Classification using Neural Network

Deepanshu Yadav
UB Person No:
50321285
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
dyadav@buffalo.edu

Abstract

In this study, artificial Neural Network with a single hidden layer was developed from scratch and was used as a classifier to match Fashion MNIST data into 10 classes. The study was performed on dataset containing 70000 samples with image size of 28x28 pixels each. Python 3.7 and related libraries were used to manipulate the data and implement the model. It was observed that after tuning the hyperparameters using training and validation datasets, the model was able to match the Fashion MNIST data to correct classes with significant accuracy. Along with it, multi-layer Neural Network and convolutional Neural Network were also developed using open-source neural-network library, Keras. Performance of the three models was compared.

1 Introduction

Artificial neural networks (ANN) or connectionist systems are computing systems that are inspired by, but not identical to, biological neural networks that constitute animal brains. Such systems "learn" to perform tasks by considering examples, generally without being programmed with task-specific rules. An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron that receives a signal then processes it and can signal neurons connected to it.
In this study, a neural network model with single hidden layer is developed from scratch using Fashion MNIST dataset to match the images to correct classes of fashion clothing. The dataset contains 70000 samples with 28x28 pixel size images. It also contains associated class of each sample. Model is developed using 60000 samples as training set and 10000 samples as validation and test set. Results show significant accuracy and very low loss values for all the partitioned datasets. It establishes the importance of neural networks in addressing real world image classification problems with the use of machine learning.

2 Related Work

The work in this report includes reading the dataset and preprocessing it before using it to develop the neural network models. The first model uses gradient descent approach to refine the weights and bias and reach optimum value with dependencies on hyperparameters namely epochs, nodes in the hidden layer and learning rate. The updated weights and bias are used to predict the outcome again in every epoch. The predicted value is then compared with actual target value to find out the cost and accuracy. An optimum solution is reached when loss is minimized and accuracy is significant. The results are plotted against epochs to obtain the loss and accuracy curves to check improvements with every epoch. The following sections of the report include dataset reading, pre-processing the data, developing model architecture and then the results for all the three models. These present detailed analyses of each step involved in developing the first model and testing it.

3 Dataset

For training and testing of the classifiers, the Fashion-MNIST dataset was used. The Fashion-MNIST is a dataset of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.
Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255. The training and test data sets have 785 columns. The first column consists of the class labels (see above), and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image.

Fig 1: Example of how the data looks like

Each training and test example are assigned to one of the labels as shown in table 1.

Table 1: Labels for Fashion-MNIST dataset

S. No.	Description
1	T-shirt/top
2	Trouser
3	Pullover
4	Dress
5	Coat
6	Sandal
7	Shirt
8	Sneaker
9	Bag
10	Ankle Boot

4 Pre - Processing

The datasets were loaded in Python using *util_mnist_reader* provided by authors of the dataset. Preprocessing of the dataset was done using NumPy library of Python.
The target values in the data range from 0–9. These has to be converted to one-hot representation, where the array contains all zeros, except the index of the value as 1 i.e, value 4 will be represented as [0,0,0,0,1,0,0,0,0]. It was done using *to_categorical ()* function from TensorFlow library of Python. The rest of the dataset containing the 784 attributes was normalized to remove the bias which could have occurred due to large variations in values between the columns as they varied from 0 – 255 in the color range.

When using a convolutional layer as the first layer to our model, we need to reshape our data to (n_images, x_shape, y_shape, channels). The channels are set to 1 for grayscale images which we have in our dataset.

X_train = X_train.reshape(-1, 28,28, 1)
X_test = X_test.reshape(-1, 28,28, 1)

5 Model Architecture

Overview: The goal of this artificial neural network model with single hidden layer was to predict the class of a sample (out of 10 classes) correctly by minimizing the cost and maximizing accuracy using machine learning. The first challenge was to preprocess the given dataset and manipulate it in a way it fits the equations used in developing the model. Also, the hyperparameters used (epochs, nodes of hidden layer and learning rate) also need to be tuned by trial and error method to reduce the cost.

