Project Report

CS 597: Reading and Special Problems

Variable Precision Deep Learning (float32/float16)

Prepared By:
Nilesh Jorwar (A20405042)
Ashish Ryot (A20405230)

Submitted to:
Prof. Ioan Raicu

Table of Contents

INTRODUCTION AND OBJECTIVES

We know that human visual system is one of the wonders of world. Humans can effortlessly recognize the handwritten digits. In each hemisphere of human's brain, we have primary visual cortex that has 140 million neurons and has lot of connections between them. Humans have such series of cortices which does more complex image processing progressively. The difficulty of visual pattern recognition becomes apparent if we attempt to write computer program to recognize handwritten digits. Neural networks approach this recognition problem by taking large number of handwritten digits as training samples and develop model or system that learns from those training samples. With the increase in training samples, model can learn more about handwriting to improve the accuracy. Hence, we proposed the C++ and Python deep learning neural network models to recognize handwritten digits and tested these models over other datasets as well. Models accept input image pixels as low precision data types to check the performance and speedup of CPU when ran using different node settings at hidden layer. We also built multithreaded model to perform the matrix multiplication at hidden and output layer. The use of AVX instructions in model for matrix multiplication gave us 10X speedup when applied O3 flags in C++ model. Our work shows the comparable results on all the three models we developed and greatly improved accuracy over the earlier proposed models.

NEURAL NETWORK ARCHITECTURE

The most common and simple structure for neural networks is the three layer structure with full interconnection. The input nodes of this architecture simple feed in the information or more generally are passive, that is, doing nothing but relaying information from their single input to their multiple output. In our case this input layer consists of 784 neurons, one neuron each for the individual pixels of a 28x28 grayscale image. In comparison the nodes/neurons of the hidden and output layer are active, which means that they modify the input signal and output the new modified data. The action of this type of neural network is determined by the weights applied on the hidden and output nodes.

The values from the input layer are sent to all of the hidden layer nodes. This is called a fully interconnected structure. The values entering a hidden layer nodes are multiplied by weights, a set of predetermined numbers stored in the program. The weighted inputs are then added to produce a single number. Before leaving the node, this number is passed through a nonlinear mathematical function called a sigmoid that limits the node's output. That is, the input to the sigmoid is a value between -∞ and +∞, while its output can only be between 0 and 1.

CPP Model and TensorFlow Model Architecture

MNIST dataset is divided in train and test sets where train set is provided as input to the Hidden Layer with weights and biases computed progressively at each batch per epoch. The output of the hidden layer (i.e. sigmoid activation function) is given as input to output

layer along with weights and biases. The activation function (i.e. sigmoid) is computed for 10 class labels. The loss function (cross entropy) is calculated at each epoch and is decreased iteratively while increasing accuracy.

