Binary Neural Networks

Jay Shah
jgshah1@asu.edu

Kunal Suthar
ksuthar1@asu.edu

Tirth Shah
tpshah2@asu.edu

Tithi Gupta
tvgupta@asu.edu

***Abstract*— Through this project, we attempt to train Binary Neural Networks(BNNs) which are essentially Neural Networks with binary weights and activations i.e. +1 and -1. BNNs boost the performance in terms of memory usage and computation complexity during the forward propagation. This is achieved by replacing most arithmetic operations with bitwise boolean operations. We compare the performance of a traditional non-binary network with a network designed using two binarization functions: Deterministic and Stochastic. We also compare the train times, performance, memory usage, and speed for the binary and non-binary versions on the fashion-MNIST dataset[3]. We achieve significant improvements with binarized neural networks.**

I. INTRODUCTION

Neural Nets are inspired from the basic structure of a human brain which consists of neurons and interconnections between them, also known as synapses. Generally, neural networks are made of input layers consisting of input neurons, one or more hidden layers and output layers made of output neurons. The weights associated between these neurons that form the link define the relation in-between these nodes. These neurons multiples the initial value received from the input neurons by some value of weights aggregates those values and then adjusts this value using a bias followed by an activation function which normalizes the resulting output.

Although one of the challenges faced for using traditional deep neural networks is, they require a large number of memory accesses for the required computation. Hence this poses a common limitation for the use of such networks on embedded designs. However, one of the methods with which we can reduce the need for such memory and computation requirements is to reduce the size of parameters and activations during the training time [Ullrich et al. (2017)]. And hence, binary neural network essentially have binarised values for its inputs, outputs, and weights, i.e +1 or -1. A much obvious benefit of this in terms of storage and computation is due to a reduction of a 32-bit number to a single-bit value. [Matthieu C et al.] clearly states that replacing the floating point values of inputs, outputs and weights accumulate operations of a matrix to simple additions and subtractions.

Dropout is a commonly used technique to solve the issue of regularization[1] which helps in mitigating the challenge of overfitting the model. Although, the approach suggested by [2] and the one implemented here, binarizes the values of both the weights and activations to zero or one which essentially is an alternative to the traditional approach used in the Dropout method of randomly setting the values of activations to be zero.

Here in this project,

1) We implement a binary neural network along the lines of [2] using binary weights and activations and generate comparisons for both deterministic and stochastic binarization mechanisms.
2) We will be comparing the implementations of two different architectures on Python framework and show that one can train BNNs on fashion-MNIST while achieving almost the same state-of-art results. We will be comparing the train times, performance, memory usage, speed for binary and non-binary implementations of both the architectures.

II. METHOD

Deep Neural networks are used to perform a wide range of tasks in the field of speech recognition, natural language processing, computer vision, etc. The key to the progress has been the processing power of GPUs. The ability to train larger models and more data has achieved concrete success. Today, most of the researchers are working on designing the new algorithms which will help to train a large number of samples and enormous data on low power devices[6].

A. Binarization

In this section, we detail the binarization function, show how it can be used to compute the parameter's gradients, and how to backpropagate the gradients through the BNN. This methodology is based on the works mentioned in section IV of this paper.

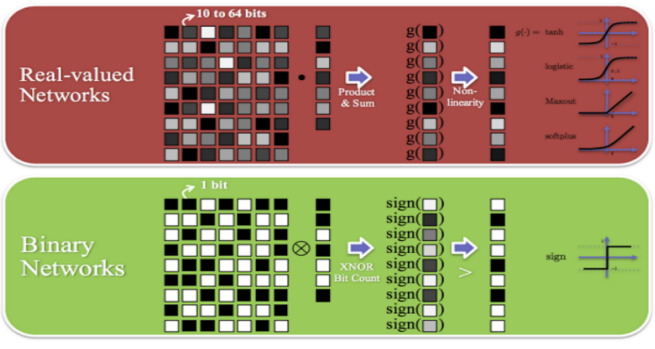

Fig. 1. Real vs Binary Architecture

As shown in Figure 1 [1], Deep Neural Networks consists of matrix multiplications and convolutions. The main operation

[1]Image-Source: https://software.intel.com/en-us/articles/accelerating-neural-networks-with-binary-arithmetic

of deep learning algorithms consists of multiplication and accumulation. Artificial neurons are multiply-accumulator which involves computation of their weighted sums of inputs. While training binary neural networks, we constrain both the weights and the activations to either +1 or -1. These two values have tremendous impact from a hardware perspective. We replace multiply-accumulator operations with simple arithmetic (addition and subtraction) operations. The gain that we extract is in terms of both expense and energy utilization. In order to perform this, we need to transform the real-valued variables into the binary values. We use two different binarization functions and compare both in terms of performance and error rate.