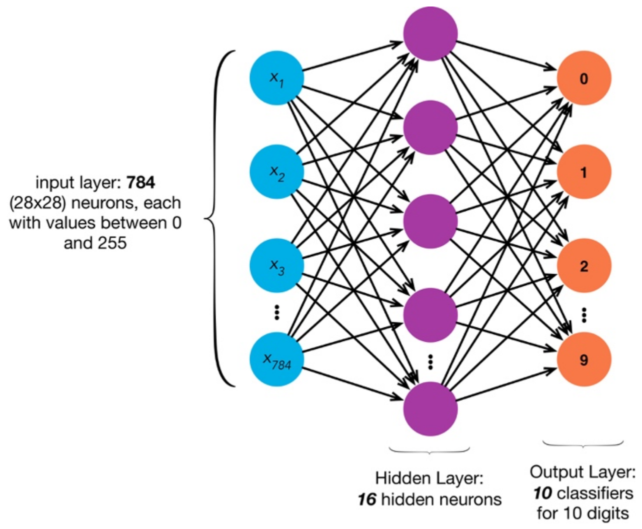

input layer: **784** (28x28) neurons, each with values between 0 and 255
x_1
x_2
x_3
x_{784}
0
1
2
9
Hidden Layer: ***16*** hidden neurons
Output Layer: ***10*** classifiers for 10 digits

Fig 2: Architecture of a 1-layer Neural Network

The below mentioned figure represents the work flow of the model development for the given dataset with m attributes. As shown below, every attribute has a weight associated with it along with 1 bias for all attributes.

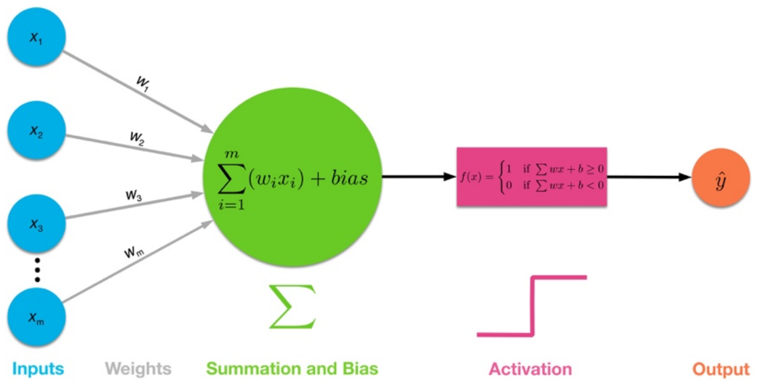

x_1
x_2
x_3
x_m
w_1
w_2
w_3
w_m
$\sum_{i=1}^{m}(w_i x_i) + bias$
$f(z) = \begin{cases} 1 & \text{if } \sum wx + b \geq 0 \\ 0 & \text{if } \sum wx + b < 0 \end{cases}$
$\hat{y}$
Inputs
Weights
Summation and Bias
Activation
Output

Fig 3: Computational Graph for Neural Network

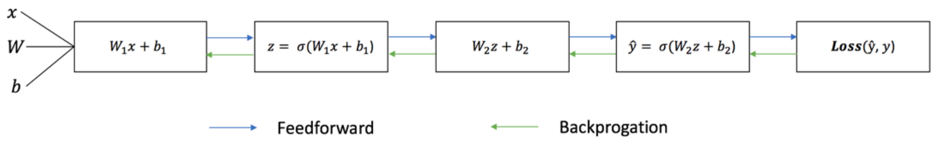


Fig 4: Sequential Graph representing Forward & Backward Propagation

Forward Propagation: In forward propagation, we need to calculate the predicted output using weights and biases and activation functions. Each iteration of the training process, we calculate the predicted output ŷ, known as forward propagation. Following equations represent the process of forward propagation:

$$z^{[1]} = w^{[1]}x + b^{[1]}$$

where $w^{[1]}$ and $b^{[1]}$ are the weights and bias, x is the input vector, and $z^{[1]}$is the output when we reach to the hidden layer from input.

$$a^{[1]} = \sigma\left(z^{[1]}\right)$$

where $a^{[1]}$ is the predicted value calculated using sigmoid as activation function for hidden layer. **Sigmoid function σ()** is defined as

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

As z goes from $-\infty$ to ∞, σ(z) goes from 0 to 1 in the manner mentioned in following graph.

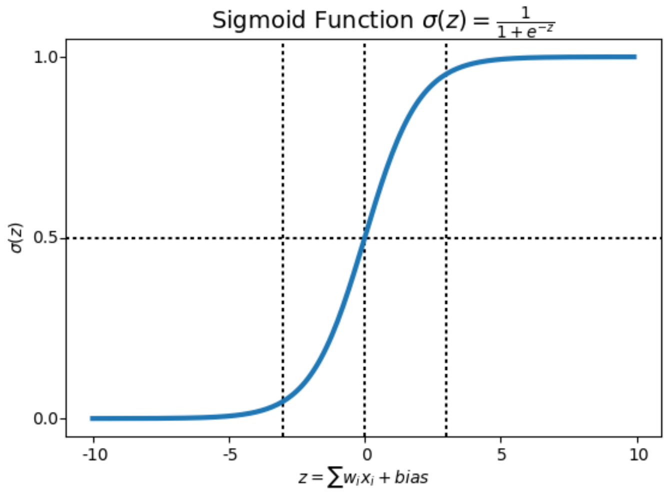


Fig 5: Sigmoid Function Plot

Then we calculate $z^{[2]}$ using the output of the first layer (hidden layer) as input and weights and bias for second layer as shown below.

