MODEL ACCURACY PREDICTION USING NEURAL NETWORKS AND CONVOLUTIONAL NEURAL NETWORKS(CNN): A MULTI-CLASS CLASSIFICATION PROBLEM

Sriparna Chakraborty
Department of Computer Science
University of Buffalo
Buffalo, NY 14214
sriparna@buffalo.edu

ABSTRACT

The purpose of this experiment is to implement neural network and convolutional neural network for the task of classification. I have used the Fashion-MNIST dataset of Zalando's article images for training, validation and testing. The dataset contains 60,000 examples of training data and 10,000 examples of testing data. Each example is a 28x28 grayscale image, associated with a label from 10 classes. The training and validation accuracy have been compared with respect to the epochs and evaluated as a graph plot. The accuracy, precision and recall values for the testing dataset have been measured with the tuned hyper-parameters (weight, bias and learning rate) in the backpropagation of the neural network. The same dataset is being evaluated once with a neural network with one hidden layer (code written in Python from scratch), a neural network with three hidden layers and a convoluted neural network using high level Neural Network libraries in Keras with tensorflow as backend.

1 INTRODUCTION

In machine learning and statistics, classification is the problem of identifying to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. In our experiment, we have implemented Neural Networks for this task of classification.

The neural network is an AI based technology based on the structure of the neurons inside a human brain. It was around the 1940s when Warren McCulloch and Walter Pitts create the so-called predecessor of any Neural Network but it was Geoffrey Hinton who made this algorithm come to the surface through his learning algorithm, BACKPROPAGATION.

Neural networks are typically organized in layers. Layers are made up of a number of interconnected 'nodes' which contain an 'activation function'. Patterns are presented to the network via the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted 'connections'. Then the hidden layers are connected to an 'output layer'. Backpropagation in neural networks is a unique property that enables us to propagate from the output layer back to the input layer via the hidden layers and perform the tuning of the weights and biases associated with each connection via gradient descent. Here 'learning' is a supervised process that occurs with each cycle or 'epoch' (i.e. each time the network is presented with a new input pattern)

through a forward activation flow of outputs, and the backward error propagation of weight adjustments. More simply, when a neural network is initially presented with a pattern it makes a random 'guess' as to what it might be. It then sees how far its answer was from the actual one and makes an appropriate adjustment to its connection weights.