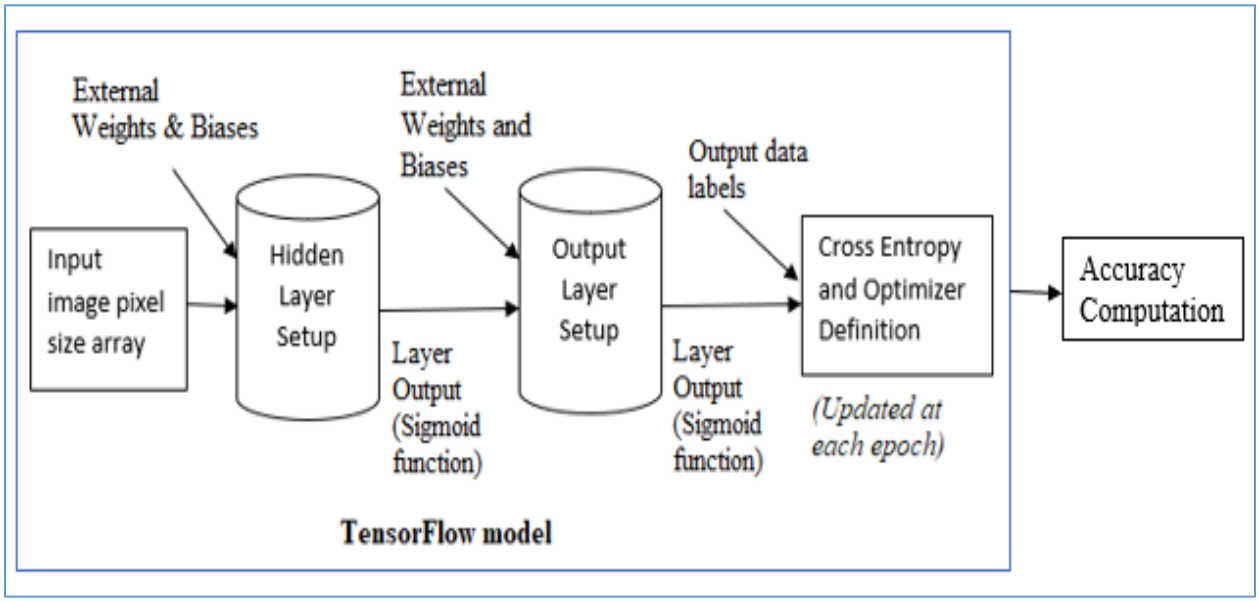
CURRENT WORK

CPP Model-

We have implemented variable precision deep neural network model in C++. Two versions of CPP model are built, multithreaded for float16 and AVX instructions for matrix multiplication for float32 data types. MNIST and Fashion datasets are tested using this model and compared with TensorFlow and Keras model built in Python.

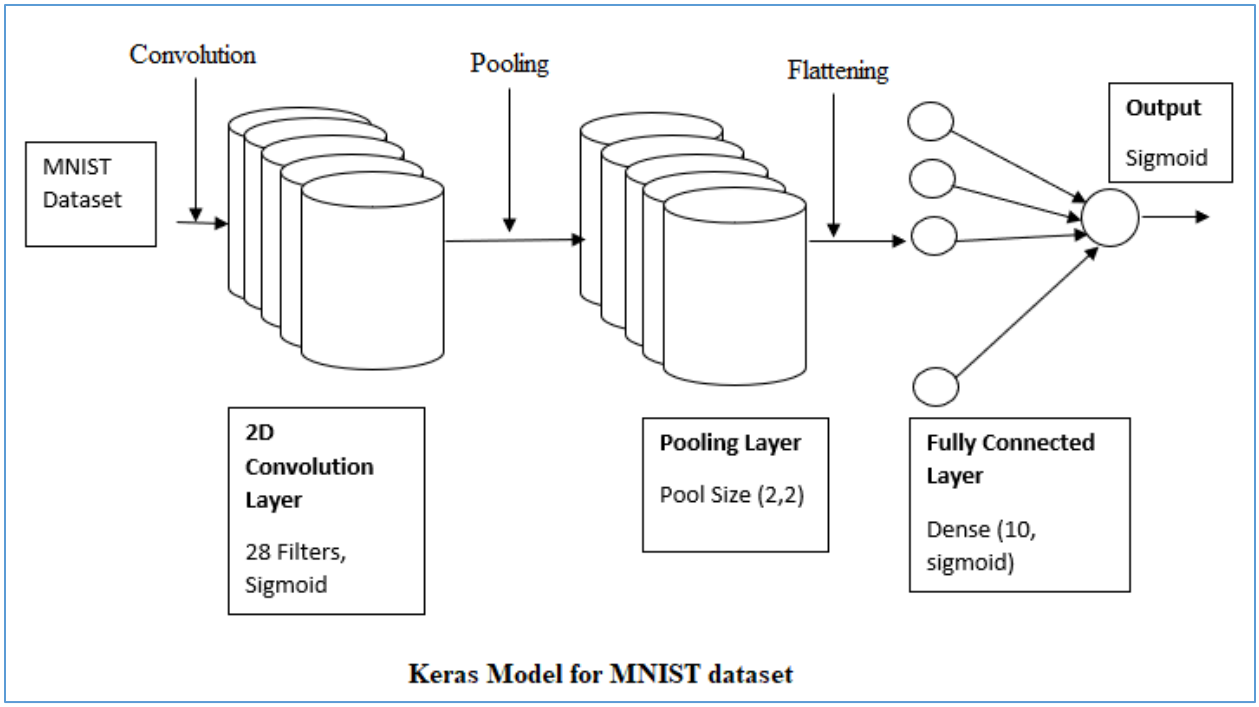
TensorFlow Model-

Python implementation of TensorFlow model deals with float32 data types and tested using MNIST dataset. This model requires to have initial setup of hidden layer and output layer based on input image size array along with loss function. This setup is run inside the TensorFlow session where loss function is updated at each epoch and accuracy is computed at the end of all epochs.



