functional units and the details of their implementation:

3.3.1. BinOp class in utility.py. This class contains utility functions which can be used to binarize the layers. The function names are very descriptive of the tasks they are responsible for - meancenterConvParams(), clampConvParams(), save_params(), binarizeConvParams(), restore() and updateBinaryGradWeight().

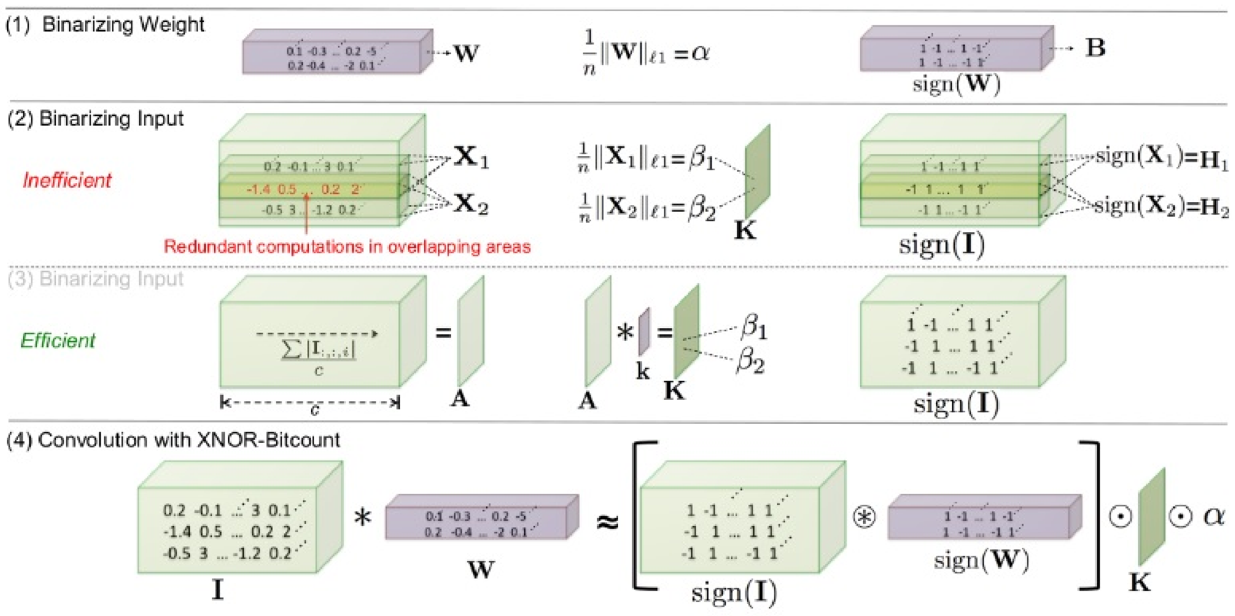

(1) Binarizing Weight
W
$\frac{1}{n}\|\mathbf{W}\|_{\ell 1} = \alpha$
B
sign(W)
(2) Binarizing Input
Inefficient
X_1
X_2
$\frac{1}{n}\|\mathbf{X}_1\|_{\ell 1} = \beta_1$
$\frac{1}{n}\|\mathbf{X}_2\|_{\ell 1} = \beta_2$
K
sign(X_1)=H_1
sign(X_2)=H_2
sign(I)
Redundant computations in overlapping areas
(3) Binarizing Input
Efficient
A
k
K
β_1
β_2
sign(I)
(4) Convolution with XNOR-Bitcount
I
W
sign(I)
sign(W)
K
α

Figure 5. The Binarized Convolution in XNOR-Net

3.3.2. binary_accuracy(preds, y) in main.py. This function takes predictions and ground truths as input parameters and returns the accuracy per batch. For example, if you get 8 predictions correct out of 10, it will return 0.8.