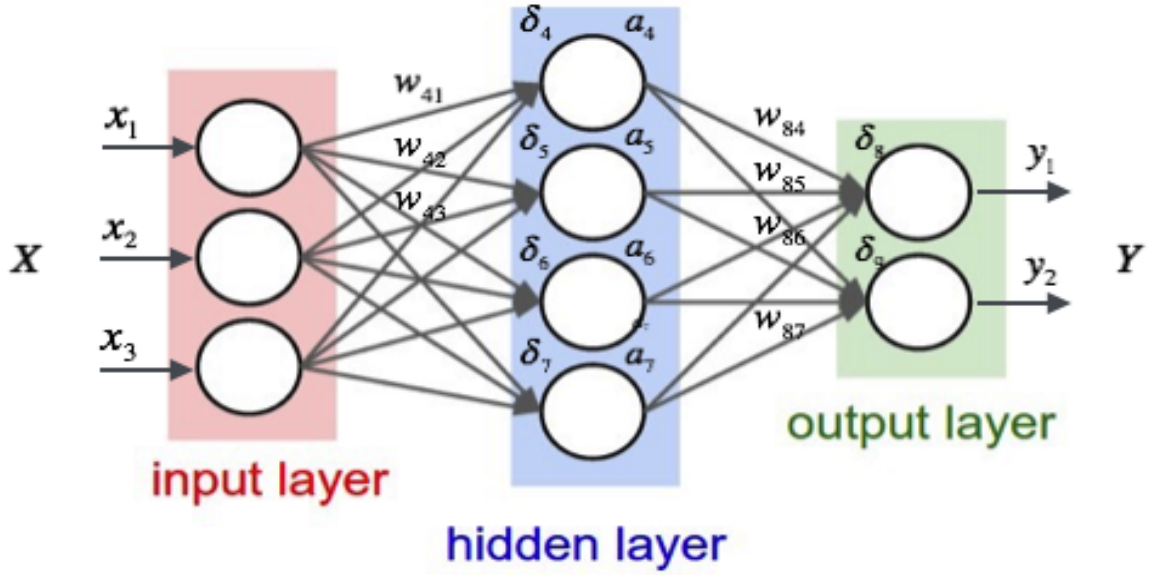


Figure 1: How a neural network looks like

In this experiment, we perform the same task of data prediction with three different algorithms:

1. **TRAIN USING NEURAL NETWORK WITH ONE HIDDEN LAYER**: We use gradient descent to train the model using a group of hyperparameters (weights and biases). The implementation code has been written in Python from scratch.
2. **TRAIN USING MULTI-LAYER NEURAL NETWORK**: This is implemented with high-level Neural Network library, Keras using a group of hyperparameters.
3. **TRAIN USING CONVOLUTIONAL NEURAL NETWORK**: This is also implemented with high-level Neural Network library, Keras using a group of hyperparameters.

The accuracy, precision and recall have been calculated from the confusion matrix for each epoch. The prediction has been improved for each epoch by tuning the hyper-parameters in a trial and error method.

The graph plots that have been used for evaluation of the model are:

a. Training Error VS Epoch
b. Training Accuracy and Validation Accuracy VS Epoch

2 DATASET

For training and testing of our classifiers, we have used the Fashion-MNIST dataset. 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.

Each training and test example is assigned to one of the labels as shown below:

1	T-shirt/Top
2	Trouser
3	Pullover
4	Dress
5	Coat
6	Sandal
7	Shirt
8	Sneaker
9	Bag
10	Ankle Boot

Figure 2: Example of how the data looks like

Each of the training datasets X_train (features) and Y_train (labels) are of dimensions 60000x784 and 60000x10 respectively and each of the testing datasets X_test and Y_test is of dimensions 10000x784 and 10000x10 respectively. The label datasets have been encoded using **one-hot encoding** to make it more expressive and efficient for proper prediction purposes.

The dataset is loaded by importing mnist_reader package in python and keras.datasets.fashion_mnist in Keras framework.

3 PREPROCESSING

Data preprocessing is an integral step in Machine Learning as the quality of data and the useful information that can be derived from it directly affects the ability of our model to learn. Therefore, it is extremely important that we preprocess our data before feeding it into our model.

The preprocessing steps that I performed in this experiment on the dataset are as follows:

1. The feature datasets X_train and X_test have been normalized by dividing each value by 255 as our input dataset consists of images only with a pixel range from 0 to 255. So it is highly advisable to normalize each largely varying pixel value to a standard value so that its mean and standard deviation are uniform.
2. The label datasets Y_train and Y_test have been encoded using one-hot encoding both in Keras and python frameworks so that the datasets become more expressive in terms of class labels for efficient prediction purposes.

4 ARCHITECTURE

TASK 1:

The basic model for the Neural Network used in this experiment is as follows:

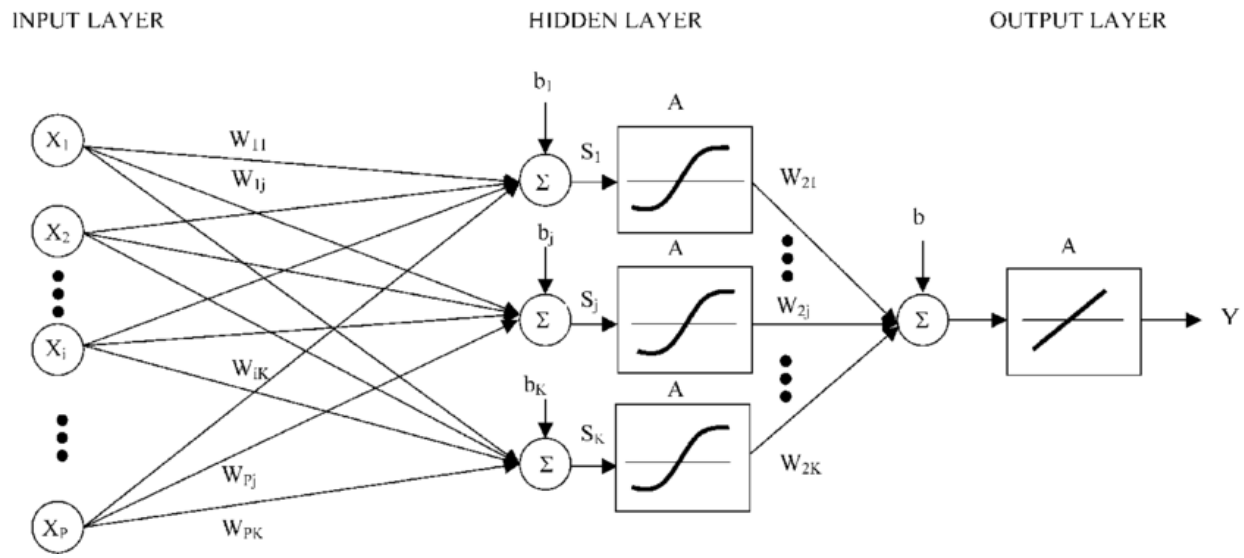


Figure 3: Structure of the neural network implemented in our experiment

The activation function used in the hidden layers is a 'sigmoid' function that scales the weights associated with each connection of the input feature to some probabilistic measures varying between 0 and 1. The activation function used for the output layer is a 'softmax' function that gives as output a probabilistic value for each of the ten output classes.

FORWARD PROPAGATION:

For each epoch I have calculated $Z_i = (W^T_i.X_i) + b_i$ for each of the features of the input dataset X (i=1,2,3,…,784) and passed it to $\sigma(Z) = \frac{1}{1+e^{-z}}$, where Z_i is the genesis function for each of the input features of X, W_i is the weight associated with each of the input features, b_i is a bias and , $\sigma(Z)$ is our sigmoid function.

In the last hidden layer of the network , the output $\sigma(Z)$ of each hidden node is passed to a softmax function that provides as output, probabilistic values for each of the ten output classes. The highest probability value among the ten values gives the output class of that particular input image.
Softmax function is defined as:

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

The equations used to calculate the forward propagation parameter values are listed below:

1. $z_1 = W_1 x + b_1$
2. $a_1 = \sigma(z_1)$
3. $z_2 = W_2 a_1 + b_2$
4. $a_2 = softmax(z_2)$
5. $L(a_2, y) = -\Sigma y log a_2$

BACKPROPAGATION:

In backward propagation of our neural network, we calculate the gradient of each of the hyperparameters from output layer→hidden layers→input layer and try to tune them by comparing with the gradient of the last updated value of the node. This way, we get an idea of how we can make our network perform better by feeding the tuned values we get by backpropagation again in forward propagation.

The equations used to calculate the gradient values in backpropagation are listed below (assuming that we have used only one hidden layer):

1. $\Delta a_2 = \frac{\partial L}{\partial a_2}$
2. $\Delta z_2 = \frac{\partial L}{\partial z_2} = a_2 - y$
3. $\Delta a_1 = \frac{\partial L}{\partial z_2} \times \frac{\partial z_2}{\partial a_1} = (a_2 - y) W_2$
4. $\Delta z_1 = \Delta a_1 \frac{\partial a_1}{\partial z_1} = (a_2 - y) W_2 \, a_1(1 - a_1)$
5. $\Delta W_2 = (a_2 - y) a_1$
6. $\Delta W_1 = (a_2 - y) W_2 \, a_1(1 - a_1) x$

Loss function used here is the Cross Entropy function denoted by L:

$$L = \frac{-(ylogp + (1 - y)\log(1 - p)}{m}$$

With the final values of W and bias, I have predicted the output values of the Testing dataset using the same genesis function and compared with the Actual output using the confusion matrix and its metrics.