$$z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$$

This is used to calculate $a^{[2]}$ which is calculated using softmax function as defined below:

$$a^{[2]} = softmax(z^{[2]})$$

Softmax function is defined as

$$softmax(z)_i = \frac{e^{zi}}{\sum_{j=1}^{k} e^{zj}}$$

As the calculation is performed, for each sample, there will be an associated loss. The average of losses associated with each sample will give us the overall cost incurred in the model which is represented be the equation below

$$L(a^{[2]}, y) = -\sum yloga^{[2]}$$

where
L = total loss
y = actual target value
$a^{[2]}$ = predicted value as of result of calculations performed by the model

Backward Propagation: Now that we've measured the error of the prediction (loss), we need to propagate the error back, and to update our weights and biases. To minimize the cost, Gradient Descent algorithm is used which is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. Gradients for weights and bias are calculated in the below mentioned way. First, we take the difference between predicted value $a^{[2]}$ and target value (y).

$$da^{[2]} = a^{[2]} - y$$

This difference is used to calculate gradient for weights used in second layer as shown below:

$$dw^{[2]} = da^{[2]}a^{[1]}$$

$$db^{[2]} = \sum da^{[2]}$$

where
$dw^{[2]}$ = gradient for weights for second layer
$db^{[2]}$ = gradient for bias for second layer difference

For calculating gradients for layer 1 parameters, we need derivative of sigmoid function with respect to the input vector

$$\frac{d\sigma(z)}{dz} = a^{[1]}(1 - a^{[1]})$$

We then define $da^{[1]}$ as shown below:

$$da^{[1]} = \left(da^{[2]}\right) * w^{[2]} * a^{[1]}(1 - a^{[1]})$$

The gradients for layer 1 are defined as:

$$dw^{[1]} = da^{[1]}x$$

$$db^{[1]} = \sum da^{[1]}$$

where
$dw^{[1]}$ = gradient for weights for second layer
$db^{[1]}$ = gradient for bias for second layer difference

These gradients are used to calculate the updated weights and bias with every epoch as mentioned below using learning rate which can be described as size of steps taken to minimize the cost. This has to be optimum as if it's too low, calculating gradient will be time consuming. If it's too high, we risk overshooting the lowest point since slope of curve is constantly changing.

Updated weights and bias are represented by below equations.

$$w^{[2]} = w^{[2]} - \propto dw^{[2]}$$

$$b^{[2]} = b^{[2]} - \propto db^{[2]}$$

$$w^{[1]} = w^{[1]} - \propto dw^{[1]}$$
$$b^{[1]} = b^{[1]} - \propto db^{[1]}$$

where $\propto$ is the learning rate.

This process is repeated for a number of iterations known as epochs, which is also a hyperparameter in the present model along with learning rate. No. of epochs are chosen such that cost gets minimized with significant accuracy and precision also obtained for given learning rate. In this study, epochs are chosen to be 30 for developing single layer neural network and learning rate is varied to get optimum values for weights and bias. The number of neurons in hidden layers is also a hyperparameter which is tuned to 20 to get the best fit.

These predicted values are then compared with actual target values to calculate the evaluation metrics explained below.

$$Accuracy = \frac{N_{correct}}{N}$$

where
$N_{correct}$ = number of correctly classified data samples
N = total number of samples of the set

For Multi-layer Neural Network and Convolutional Neural Network, high level Python libraries were used to develop the models and train them for certain number of epochs, layers, neurons in each hidden layer and batch size.

6 Results of Experimentation

The task to be performed was predicting the class (1 – 10) of each sample with some confidence using Single Layer Neural Network, Multi-Layer Neural Network and Convolutional Neural Netowrk.

Evaluation metrics: I have used Accuracy and Confusion Matrix metrics for each dataset for evaluation of the efficiency of the models. Also, the cost values for training and validation datasets have been evaluated with respect to epochs. As the weights are initialized with random values before running the model every time, the model gives varied values of the evaluation metrics every time the model is run.

Single Layer Neural Network:

The hyperparameters for the first model, i.e. epochs, neurons in hidden layer and learning rate were set to get the optimum results by trial and error method. Cost was calculated for various values and the one with minimum cost was chosen as the final hyperparameters for the model. Figures below represent loss plots for various values of hyperparameters.