Keras Model-

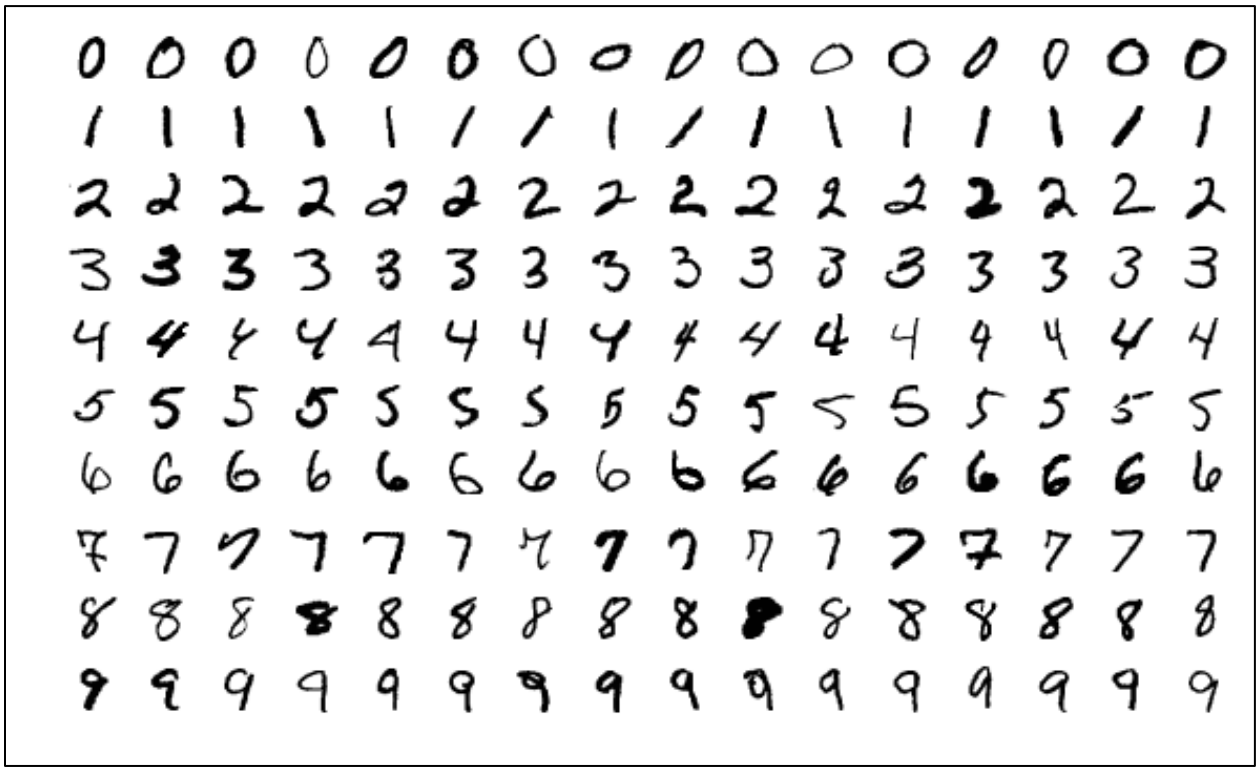
Keras model of Python is Convolution Neural Network which builds the model using three layers viz., convolution, pooling and flattening. The model built is compiled using various parameters like loss function and accuracy type and fit to trained data. The trained model is tested over the test data of MNIST dataset and evaluated for accuracy.