3.3.3. train(model, iterator, optimizer, criterion) in main.py. This functions defines the training loop and backpropagation of our neural model. Moreover, it also talkes care of optimizations like setting the gradient to zero before starting the training loop, so the parameters are updated correctly. It also calculates the loss according to the criterion which is passed as an input parameter to this function. This returns average epoch loss and average epoch accuracy.

3.3.4. evaluate(model, iterator, criterion) in main.py. This contains the definition of evaluation mode. It iterates over the test cases in batches, feeds those as input to the neural mode and returns the average accuracy and loss results.

3.3.5. BinActive class in models directory. This class is the equivalent of activation in a binarized model. This binarizes the input activations and calculates the mean across channel dimension. It has two functions forward() and backward(), which are used depending on the direction of propagation.

3.3.6. BinConv2d class in models directory. This class is the equivalent of Conv2d in a binarized model. All the functionality provided are similar to Conv2d except the weights and inputs of this layer will be binarized. It is a misnomer to call it BinConv2D as essentially it is a unit in itself with shuffled operations as described in Figure 6. The operations are shuffled to keep the accuracy.

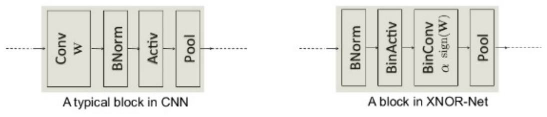


Figure 6. Modified Sequence of Operations for XNOR-Net

3.3.7. Full precision model. There is class which defines the full precision model on the dataset under consideration. This class contains the definition of each layer and the hyperparameters each one takes. Moreover, this class contains a function for forward propagation in which the ordering of the layers and dropout is defined.

3.3.8. Binarized model. There is class which defines the binarized model on the dataset under consideration. This class contains the definition of each layer and the hyperparameters each one takes. Moreover, this class contains a function for forward propagation in which the ordering of the layers and dropout is defined. This class is exactly similar to the full precision class except in the part where the layers are defined. All the convolution layers which are binarized use BinConv2d rather than Conv2d. Rest of the code is exactly same.

4. Experimentation

The following sections detail on our experimental setup and binarization decisions on the three different datasets.

First, we demonstrate that our implementation is correctly binarizing the weights and inputs for binarized convolutional layer in MNIST. Next, we observe the accuracies deploying the same technique on image classification on CIFAR-10, which is a much larger dataset. In the end, we extend the XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks [16] by evaluating the binarization technique for Sentiment Analysis on IMDB Movie Reviews dataset.

4.1. Hand-Written Digit Classification on MNIST

4.1.1. Dataset. The MNIST database [12] is a collection of images that contain hand-written digits. It has been derived by taking a collection of samples from the NIST database. The database contains 60,000 training images and 10,000 testing images. The images are centered in a 28x28 image. In some cases though, this field was extended to 32x32 with background pixels. So far, the best system of convolutional neural networks on the MNIST database was able to achieve an error rate of 0.23% [1] This system has used full precision layers throughout its model for the computation of weights and biases and updating the model.

4.1.2. Model. One of the more prevalent models used for the classification of the MNIST database is the Lenet-5 model. [11]. This multilayer neural network uses a gradient based learning technique and has two convolutional units and one fully connected layer. Each convolutional unit comprises of a convolutional layer, a ReLU and a max pooling unit. This relatively less complex convolutional neural network is able to achieve an accuracy of 99.34% on the test set. Since all of its layers are full-precision, all of the weights and biases each consume 32 bits in the memory space.

The model that we proposed is similar to Lenet-5 model except that the second convolutional layer is not a full precision layer. The first convolutional layer and the last fully connected layer are not binarized. The output from the first convolutional layer (the first activations) is passed through a binarization function. Once the activations are binarized they are passed as inputs to the second convolutional layer. The weights in this convolutional layer are binarized as well.

4.1.3. Training. The MNIST dataset has been trained with the model that has been proposed. It has been observed that after 30 epochs, the maximum validation accuracy achieved was 99.24% and minimum cross-entropy loss was found to be 0.036 respectively as shown in Figure 7. The histogram in Figure 8 shows the variation in biases and weights of the Binarized Second Convolutional Layer with each epoch till model convergence.