In this section we give a detailed discussion about two different binarization functions, considering how to discretize two values, how to train the samples within two values and how to perform inference.

1) **Deterministic Binarization:** The binarization operation transforms the real-valued weights to two possible values. The deterministic binarization is based on the sign function. It is given by:

$$x^b = Sign(x) = \begin{cases} +1 & \text{if } x \geq 0. \\ -1 & \text{otherwise.} \end{cases} \tag{1}$$

where x^b is the binarized variable and x is the real-valued variable. This deterministic operation is characterized by normalizing the discretization over all the input weights of the hidden unit to compensate for the loss of information.

2) **Stochastic Binarization:** The alternative approach that gives a more correct averaging process is stochastic binarization. It is a modification of the logistic sigmoid function which outputs 1 with the probability p associated with it and -1 otherwise.

$$x^b = Sign(x) = \begin{cases} +1 & \text{with probability } p = \sigma(x). \\ -1 & \text{with probability } 1 - p. \end{cases} \tag{2}$$

where $\sigma(x)$ is also termed as hard sigmoid function and is given by:

$$\sigma(x) = clip(\frac{(x+1)}{2}, 0, 1) = max(0, min(1, \frac{(x+1)}{2})) \tag{3}$$

Generally, we tend to opt hard sigmoid function as the binarization function for its better performance and less error rate, but it is hard to implement because the hardware undergoes quantizing and generates random bits, which is computationally expensive [2].

B. Gradient Propagation Through Discretization

The sign function has derivative as zero for almost all the values, which is also true for stochastic function. This makes them unsuitable for back-propagation. (Bengio, 2013)[9] found through his experiments that "straight through estimator" mentioned in (Hinton's, 2012) [10] lecture provided fastest training.

We also consider the effect of saturation in the straight through estimator and use deterministic(instead of stochastic) sampling of the bit. The use of the straight through estimator is shown in training algorithm in subsection E. We are required to use the variants for BatchNorm and ADAM.

C. Batch Normalization

The transformation from original weights to binary weights will result into some information loss. Batch Normalization is used to reduce the impact of overall weight scaling and it also increases the training speed. The values in the hidden layer keep changing all the time. When normalizing helps to speed up for the input layers, we shall also apply it for hidden layers which can speed up the performance by 10 times. Normalization also helps to regularize the model and reduces overfitting[2]. It allows each layer to work independently. Batch Normalization does lot of multiplications and then involves division by the standard deviation. Thus, the number of scaling calculations involved is too large. The SGD undoes this and demoralizes two weights for each activation in such a way that there is no accuracy loss.

D. ADAM

The alternative approach to reduce the impact of weight scaling factor is ADAM learning rule. It is a first-order gradient based optimization for stochastic function [13]. The important factor about ADAM is the magnitudes of parameter updates does not depend on how the gradient is re scaled.

E. Training Algorithm

For training the Binary Neural Network (BNN), we tweak the generic Neural Network training algorithm by adding binarization steps. We pass the initial input to the network for performing forward propagation. In that, the parameters/weights of the network are binarized first. Then we perform Batch Normalization on the activated output. Then the activated outputs are subjected to Binarization. After the forward propagation, we perform back propagation by propagating the binarized gradient and calculating the binarized gradients for the previous layers. After we have completed a step of forward and backward propagation, we update the parameters/weights and the biases. The pseudo code for the algorithm is as follows[2]:

Input: A minibatch of inputs a_0

Forward Propagation:

for k=1 to L **do**
W_k^b = Binarize(W_k)
s_k=a_{k-1}^b W_k^b
a_k=BatchNorm(s_k,θ_k)

[2]Source: https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c

if k $<$ L **then**
a_k^b=Binarize(a_k)
end if
end for

Backward Propagation:

We compute g_{aL}= $\partial C/\partial a_L$ knowing a_L and a^*.
for k=L to 1 **do**
if k $<$ L **then**
g_{ak} = g_{abk} o $1_{|ak|<=1}$
end if
(g_{sk},$g_{\theta k}$)= BackBatchNorm(g_{ak},s_k,θ_k)
g_{abk-1}= g_{sk} W_k^b
g_{Wbk}= g_{sk}^T a_{k-1}^b
end for

Updating the parameters and biases

for k= 1 to L **do**
θ_k^{t+1}= Update(θ_k,η,$g_{\theta k}$)
W_k^{t+1}=Clip(Update(W_k,$\gamma_k\eta$,g_{Wbk}),-1,1)
η^{t+1}= $\lambda\eta$
end for

C : Cost Function
λ : The learning rate decay factor
L : The number of layers
BatchNorm(): performs batch-normalization on the activations
W_k^b: Binarized weight of the kth layer
W: Weights
θ: BatchNorm parameters
η: Learning rate

III. EXPERIMENTS AND RESULTS

We perform the above training algorithm on the fashion-MNIST dataset which has 60000 training images and 10000 testing images of dimensions 28 * 28 with 10 classes. We further split the 60000 training images into 50000 training images and 10000 validation images. We train the BNN in mini-batches of 5000 as the whole batch of 50000 images does not fit into the memory. We train the network over 1000 epochs and over 500 epochs in some experiments. We perform validation and compute the costs while training on the validation dataset. For computing the training time every epoch, we note the starting time of every epoch and subtract it from the ending time of the epoch. For measuring the memory consumption, we use the *psutil* library. It enables us to gauge the total memory consumption of the running training process.

A. Architecture (784,128,10) (Epochs: 1000, LR: 0.2)

1) **Train Times for Binary and Non-binary versions**:

Binary	Non-binary
5073 secs	7084 secs

2) **Performance for Binary and Non-binary versions**:

Accuracy Type	Binary	Non-binary
Training	68.70%	83.80%
Testing	67.90%	82.50%

3) **Memory usage for Binary and Non-binary versions**:

Binary	Non-binary
869.63 MB	1016.32 MB

4) **Speed for Binary and Non-binary versions**:

Binary	Non-binary
5 secs/epoch	7 secs/epoch

5) **Error/Cost**: Figure 2 and 3

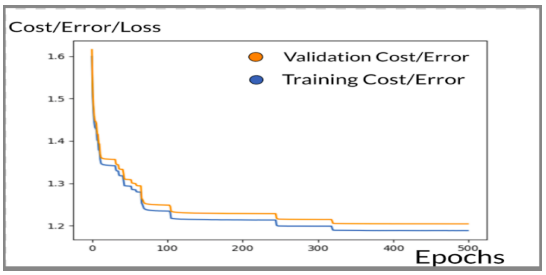

Fig. 2. Error vs Epoch for binarized neural network

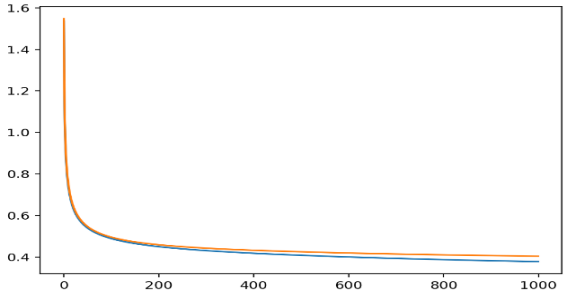

Fig. 3. Error vs Epoch plot for Non-binarized network

B. Architecture (784,1024,10) (Epochs: 1000, LR: 0.2)

1) **Train Times for Binary and Non-binary versions**:

Binary	Non-binary
16293 secs	28496 secs

2) **Performance for Binary and Non-binary versions**:

Accuracy Type	Binary	Non-binary
Training	87.10%	90.60%
Testing	85.10%	87.20%

3) **Memory usage for Binary and Non-binary versions**:

Binary	Non-binary
1364.09 MB	1867.32 MB

4) **Speed for Binary and Non-binary versions**:

Binary	Non-binary
16.3 secs/epoch	28.5 secs/epoch

5) **Error/Cost**: Figure 4 and 5

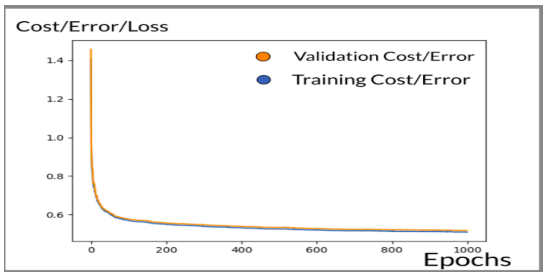

Fig. 4. Error vs Epoch for Binarized neural network

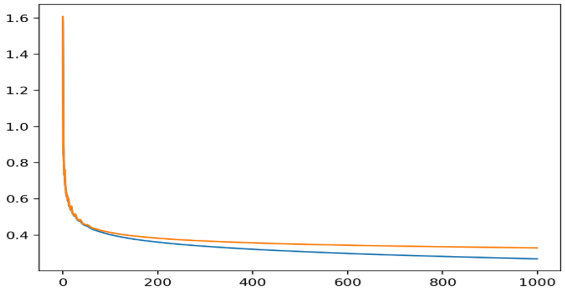

Fig. 5. Error vs Epoch plot for Non-binarized network

NOTE: The above results for the binarized neural network were obtained using deterministic binarization function.

C. Comparisons of Stochastic & Deterministic binarization

For architecture(784, 1024, 10) Epochs: 1000, LR: 0.2:

1) **Train Time vs Epoch plot for Stochastic and Deterministic functions**:

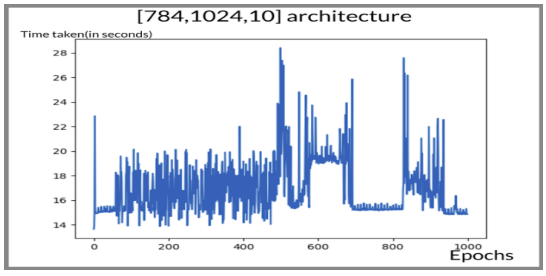

Fig. 6. Train time vs Epoch for deterministic function

2) **Performance for Stochastic and Deterministic functions**:

Accuracy Type	Stochastic	Deterministic
Training	84.00%	87.10%
Testing	82.60%	85.10%

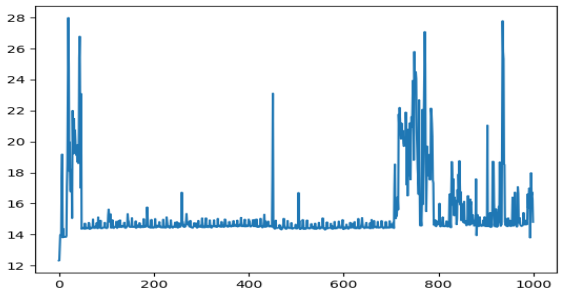

Fig. 7. Train time vs Epoch for Stochastic function

3) **Memory usage for Stochastic and Deterministic versions**:

Stochastic	Deterministic
923.57 MB	1364.09 MB

4) **Error/Cost**: Figure 8 and 9

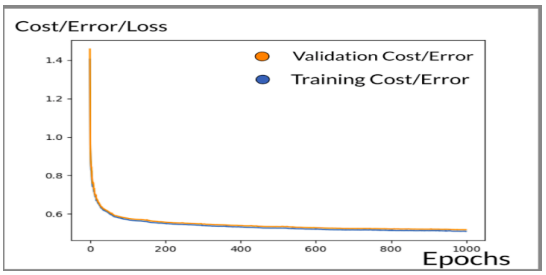

Fig. 8. Error vs Epoch for Deterministic function

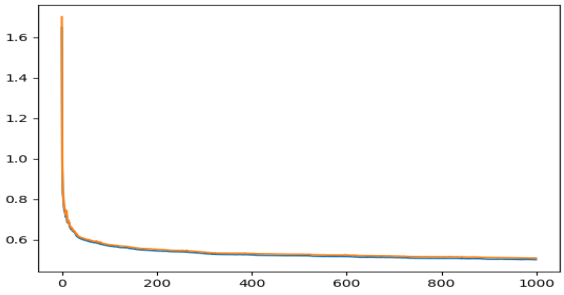

Fig. 9. Error vs Epoch for Stochastic function

D. Comparisons of Stochastic & Deterministic binarization

For architecture(784, 128, 10) Epoch: 500, LR: 0.2

1) **Train Time vs Epoch plot for Stochastic and Deterministic functions**: Fig. 10 and 11

2) **Performance for Stochastic and Deterministic functions**:

Accuracy Type	Stochastic	Deterministic
Training	69.20%	68.20%
Testing	68.30%	67.30%

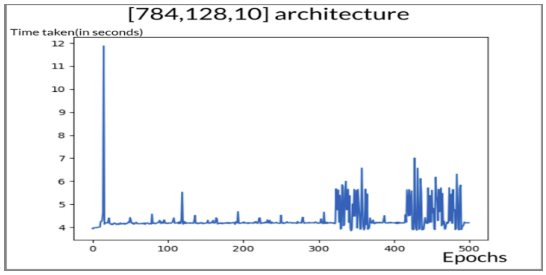

Fig. 10. Train time vs Epoch for deterministic function

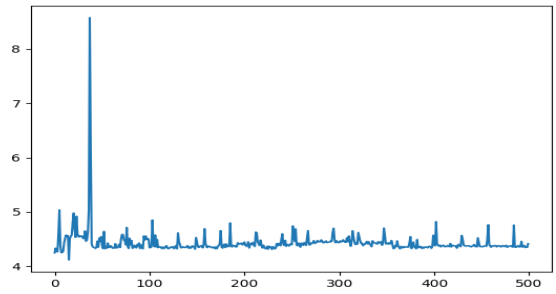

Fig. 11. Train time vs Epoch for Stochastic function

3) **Memory usage for Stochastic and Deterministic versions**:

Stochastic	Deterministic
869.63 MB	931.56 MB

4) **Error/Cost**: Figure 12 and 13

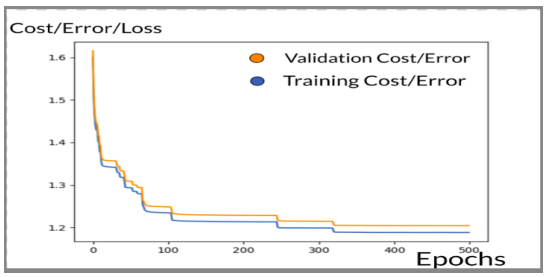

Fig. 12. Error vs Epoch for Deterministic function

IV. RELATED WORK

A. Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to +1 or –1 [2]

In this paper, the authors have introduced Binary Neural Networks which are basically Deep Neural Networks which calculates binary weights and activations at run-time and parameter gradients at train-time.

Two sets of experiments were conducted on two different frameworks, Torch7 and Theano to train BNNs and achieve state-of-the-art results.In the Torch7 experiments, the activations are stochastically binarized at train-time, whereas in Theano experiments they are deterministically

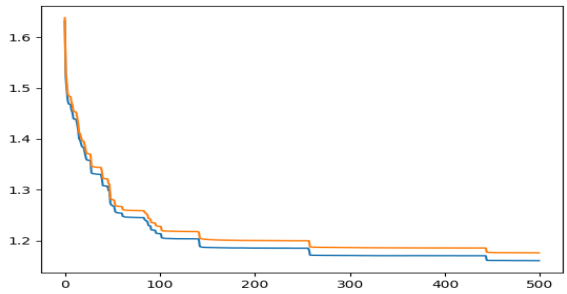

Fig. 13. Error vs Epoch for Stochastic function

binarized. In forward pass, most of the arithmetic operations are replaced with bit-wise operations reducing memory size and accesses(more power efficiency). Also, a binary matrix multiplication GPU kernel is designed which makes it possible to run the MNIST MLP 7 times faster than unoptimized GPU kernel.

B. Learning to Train a Binary Neural Network[12]

The authors of this paper have presented a work which is very helpful to understand the working of binary neural networks, especially the training process.

They have presented a framework where they have systematically evaluated different network architectures and hyperparameters to provide useful insights on how to train a binary neural network. Results of their work show that increasing the number of connections between layers of a binary neural network can improve its accuracy in a more efficient way than simply adding more weights.

C. BinaryConnect: Training Deep Neural Networks with binary weights during propagations[7]

In this paper, the authors have proposed a novel way to train a DNN with binary weights for the forward and backward propagations as well as retaining precision of the stored weights in which gradients are accumulated. This model is termed as BinaryConnect.

Their results show that it is possible to train DNNs on the permutation invariant data-sets and achieve nearly state-of-the-art results. This method affect specialized hardware implementations by removing about 2/3rd of the multiplications causing a speed-up of about a factor of 3 during training. If the deterministic version of BinaryConnect is used, the impact on testing time is more major as it gets rid of the multiplication altogether reducing the memory requirement by a factor of 16. This affects the memory computation bandwidth ratio and the size of models.