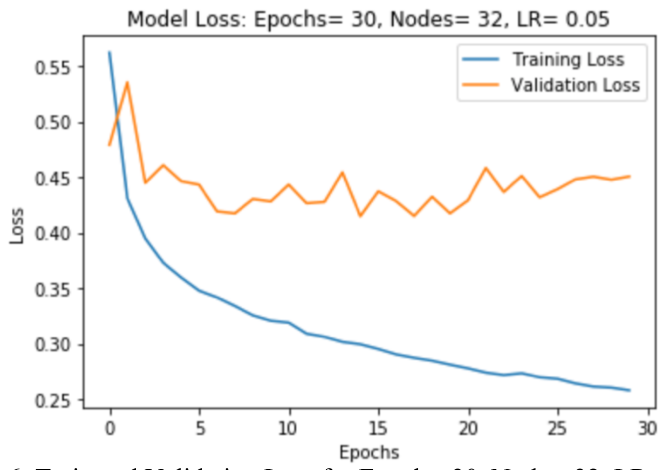


Fig 6: Train and Validation Loss for Epochs: 30, Nodes: 32, LR: 0.05

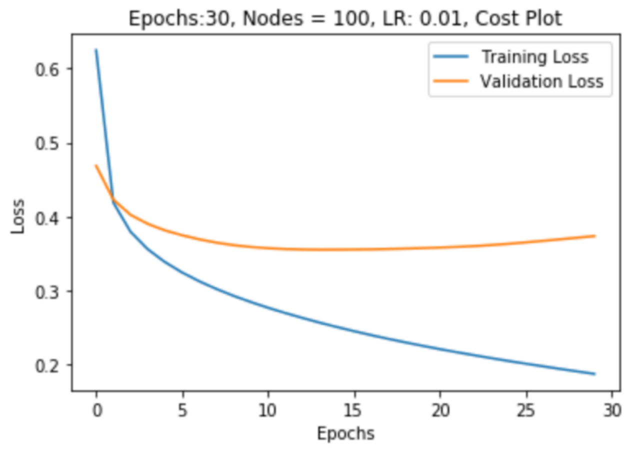


Fig 7: Train and Validation Loss for Epochs: 30, Nodes: 100, LR: 0.01

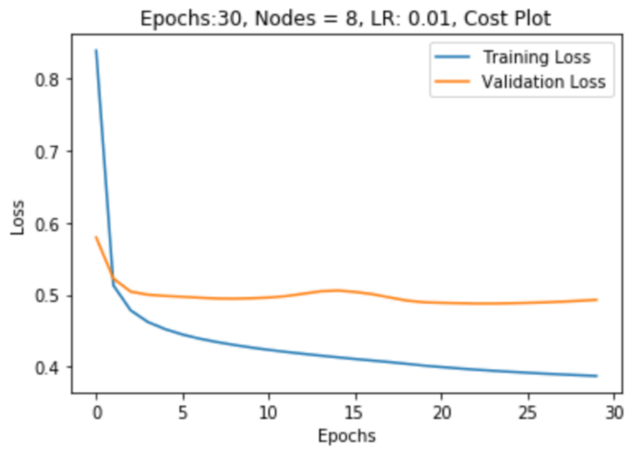


Fig 8: Train and Validation Loss for Epochs: 30, Nodes: 8, LR: 0.01

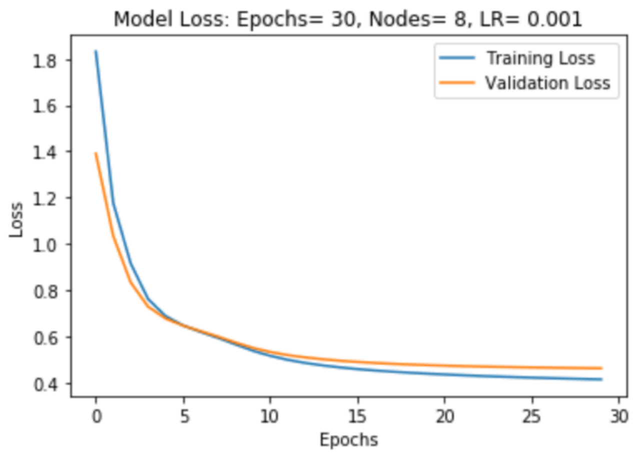
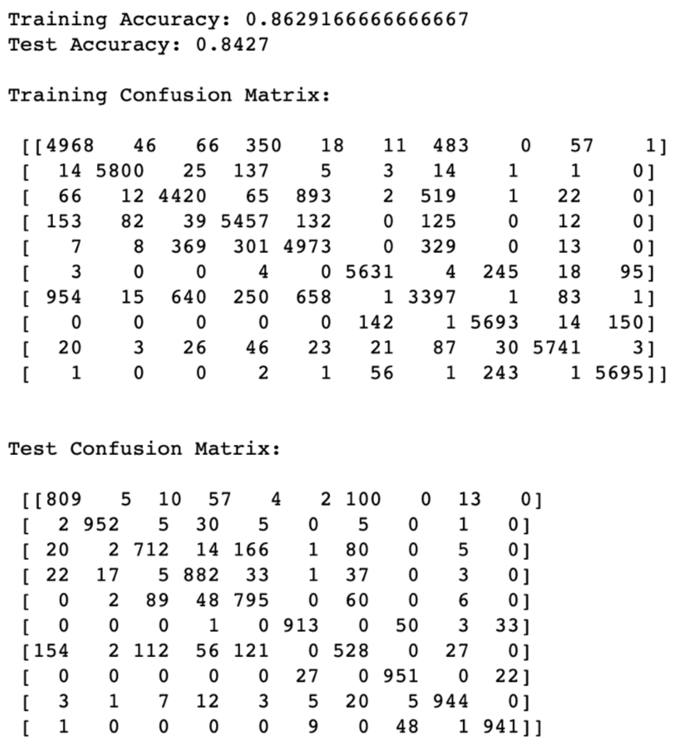