Keras Model for MNIST dataset

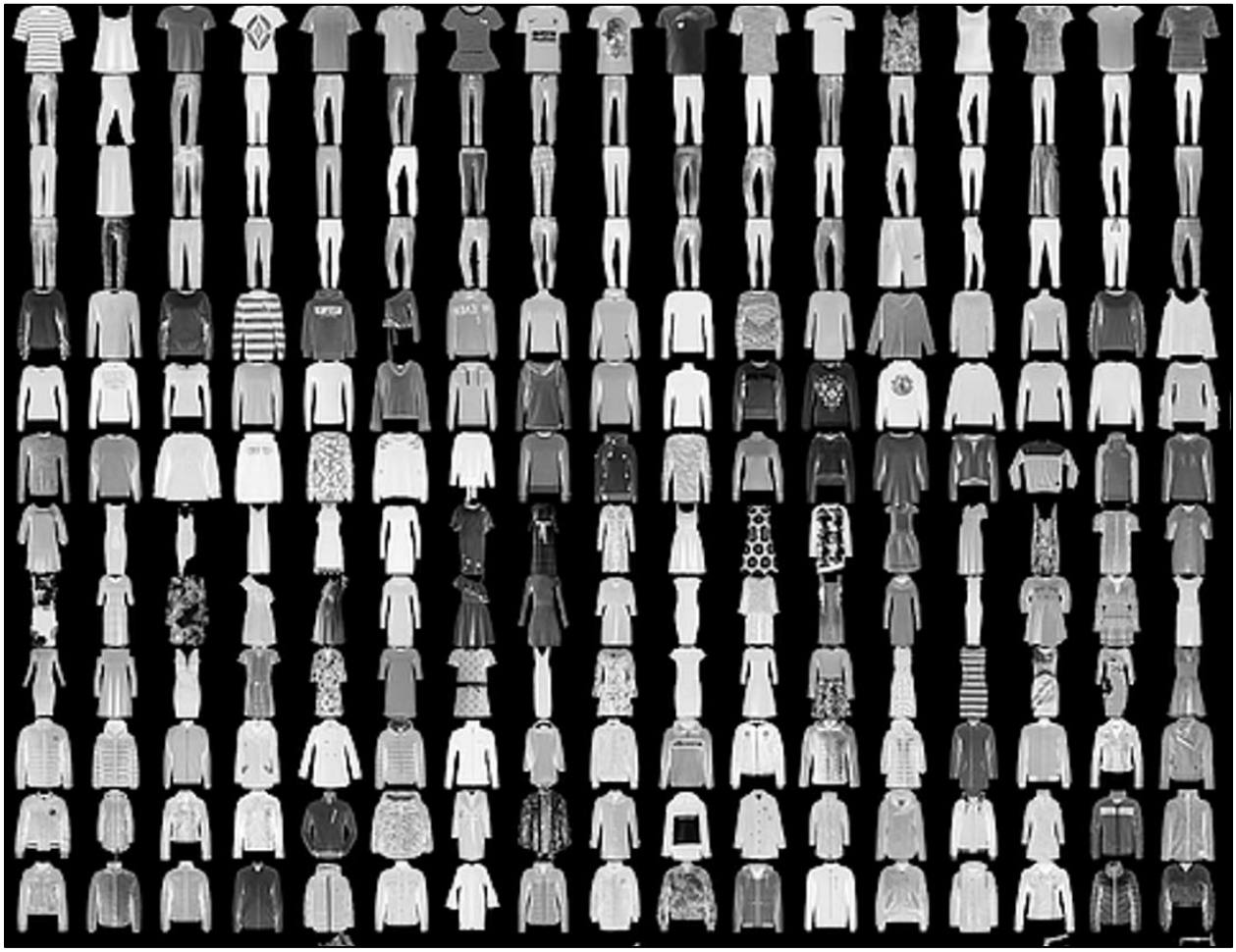
MNIST Handwritten Digits:

This dataset contains 60,000 training images with their corresponding classes and 10,000 test images. The size of every image is 28x28 pixels.



Fashion dataset:

This dataset consists of 50,000 training images with their respective labels and 10,000 testing images. This size of each image in this dataset is also 28x28 pixels.



EVALUATION

The experiments were conducted on Float32 and Float16 data types. The experiments were also conducted on TensorFlow and Keras to compare and contrast the results of our model with them. Given below is the confusion matrix from one of the experiments conducted using Float16 (half) data type:

EVALUATION MATRIX

1033	0	32	22	1	11	37	2	40	2
0	1044	59	7	0	0	6	5	14	0
25	17	881	11	16	2	31	14	24	11
8	4	36	881	0	33	6	19	13	10
1	1	11	2	860	3	27	5	7	65
31	4	15	96	19	642	18	19	33	15
18	3	14	0	9	14	888	1	11	1
4	11	33	2	7	2	6	913	6	44
23	9	16	65	18	40	17	14	743	29
8	7	10	10	37	25	4	39	9	860

The rows represent the actual class labels and the columns are the predicted class labels. Therefore, the diagonals represent the true positive (TP).

The false positive (FP): sum of respective row minus the diagonal element.

False negative (FN): sum of respective column minus the diagonal element.

True negative (TN): sum of all elements in the matrix minus the corresponding row and column of class label.