4.1.4. Visual Validation. The weights and the inputs in the Binarized Second Convolutional Layer is shown in Figure 9 and Figure 10 respectively. Both the filters and the inputs are either complete white or complete black. This, on top of the training histograms shows that our model is behaving as expected.

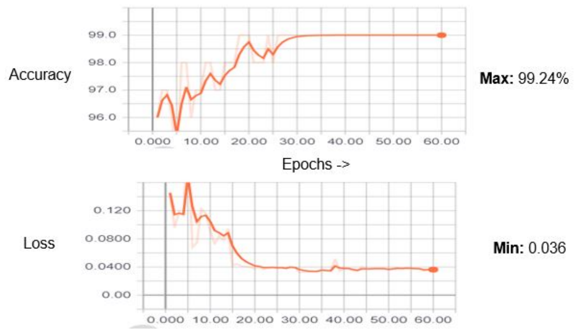


Figure 7. Accuracy and Cross-Entropy Loss Results for Binarized MNIST Model

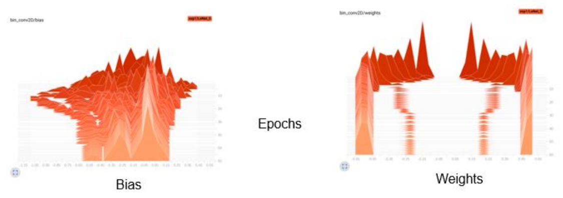


Figure 8. Histogram of Binarized Convolutional Layer in MNIST Model

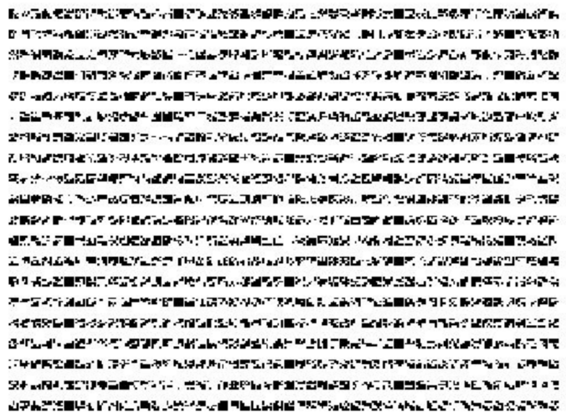


Figure 9. Filters of the Binarized Convolutional Layer in LeNet-5

4.2. Image Classification on CIFAR-10

4.2.1. Dataset. The CIFAR-10 dataset [10] consists of 60000 32x32 colour images in 10 mutually exclusive classes, with 6000 images per class. There are 50000 training images and 10000 test images. For this dataset, we apply the same global contrast normalization and ZCA whitening as was used by Goodfellow et al. in the maxout network

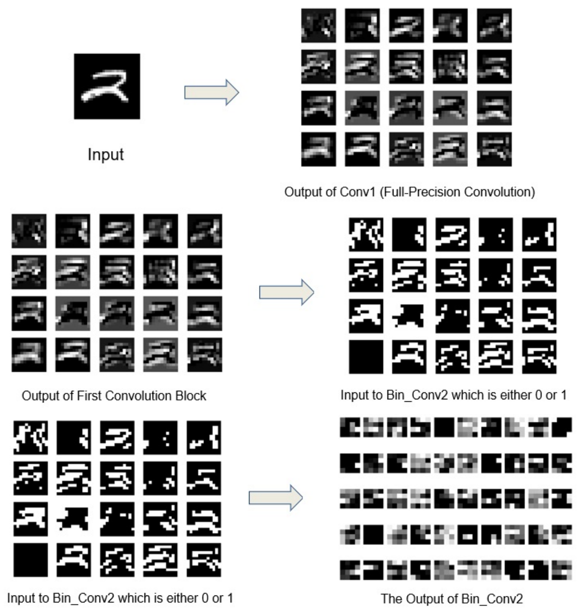

Input
Output of Conv1 (Full-Precision Convolution)
Output of First Convolution Block
Input to Bin_Conv2 which is either 0 or 1
Input to Bin_Conv2 which is either 0 or 1
The Output of Bin_Conv2

Figure 10. Activations and inputs of Binarized Convultional Layers in LeNet-5

[6]. We use the last 10,000 images of the training set as validation data.