Fig 9: Train and Validation Loss for Epochs: 30, Nodes: 8, LR: 0.001

For Single Layer Neural Network model, the results for final model trained for 30 epochs, 8 neurons in the hidden layer and learning rate of 0.001 are as mentioned below:



```
Training Accuracy: 0.8629166666666667
Test Accuracy: 0.8427

Training Confusion Matrix:

 [[4968   46   66  350   18   11  483    0   57    1]
 [  14 5800   25  137    5    3   14    1    1    0]
 [  66   12 4420   65  893    2  519    1   22    0]
 [ 153   82   39 5457  132    0  125    0   12    0]
 [   7    8  369  301 4973    0  329    0   13    0]
 [   3    0    0    4    0 5631    4  245   18   95]
 [ 954   15  640  250  658    1 3397    1   83    1]
 [   0    0    0    0    0  142    1 5693   14  150]
 [  20    3   26   46   23   21   87   30 5741    3]
 [   1    0    0    2    1   56    1  243    1 5695]]


Test Confusion Matrix:

 [[809   5  10  57   4   2 100   0  13   0]
 [  2 952   5  30   5   0   5   0   1   0]
 [ 20   2 712  14 166   1  80   0   5   0]
 [ 22  17   5 882  33   1  37   0   3   0]
 [  0   2  89  48 795   0  60   0   6   0]
 [  0   0   0   1   0 913   0  50   3  33]
 [154   2 112  56 121   0 528   0  27   0]
 [  0   0   0   0   0  27   0 951   0  22]
 [  3   1   7  12   3   5  20   5 944   0]
 [  1   0   0   0   0   9   0  48   1 941]]
```

Fig 10: Accuracy and Confusion Matrix for Single Layer Neural Network

Multi-Layer Neural Network:

The hyperparameters for the second model, i.e. epochs, number of hidden layers and neurons in hidden layer were set to get the optimum results by trial and error method. Cost was calculated for various values and the one with minimum cost was chosen as the final hyperparameters for the model. Figures below represent loss plots for various values of hyperparameters.

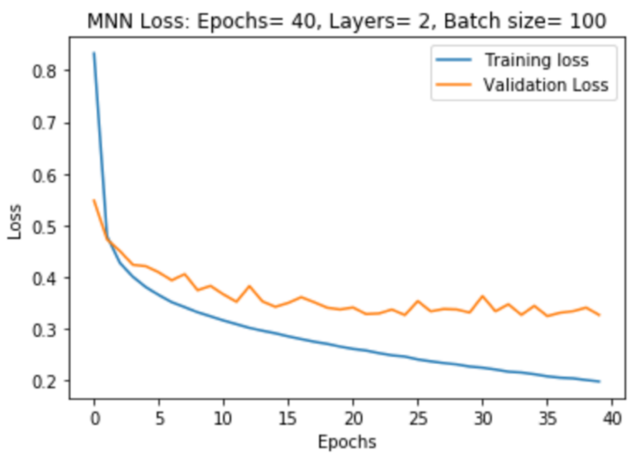


Fig 11: Train and Validation Loss for Epochs: 40, Hidden Layers: 2, Neurons: 128, Batch size: 100