Precision: TP / (TP+FP)

Recall: TP / (TP+FN)

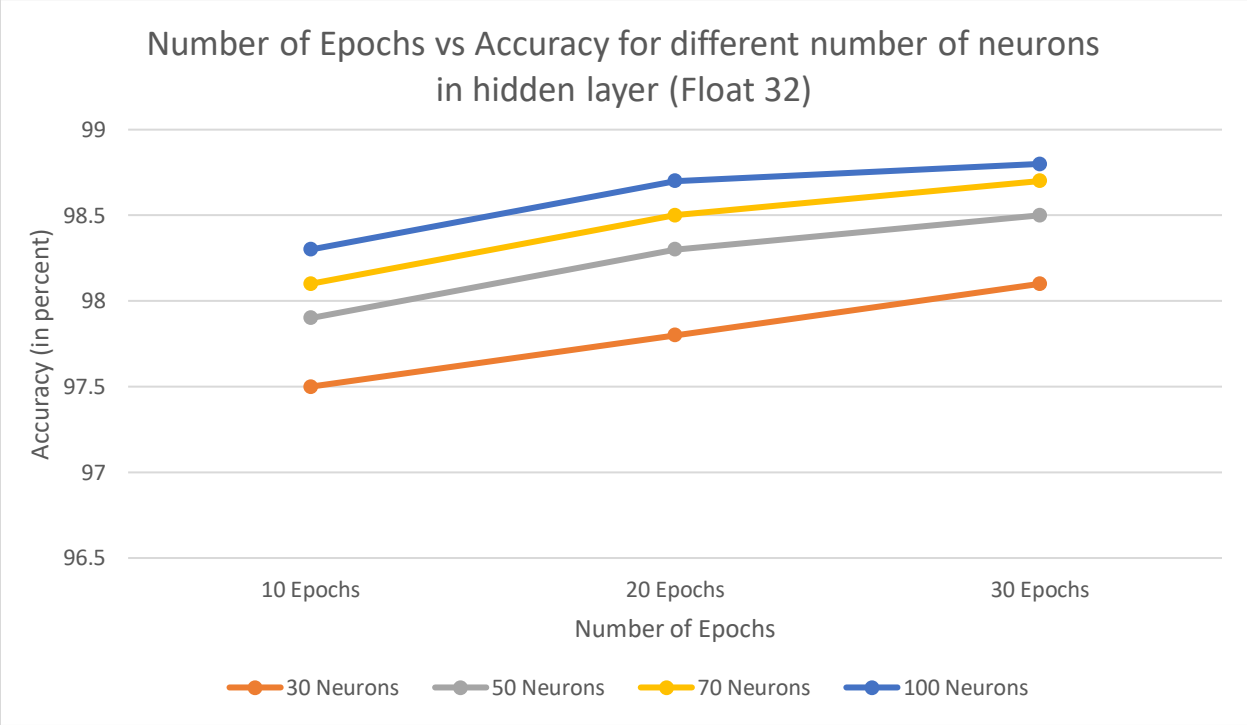
Accuracy: (TP+TN) / (TP+FP+TN+FN)

```
True Positive :
1033 1044 881 881 860 642 888 913 743 860


False Positive :
147 91 151 129 122 250 71 115 231 149


False Negative:
118 56 226 215 107 130 152 118 157 177


True Negative :
8696 8808 8736 8768 8904 8976 8888 8848 8864 8808


Precision: 0.853516
0.875 0.919434 0.853516 0.87207 0.875488 0.719727 0.925781 0.887695 0.762695 0.852051

Recall: 0.855469
0.897461 0.94873 0.79541 0.803711 0.88916 0.831543 0.853516 0.885254 0.825195 0.829102

Accuracy: 0.970703
0.974121 0.985352 0.962891 0.965332 0.977539 0.961914 0.978027 0.977539 0.961426 0.967773
```