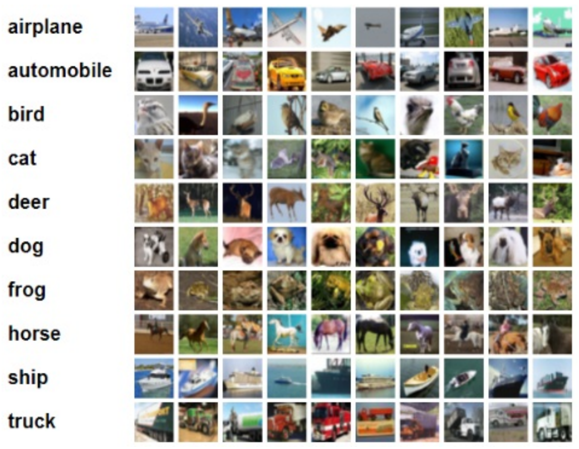


Figure 11. CIFAR-10 Dataset: 10 Random Images of Each Class

4.2.2. Model. A utilize a novel deep network structure called Network In Network (NIN) [13] which enhances model discriminability for local patches within the receptive field. The conventional convolutional layer uses linear filters followed by a non-linear activation function to scan the input. Instead, the NIN structure uses micro neural networks with more complex structures to abstract the data within the receptive field. The micro neural network gets instantiated with a multilayer perceptron, which is a potent function approximator. The feature maps are obtained by sliding the micro networks over the input in a similar manner as CNN; they are then fed into the next layer. Deep NIN can be implemented by stacking multiple layers of the above described structure. Enhanced local modeling via the micro network utilizes global average pooling over feature maps in the classification layer, which is easier to interpret and less prone to overfitting than traditional fully connected layers.

The Binarized model which we have proposed is exactly similar to the NIN model except that all the stacked layers apart from the First Convolutional Layer and the Last Fully Connected Layer have been binarized to BinConv2d class rather than Conv2d in which the weights and biases for these stacked layers get binarized. This makes the inner functioning of those stacked layers quite different from the NIN model even if the overall structure remains same.

4.2.3. Training. As per the Binarized NIN Model, the stacked layers, i.e. Layers 3, 4, 6, 7, 8, 10 and 11 were converted to the BinConv2d class from the Conv2d class in which all the weights and biases of these layers got binarized. However, all the existing hyper-parameters, i.e. local receptive field size, weight decay and dropout, etc. were kept the same as in the original NIN model. The model was trained until the maximum accuracy achieved did not improve with 25 subsequent epochs. The maximum validation accuracy achieved was 85.02% and the minimum cross-entropy loss was found to be 0.595 as can be seen in Figure 12.

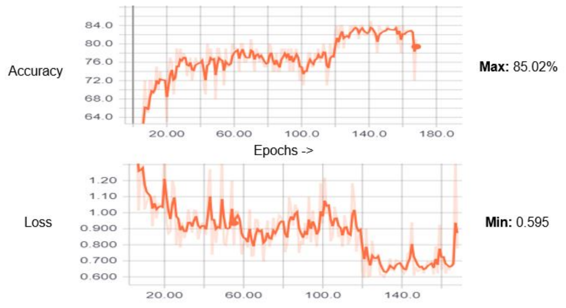


Figure 12. Accuracy and Cross-Entropy Loss Results for Binarize NIN Model

The histograms showing the variations in the bias and weights of the binarized stacked layers, i.e. Layers 3, 4, 6, 7, 8, 10 and 11 with every successive epoch can be seen in Figure 13. As can be seen in all of the histograms, the biases and weights of all the stacked layers fluctuate a lot with every epoch until the model converges and a conformity can be seen in them. Also, we can see that in most of the binarized stacked layers, the weights converge sooner than the biases.