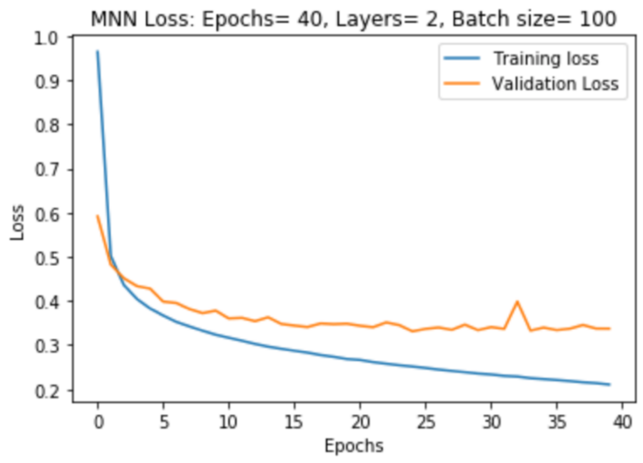


Fig 12: Train and Validation Loss for Epochs: 40, Hidden Layers: 2, Neurons: 64, Batch size: 100

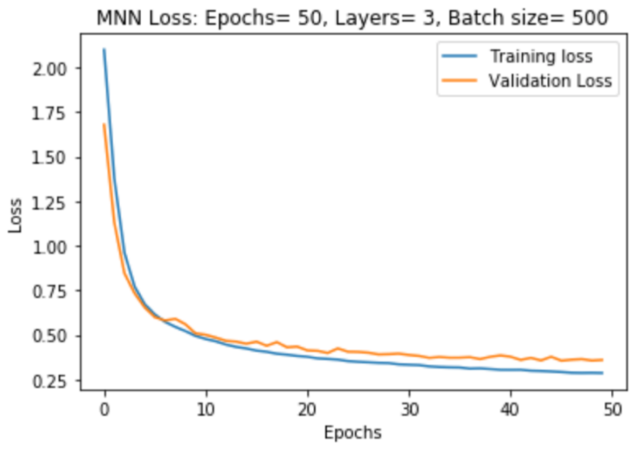
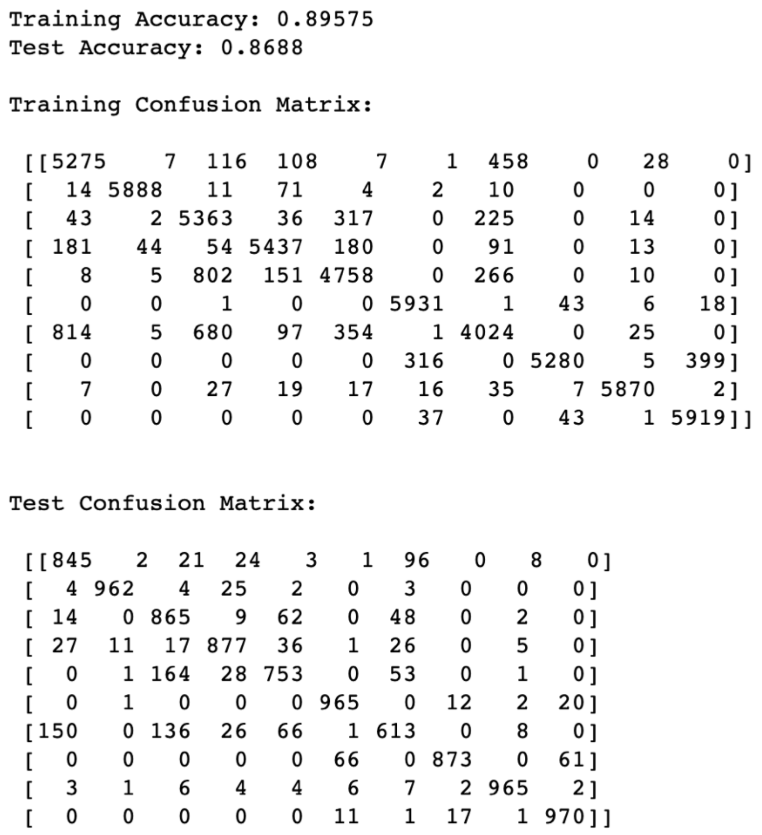


Fig 13: Train and Validation Loss for Epochs: 50, Hidden Layers: 3, Neurons: 64, Batch size: 500