TABULATION AND RESULTS

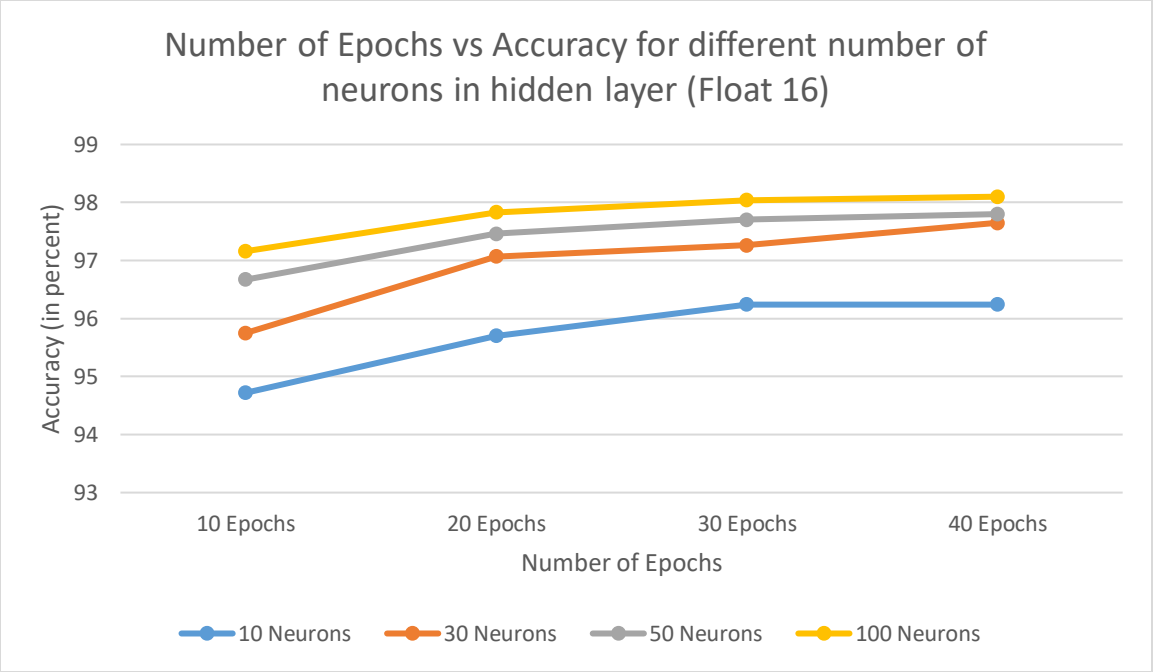
The neural network model was trained on the MNIST Hand written digits dataset and later on the Fashion dataset using 10, 30, 50, 70, 100 neurons in the hidden layers and varying number of epochs like 10, 20, 30, etc.

Effect of different number of neurons and epochs on model accuracy (Float32):



From the above diagram we can see that as we increase the number of neurons in the hidden layer, the accuracy improves. This is because the model gets more adaptive and can learn smaller details. We also see that the increase in the number of epochs increases the accuracy. But increasing the epochs very much can also cause the model to over-fit on the train data and consequently decrease the accuracy on the test data. We have to be careful to choose the proper number of epochs so as to maintain a good accuracy and avoid the overfitting of the data. This technique is called early stopping where the number of epochs are limited to the point where the error per epoch is the lowest.

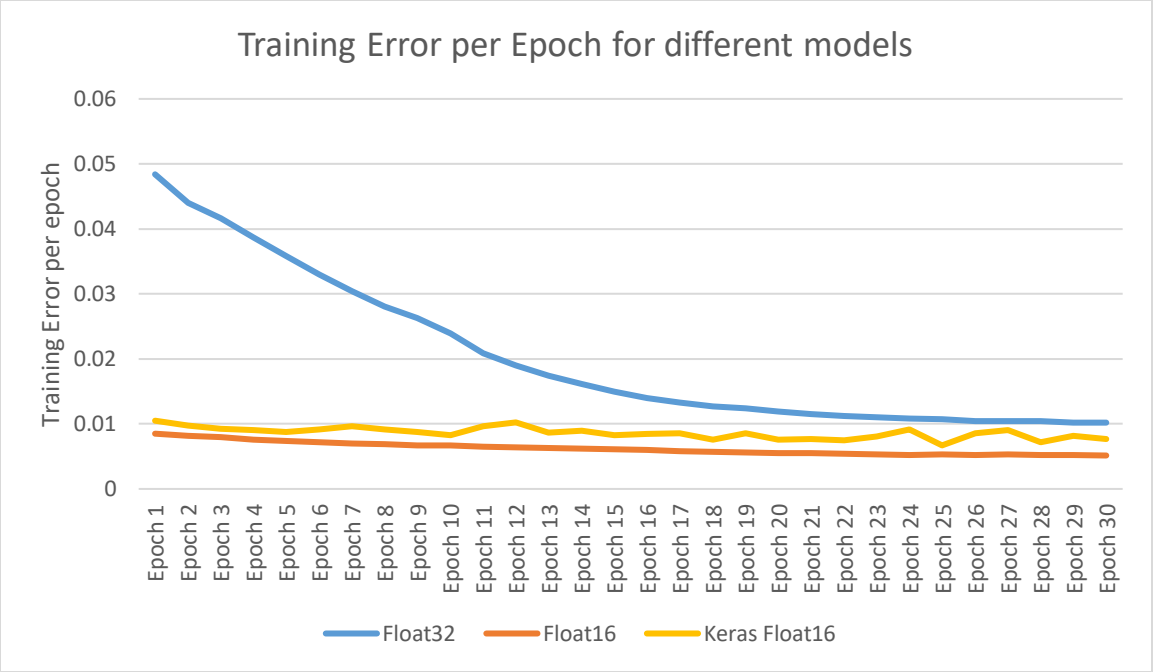
Effect of different number of neurons and epochs on model accuracy (Float16):



The above graph gives us the trend followed by the model on different number of neurons for varying epochs. We can see that the accuracy increases with the increasing number of epochs and also with the increasing number of neurons in the hidden layer. This is because, increasing the neurons in the hidden layers allows for smaller details to be learnt about the data and the increasing number of epochs reduces the error per epoch over the training data.