4.3. Text Classification on IMDB

4.3.1. Dataset. IMDB movie review dataset is one of the benchmark datasets used in sentiment analysis in text data.

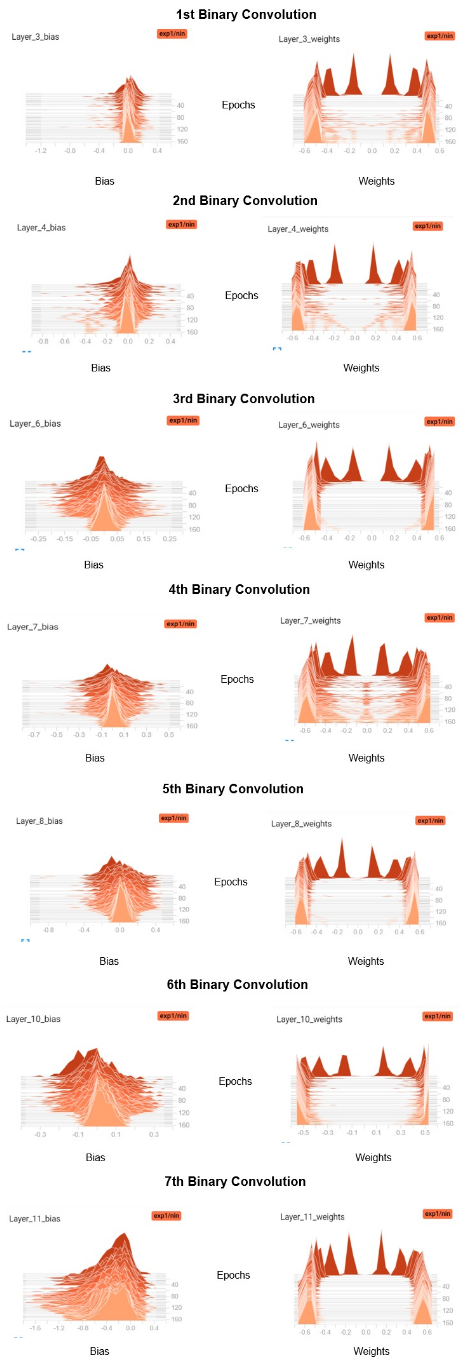
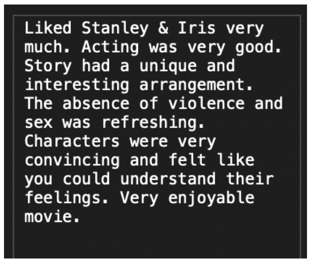


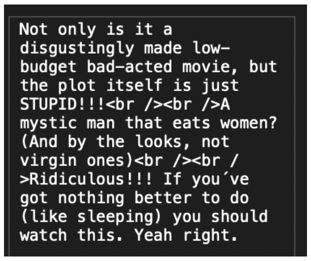
Figure 13. Histograms of Binarized Stacked Layers in CIFAR-10 Model

It consist of fifty thousand reviews, labelled with one of the two binary sentiments, position or negative. For unsupervised learning, it contains additional fifty thousand unlabelled reviews. In the dataset, there are at most 30 reviews for a single movie. This is done to avoid over-representation of a specific model, thus preventing the dataset from being skewed. Also, most of the reviews of a single movie are highly correlated and having a lot of reviews from a single movie wouldn't make the dataset rich. The training and testing sets are also provided by the dataset and the two sets are disjoint on the basis of movies i.e. the the training and testing sets do not have reviews for common movies, if a movie review belongs to the training set, then all the reviews for that movie will also be in training set. Examples of a positive and a negative review are shown in Figure 14 and Figure 15.



```
Liked Stanley & Iris very
much. Acting was very good.
Story had a unique and
interesting arrangement.
The absence of violence and
sex was refreshing.
Characters were very
convincing and felt like
you could understand their
feelings. Very enjoyable
movie.
```