V. CONCLUSIONS AND FUTURE WORKS

We have implemented here a Binary Neural Network (BNN) that makes use of binarized weights and activations during the train time, as well as during the run-time. At the training time, these binarized values are used for computing gradients. This framework helps to train a network on

machine with relatively fewer resources. We have used here two methodologies for binarizing the values of the network, deterministic and stochastic and compared both of them to two architectures of traditional non-binary implementation of a neural network on the Fashion-MNIST dataset. With both the binary versions we were not only able to achieve nearly state-of-art results but recorded a significant reduction in memory usage and total time taken during training the network. This was possible because the 32-bit arithmetic operation were replaced by bitwise operations.

In terms of total time taken for training, the deterministic binary version performed 28% and 42% better respectively for both the architectures. For the memory usage, the deterministic version performed 14% and 26% better respectively. We also compared the stochastic version of binary implementation which again was able to reach state-of-art performance but also able to slightly improve the deterministic version's memory usage and time taken for training. These results suggest that with minor trade-offs in performance, with binary neural networks, we can improve the memory and time consumption of training a neural network which is a key challenge in traditional neural networks.

Future works include implementing ADAM learning rate and binarization of gradient parameters here in order to further improve the training times and extending these implementations to other benchmark datasets.

VI. DIVISION OF WORK

We divided the work for this project into two major parts among the four team members.

1) Both the Binary implementations of the Neural Network, namely Deterministic and Stochastic were implemented by Jay Shah and Kunal Suthar respectively.
2) And the Non-binary Implementation of the Neural Network in order to compare the state-of-art results was done by Tirth Shah and Tithi Gupta.
3) We all collaboratively made this report and slides for the Project Presentation and were able to achieve consensus on the concluding results mentioned above.

VII. SELF-PEER EVALUATION TABLE

Jay Shah 20	Kunal Suthar(Myself) 20
Tirth Shah 17	Tithi Gupta 16

VIII. ACKNOWLEDGMENTS

We would like to thank and appreciate the efforts of Professor Hemanth Kumar Demakethepalli Venkateswara and Teaching Assistant Kevin Ding who took time to define all the topics clearly and guide us during the course explaining the basics of the topic which helped us immensely in our choice and implementation of the project.

REFERENCES

[1] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15, 19291958 (2014).

[2] M. Courbariaux and Y. Bengio, Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1, CoRR, vol. abs/1602.02830, 2016.

[3] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[4] Collobert,Ronan,Kavukcuoglu,Koray,andFarabet,Cle ment. Torch7: A Matlab-like environment for machine learning. In BigLearn, NIPS Workshop, 2011.

[5] Bergstra, James, Breuleux, Olivier, Bastien, Fre de ric, Lam- blin, Pascal, Pascanu, Razvan, Desjardins, Guillaume, Turian, Joseph, Warde-Farley, David, and Bengio, Yoshua. Theano: a CPU and GPU math expression compiler. In Proceedings of the Python for Scientific Computing Conference (SciPy), June 2010. Oral Presentation.

[6] Bastien, Fre de ric, Lamblin, Pascal, Pascanu, Razvan, Bergstra, James, Goodfellow, Ian J., Bergeron, Arnaud, Bouchard, Nico- las, and Bengio, Yoshua. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[7] Courbariaux, Matthieu, Bengio, Yoshua, and David, Jean-Pierre. Binaryconnect: Training deep neural networks with binary weights during propagations. ArXiv e-prints, abs/1511.00363, November 2015.

[8] Srivastava, Nitish. Improving neural networks with dropout. Masters thesis, U. Toronto, 2013.

[9] Bengio, Yoshua. Estimating or propagating gradients through stochastic neurons. Technical Report arXiv:1305.2982, Uni- versite de Montreal, 2013.

[10] Hinton, Geoffrey. Neural networks for machine learning. Cours- era, video lectures, 2012.

[11] Footnote-2:https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c

[12] J. Bethge, H. Yang, C. Bartz, and C. Meinel, Learning to train a binary neural network, 2018

[13] Kingma, Diederik and Ba, Jimmy. Adam: A method for stochas- tic optimization. arXiv preprint arXiv:1412.6980, 2014.