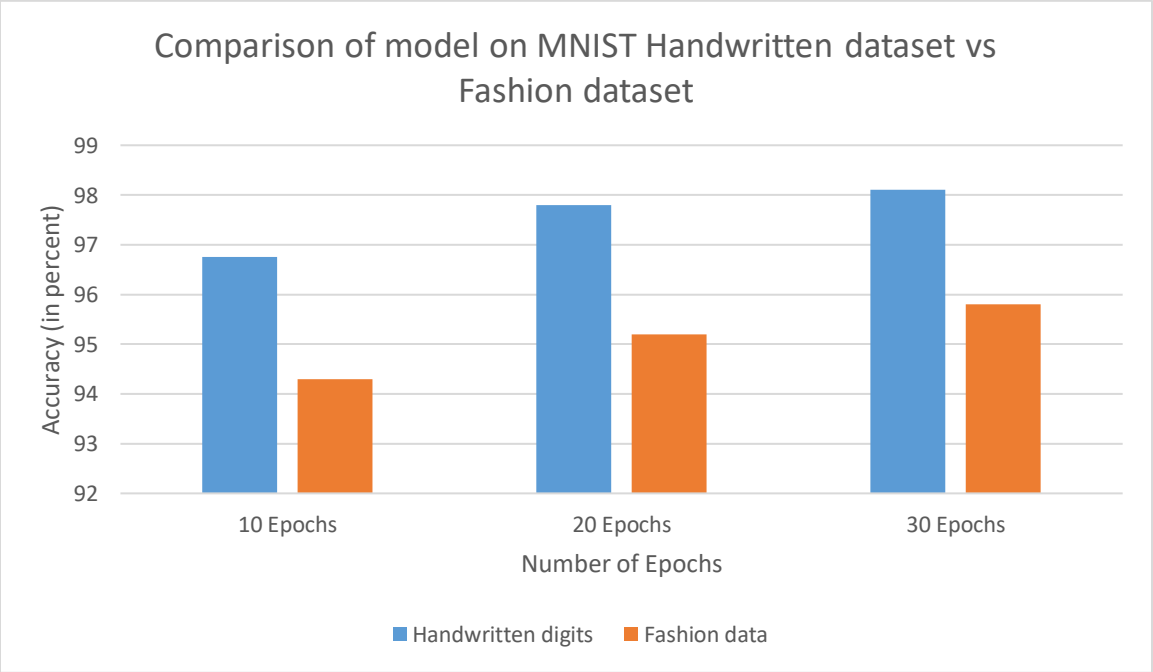
Comparing the above 2 graphs we can see that the accuracy of both the high precision and low precision models follow the same trend and also the low precision model is comparable to the high precision model. The reduced accuracy in the low precision model is attributed to the fact that noise is introduced in the data when converting it from high to low precision and then in the inner loop computations that noise creates errors and those errors accumulate, thereby reducing the accuracy.

Comparison of training error per epoch for different models:



The above diagram shows us the training error at each epoch for the models on different platforms. We can see that in case of float32 the error in the beginning is very high and as the epochs increase the training error keeps on decreasing at each epoch. After a certain number of epochs for a specified number of neurons, we should stop training the model otherwise we may risk over-fitting the model which may result an excellent train accuracy but a very poor test accuracy.

Comparison of Handwritten and Fashion dataset:



On conducting the experiments on the different dataset we found out that the result follows the same trend as the comparatively simple handwritten digits dataset. Since there were less features in the handwritten digits to learn as compared to the more complex fashion dataset, we saw that the handwritten digits data set required comparatively less number of neurons in the hidden layer to achieve a high accuracy. The same number of neurons for the fashion dataset were not sufficient to distinguish the images with more accuracy and therefore the accuracy reduced. The above graph shows that as we increase the number of neurons in the hidden layer, the accuracy for the fashion data set increases as the model becomes more adaptive and is able to learn more details about the data. Also, an increase in the number of hidden layers may affect the learning capability of the model, thereby further improving accuracy of the complex model.