Figure 14. Sample Positive Review



```
Not only is it a
disgustingly made low-
budget bad-acted movie, but
the plot itself is just
STUPID!!!<br /><br />A
mystic man that eats women?
(And by the looks, not
virgin ones)<br /><br /
>Ridiculous!!! If you´ve
got nothing better to do
(like sleeping) you should
watch this. Yeah right.
```

Figure 15. Sample Negative Review

4.3.2. Model. Full Precision Model (Figure 4.3.2) Our full precision model uses the n-gram model for training. The first layer of the model is convolution layer with the size of the filter equal to the number of dimensions in embedding time the n in n-gram. We have used the pre-trained word vectors with 100 dimensions and the three n-gram models we have consider 3, 4 and 5 lines at a time respectively. For each of the n-gram we are using 100 filters. So, the first convolution layer will have 100 filters each of size 3x100, 4x100 and 5x100. The output of the first layer will be 300 one dimensional vectors. Next layer is max pooling layer which is applied on each of these 300 one dimensional vectors, which gives us 300 values. These 300 values are appended together to form a 1x300 vector which is fed as input to the linear neural layer.

Binarized model (Figure 4.3.2) This model is exactly similar to the full precision model except that the layers with filter size 4x100 and 5x100 are BinConv2d rather than Conv2d and the weights for these two layers are also binarized. This makes the inner functioning of those layers different than the full precision model but the overall structure remains same.

4.3.3. Pre-Trained Word Vectors. In the absence of a large supervised training dataset, using pre-trained word

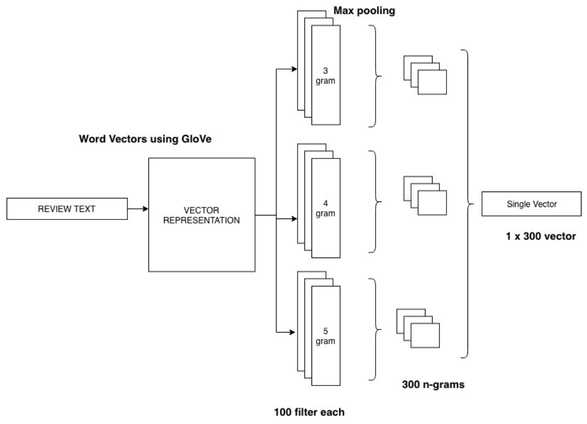


Figure 16. IMDB Full Precision Model

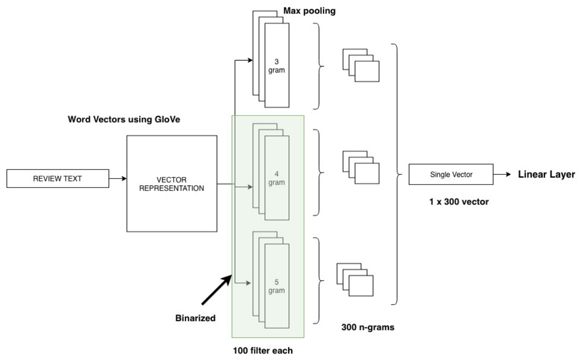


Figure 17. IMDB Binarized model

vectors, which have been obtained from some unsupervised neural language model, for the representation of words in the dataset is a very common practice in natural language processing. We have used Global Vectors for Word Representation (GloVe) [15] which are provided for public used by the Standford NLP community. The pre-trained word vectors that we have used has 6 billion tokens, 100 dimensions and 400k vocabulary words.