For Multi-Layer Neural Network model, the results for final model trained for 50 epochs, 64 neurons in the 3 hidden layers and batch size of 500 are as mentioned below:



```
Training Accuracy: 0.89575
Test Accuracy: 0.8688

Training Confusion Matrix:

 [[5275    7  116  108    7    1  458    0   28    0]
 [  14 5888   11   71    4    2   10    0    0    0]
 [  43    2 5363   36  317    0  225    0   14    0]
 [ 181   44   54 5437  180    0   91    0   13    0]
 [   8    5  802  151 4758    0  266    0   10    0]
 [   0    0    1    0    0 5931    1   43    6   18]
 [ 814    5  680   97  354    1 4024    0   25    0]
 [   0    0    0    0    0  316    0 5280    5  399]
 [   7    0   27   19   17   16   35    7 5870    2]
 [   0    0    0    0    0   37    0   43    1 5919]]


Test Confusion Matrix:

 [[845   2  21  24   3   1  96   0   8   0]
 [  4 962   4  25   2   0   3   0   0   0]
 [ 14   0 865   9  62   0  48   0   2   0]
 [ 27  11  17 877  36   1  26   0   5   0]
 [  0   1 164  28 753   0  53   0   1   0]
 [  0   1   0   0   0 965   0  12   2  20]
 [150   0 136  26  66   1 613   0   8   0]
 [  0   0   0   0   0  66   0 873   0  61]
 [  3   1   6   4   4   6   7   2 965   2]
 [  0   0   0   0   0  11   1  17   1 970]]
```

Fig 14: Accuracy and Confusion Matrix for Multi-Layer Neural Network

Convolutional Neural Network:
The hyperparameters for the third model, i.e. epochs, number of hidden layers and neurons in hidden layer were set to get the optimum results by trial and error method. Loss was calculated for various values and the one with minimum loss was chosen as the final hyperparameters for the model. Figures below represent loss plots for various values of hyperparameters.