Comparison of accuracy and runtime over different epochs for different models:

		cpp Model Float32 (avx)	cpp Model Float16 (multi-threaded)	TensorFlow Model (Keras) Float16
10 Epochs	**Accuracy (%)**	96.8	95.72	98.32
	Runtime (secs)	40.51	654	761
20 Epochs	**Accuracy (%)**	97.8	97.07	98.43
	Runtime (secs)	133.50	1593	1530
30 Epochs	**Accuracy (%)**	98.1	97.26	98.55
	Runtime (secs)	205.47	2425	2276

The above table shows the accuracy in percent and the runtime in seconds for the two cpp models using *float32* and *float16* data types and the TensorFlow (Keras) *float16* model for different number of epochs.

From the above table we can clearly see that the individual runtime for every model increases as the number of epochs increase. We can say that the runtime is proportional to the number of epochs.

Also, we can see that the accuracy for every model increases with the increasing number of epochs.

For cpp float32 and float16 models we can see that there is a huge difference between the runtime. This is because the library used to implement float16 (half.hpp) is not an internal library. Therefore, this file has to be read externally each time we implement the float16, thereby increasing the runtime. Also, the accuracy for float16 cpp model is less than the float 32 cpp model. This is because the float16 has to round the data and the

precision of the data is lost. Where in float32 we have 23 bits that represent the precision, in float16 we are only left with 10 bits, thereby decreasing precision and losing data.

In the above table we can see that the f32 model has a very good running time. This is because of the use of AVX2 SIMD instructions and O3 system flags. Also in case of f16 models we can see improvement as compared to the previous models. This improvement in the training time and testing time is attributed to the implementation of multithreading and vectorization of the matrices to perform matrix multiplication. The training time for the models could be further improved by playing with parameters like the mini batch size, which enables us to update the weights and biases calculated using gradient descent. We found that a larger value for the batch size decreases the training time effectively.

We can also see that our cpp models follows the same trend as the TensorFlow (Keras) model for the accuracy and the runtime but the accuracy of Keras model is higher than ours. This may be because of the complexity of the Keras model and the add convolution layer. But we see that our Float32 model runs much faster than the TensorFlow (Keras) model and produces comparable results.

Comparison of cpp Float32 and TensorFlow models:

		cpp Model Float32	TensorFlow Model Float32
10 Epochs	**Accuracy (%)**	96.2	95.69
	Runtime (secs)	51.33	30
20 Epochs	**Accuracy (%)**	97.8	95.65
	Runtime (secs)	152.52	60.928
30 Epochs	**Accuracy (%)**	97.9	95.72
	Runtime (secs)	231.37	90.88

From the above table we can see that the runtime of both the models is proportional to the number of epochs, that is, with the increasing number of epochs the runtime also increases. The accuracy of both the models increase with the increasing number of epochs.

We can also see that our cpp model gives us better accuracy over our dataset as compared to the TensorFlow model. But the runtime of the TensorFlow is much less than our runtime.

CONCLUSION AND FUTURE WORK

We successfully implemented image classification using deep neural network models (CPP, TensorFlow and Keras) for MNIST dataset and tested over Fashion dataset. To check the correctness of the results obtained in CPP model, we compared them with TensorFlow and Keras model. Multithreading implementation for float16 data types gave us 5 times faster speedup than float32 types while float32 data types using AVX instructions gave us 10 times faster speed than simple float32 types. In CPP model, using low precision data types such as float16, int, and int8 causes loss of data. Also, matrix multiplication and other mathematical non-linear operations performed at each layer resulted in loss of information thereby reducing accuracy. Because of absence in-built low precision libraries, we had to import the external libraries thereby increasing CPU runtime.

This project has scope of reducing CPU computation time with Intel's advanced hardware release that supports AVX 512 VNN instructions. Conversion of CPP model into CUDAs model would help understand the GPUs capability to recognize pattern recognition faster than CPU.

REFERENCES

[1] https://thinkingandcomputing.com/posts/using-avx-instructions-in-matrix multiplication.html

[2] https://adventuresinmachinelearning.com/python-tensorflow-tutorial/

[3] https://cognitivedemons.wordpress.com/2017/07/06/a-neural-network-in-10-lines-of-c-code/

[4] https://ginac.de/CLN/cln.html

[5] S. Gupta, A. Agarwal, K. Gopalakrishnan, P. Narayanan. "Deep Learning with Limited Numerical Precision", ICML'15 Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, Pages 1737-1746.