4.3.4. Training. The IMDB dataset has been trained with the above discussed n-gram model. It was observed that the weights got fixed after first few initial epochs and after that the model was getting trained only by changing change in bias 18. After 5 epochs the loss started to increase implying that the model starts to overfit after 5 epochs. The maximum accuracy achieved was 92.6% in the binarized model, which only dropped by 4% from the full precision model.

5. Observations

Figure 20 describes the final results. The models trained using the binarizing techniques fared well when compared to the full precision models. For MNIST data set, the accuracy for the binarized neural network was almost at par with the full-precision model. The LeNet5 model displayed a test accuracy of 99.34% whereas our binarized model showed a

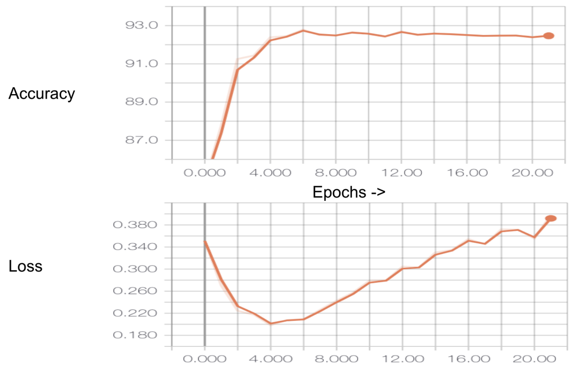


Figure 18. Accuracy and Cross-Entropy Loss Results for Binarized IMDB Model

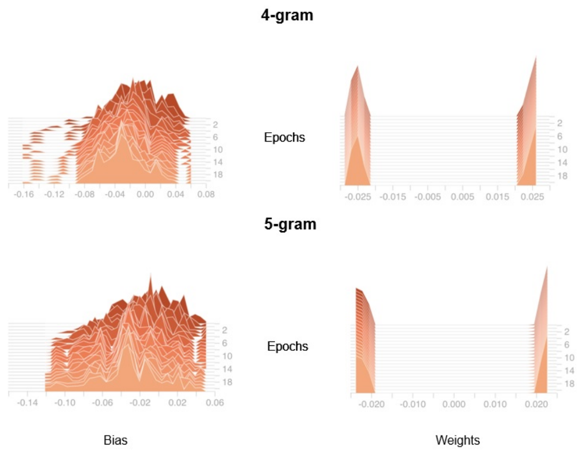


Figure 19. Histogram of Binarized Convolution Layer in IMDB Model

99.24% on the accuracy. CIFAR-10 and IMDB being huge datasets require relatively larger and complex models to train on. Binarizing such large number of weights and activations resulted in slightly larger difference in accuracies. The NIN model produced a test accuracy of 89.67% on the CIFAR-10 whereas our model had an accuracy of 85.02%. Similarly the test accuracy on the IMDB dataset for the full precision model is about 87.68, whereas that for the binarized model has been about 84.79%. Considering the x32 memory savings that the binarization of the weights and activations bring, this slight decrease in accuracy seems reasonable.

6. Team Breakdown

Even with the simplicity offered by PyTorch framework, training the neural network was a very time-consuming task. It took each one of us equal efforts to understand the paper [16] and training tricks. All of us worked hard to implement the binarization on MNIST.

After which, Sai Krishna Aditya started working on the LeNet Visualization. Tanmay and Sameer worked on extending the technique on larger image datasets. After getting CIFAR-10 working, Sameer started to work with

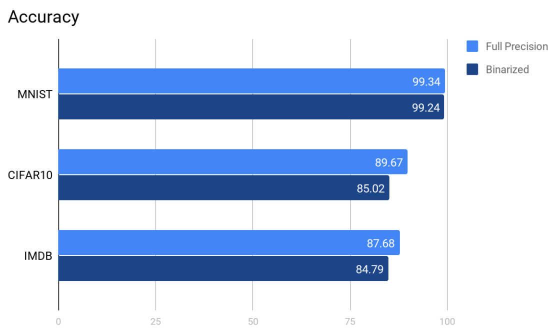


Figure 20. Test Accuracies on Different Datasets

AlexNet for ImageNet and Tanmay began looking on the ways to execute bitwise operations to compare actual speed-up. Unfortunately, we couldn't go all the way through our last two efforts. And finally, Piyush worked on the sentiment analysis using the same binarization technique and related problems that come with while handling textual data.