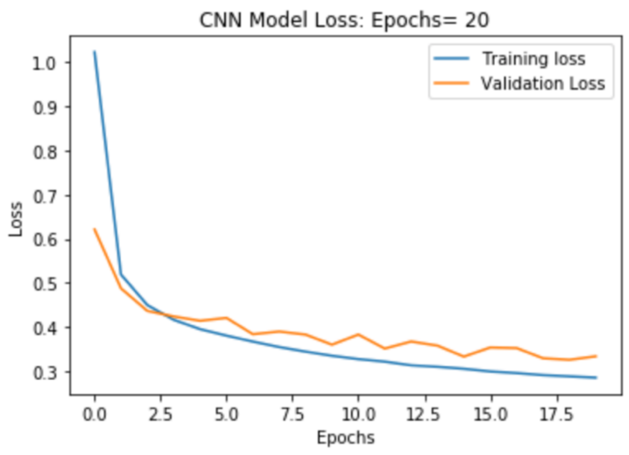


Fig 15: Train and Validation Loss for Epochs: 20, Neurons: 64, Batch size: 64

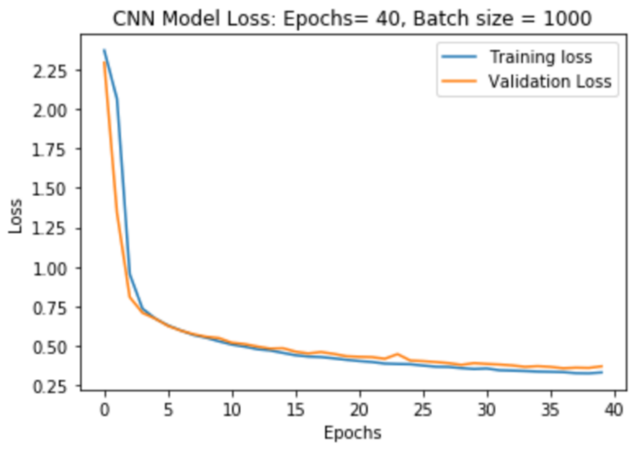


Fig 16: Train and Validation Loss for Epochs: 40, Neurons: 64, Batch size: 1000

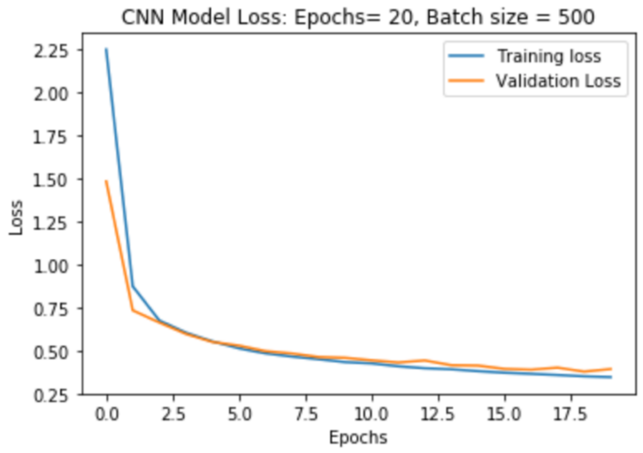
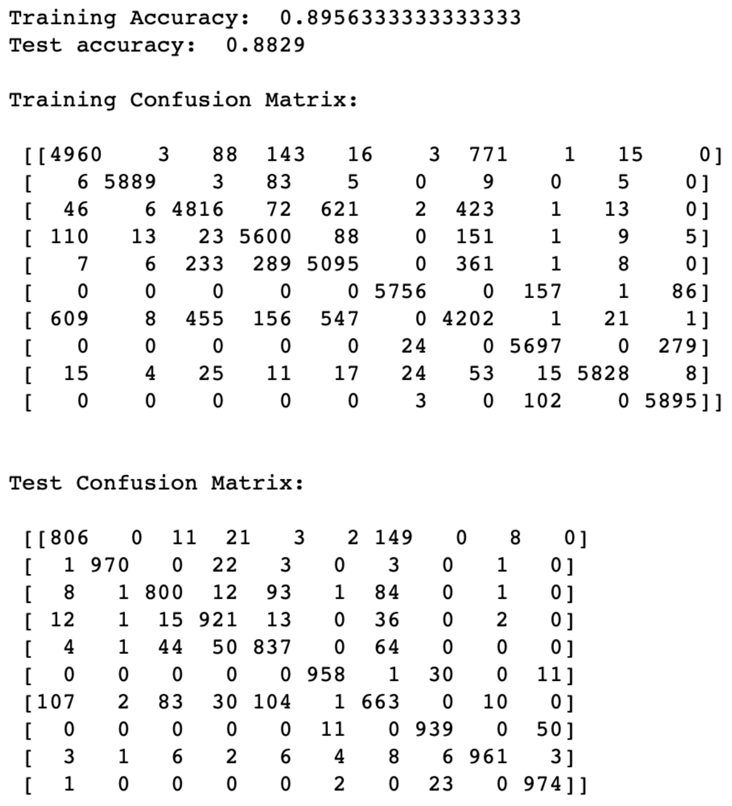


Fig 17: Train and Validation Loss for Epochs: 20, Neurons: 64, Batch size: 500

For Convolutional Neural Network model, the results for final model trained for 50 epochs, 64 neurons and batch size of 500 are as mentioned below:



```
Training Accuracy:  0.8956333333333333
Test accuracy:  0.8829

Training Confusion Matrix:

 [[4960    3   88  143   16    3  771    1   15    0]
 [   6 5889    3   83    5    0    9    0    5    0]
 [  46    6 4816   72  621    2  423    1   13    0]
 [ 110   13   23 5600   88    0  151    1    9    5]
 [   7    6  233  289 5095    0  361    1    8    0]
 [   0    0    0    0    0 5756    0  157    1   86]
 [ 609    8  455  156  547    0 4202    1   21    1]
 [   0    0    0    0    0   24    0 5697    0  279]
 [  15    4   25   11   17   24   53   15 5828    8]
 [   0    0    0    0    0    3    0  102    0 5895]]


Test Confusion Matrix:

 [[806   0  11  21   3   2 149   0   8   0]
 [  1 970   0  22   3   0   3   0   1   0]
 [  8   1 800  12  93   1  84   0   1   0]
 [ 12   1  15 921  13   0  36   0   2   0]
 [  4   1  44  50 837   0  64   0   0   0]
 [  0   0   0   0   0 958   1  30   0  11]
 [107   2  83  30 104   1 663   0  10   0]
 [  0   0   0   0   0  11   0 939   0  50]
 [  3   1   6   2   6   4   8   6 961   3]
 [  1   0   0   0   0   2   0  23   0 974]]
```

Fig 18: Accuracy and Confusion Matrix for Convolutional Neural Network

From the results obtained from the three models, we can observe that the accuracy and confusion matrix are getting refined when we move from single layer neural network to multi-layer and then to convolutional neural network. The accuracy for test set is found to be 84% for Single Layer Neural Network, 86% for Multi-Layer Neural Network and 88% for Convolutional Neural Network which implies that the models developed are working with significant efficiency to classify image data.

7 Conclusion

In this study, artificial neural network models were developed to be used as classifiers to classify Fashion MNIST data into 10 classes. Models were trained for various values of hyperparameters. The evaluation metrics obtained showed significant efficiency of the models in classifying the data into respective classes for a particular set of hyperparameters for each model. The performance of the models when measured with accuracy and confusion matrix metrics was observed to improve when we moved from Single Layer Neural Network to Multi-Layer Neural Network and then to Convolutional Neural Network.

References

[1] https://en.wikipedia.org/wiki/Artificial_neural_network

[2] https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f

[3] https://towardsdatascience.com/mnist-cnn-python-c61a5bce7a19

[4] https://towardsdatascience.com/neural-networks-from-scratch-easy-vs-hard-b26ddc2e89c7

[5] https://towardsdatascience.com/fashion-product-image-classification-using-neural-networks-machine-learning-from-scratch-part-e9fda9e47661

[6] https://pravarmahajan.github.io/fashion/