Having said all of that, we continuously relied upon each other for help and guidance. It was a great learning experience and we learnt new techniques in a very short period.

7. Conclusion

As per our observations, Binarized Neural Network does seem to hold a lot of promise in the future. Even though the model suffers some accuracy loss, yet the overall computational demand is significantly reduced. In addition, the reduced power requirement and model-size makes the model amiable to be deployed on the edge devices which generally have low power and memory budgets. The presented XNOR-net is a simple technique that can have tremendous impact. It will reduce the dependency of critical devices on continuos internet connection. Moreover, as the inference and training can be performed on-device, it also improves the data privacy.

The bottleneck of this technique involves storing the full-precision weights while training to generate gradients and update weights. The scientific community is actively working on the improving the binarization techniques without hurting the accuracies.

We started with a motivation to make computations on Deep Neural Networks more efficient. Next, we showed how bitwise operations can replace arithmetic operations. Discussing the popular binarization technique, we analyzed and implemented one which scales the best to large datasets. We showed its performance on MNIST, CIFAR-10 and IMDB datasets. However, there is a lot of work that could have been done. Next section details the same.

8. Future Work

In this work we analyzed the performance of binarization on a small image dataset (MNIST) and a comparatively larger image dataset (CIFAR-10). We would like to see the performance on a larger image dataset where the benefit of binarization would be a lot higher in terms of speed than on any of the image datasets that we have used. One such dataset is the ImageNet dataset that looks like an obvious next step to us. Similary, we would also like to test this model on a larger text based dataset like i2b2 in which the data to handle is very large compared to the current IMDB dataset, which only has 50k labelled reviews.
Another thing that would be the next step in improving the speed and accuracy of the model would be to optimize by using actual Bitwise Operations which is a very high effort task because for most of the mathematical operations we are directly using a very high level library - Pytorch and this change would mean messing around with the library for numerical computation. Furthermore,we can optimize how the binary values are saved because as of now even though the valued are binary in nature they are being stored similar to a 32 bit numerical value.

Acknowledgments

We would like to thank Dr. Bobak Mortazavi for his supervision and proposal reviews. His lectures were very helpful in developing firm understanding of the concepts and their application in this project. We would also like to extend our gratitude to Xiaohan Chen who has been a great help in clarifying the doubts we had related to the concepts of neural networks and convolutional layers.

References

[1] Dan Ciresan, Ueli Meier, and Jrgen Schmidhuber. Multi-column deep neural networks for image classification. In *IN PROCEEDINGS OF THE 25TH IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR 2012*, pages 3642–3649, 2012.

[2] Adam Coates, Brody Huval, Tao Wang, David J. Wu, and Andrew Y. Ng. Deep learning with cots hpc systems.

[3] Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016.

[4] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Low precision arithmetic for deep learning. *CoRR*, abs/1412.7024, 2014.

[5] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3123–3131. Curran Associates, Inc., 2015.

[6] Ian J. Goodfellow, David Warde-Farle, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. 2013.

[7] 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, 2015.

[8] Tzu-Wei Huang. lanpa/tensorboardx, Nov 2018.

[9] Minje Kim and Paris Smaragdis. Bitwise neural networks. *CoRR*, abs/1601.06071, 2016.

[10] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[11] Yann Lecun, Lon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[12] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[13] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[14] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. Pytorch, 2017.

[15] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *In EMNLP*, 2014.

[16] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *CoRR*, abs/1603.05279, 2016.

[17] V. Sze, Y. Chen, T. Yang, and J. S. Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, Dec 2017.