CONFUSION MATRIX:

A confusion matrix, also known as an error matrix, is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa).

The four metrics of the confusion matrix are:

a. TP/ True Positive: Actual output is positive and Predicted is positive
b. TN/ True Negative: Actual output is negative and Predicted is negative
c. FP/ False Positive: Actual output is negative and Predicted is positive
d. FN/ False Negative: Actual output is positive and Predicted is negative

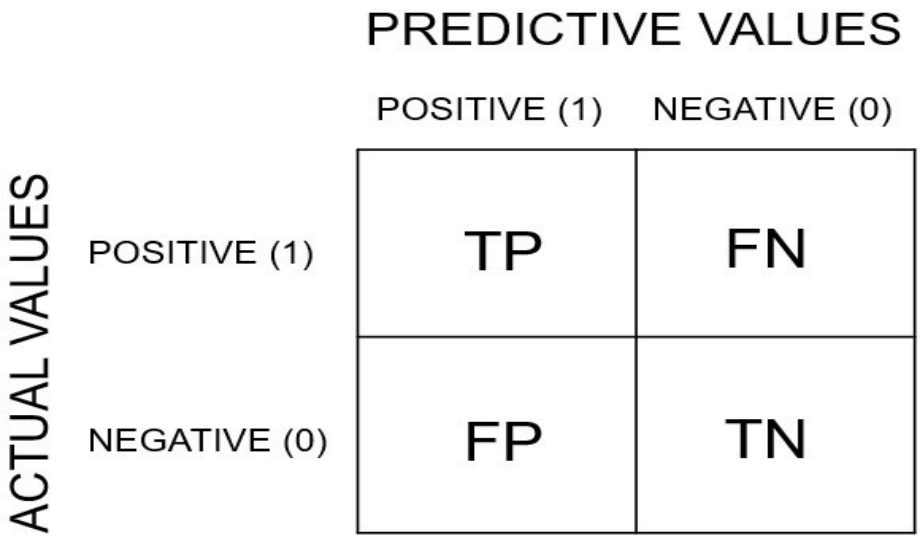


Figure 4: Confusion Matrix structure

The resultant values Accuracy, Precision and Recall are calculated from the confusion matrix values as follows:

$$\text{ACCURACY} = \frac{(TP+TN)}{(TP+TN+FP+FN)}$$

$$\text{PRECISION} = \frac{TP}{(TP+FP)}$$

$$\text{RECALL} = \frac{TP}{(TP+FN)}$$

TASK 2:

- In Keras library with tensorflow as backend, it is quite simple to build a neural network model due to the presence of a large number of high level Neural Network libraries.
- I have implemented a three-layer neural network and have utilized 'Dense' method of **<keras.layers>** package to build the hidden layers with activation functions defined as 'sigmoid' or 'softmax'. For each of the hidden layers, number of hidden nodes taken are 128.
- The model is compiled using 'sgd'/stochastic gradient descent optimizer, 'sparse_categorical_crossentropy' as the Loss function and 'accuracy' as the metrics to be calculated.
- The model is then trained with 48,000 training data and 12,000 validation data for 100 epochs and mini-batches of size 120 for each epoch.
- Finally, the model is evaluated on the test dataset using the 'evaluate' method of the model classifier built as an object of **<keras.sequential>**

TASK 3:

- **<keras.layers>** has methods like 'Conv2D' and 'MaxPooling2D' to build a CNN model classifier.
- The activation function used in the hidden layers is sigmoid function and that of the output layer is a softmax function.
- In our experiment, we have used a kernel of size 2x2 and number of hidden nodes of the first layer to be 64. The maxpool of size 2x2 used in the first layer reduces the dimensions of each input image of 28x28 to 14x14. The output from this layer is fed to the next hidden layer for the second convolution.
- The second layer has 32 hidden nodes with a kernel of size 2x2. After convolving each of the resultant image with the kernel, the output is again fed into a maxpool function which again reduces the image size to 7x7.
- Now the final output from the second layer is fed to the softmax regularizer that gives as output ten probabilistic values for each input image.
- The model is then compiled using 'sgd' as the optimizer, 'categorical_crossentropy' as the loss function and 'accuracy' as the metrics to be considered for evaluation of the model.
- The model is then trained with 48,000 training data and 12,000 validation data for 100 epochs and mini-batches of size 120 for each epoch.
- Finally, the model is evaluated on the test dataset using the 'evaluate' method of the model classifier built as an object of **<keras.sequential>.**

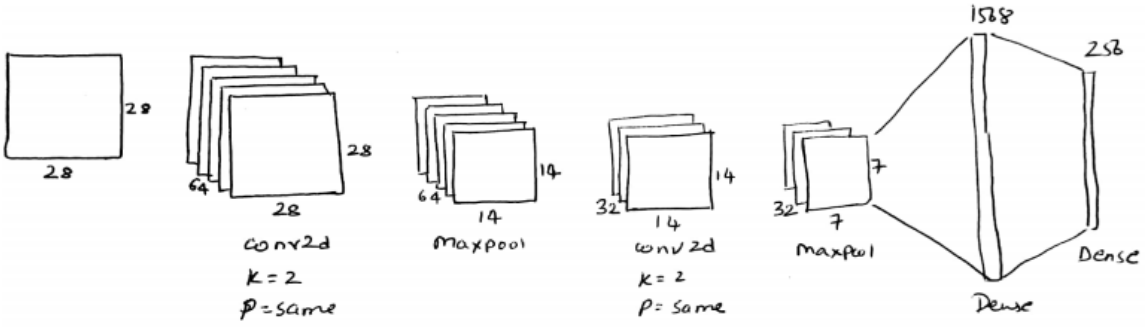


Figure 5: CNN model structure used in our experiment

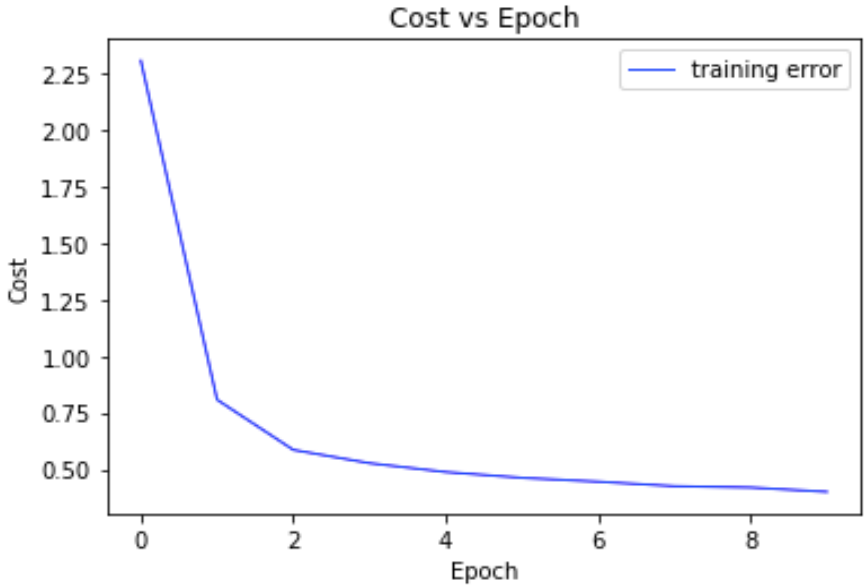
5 RESULTS

TASK 1:

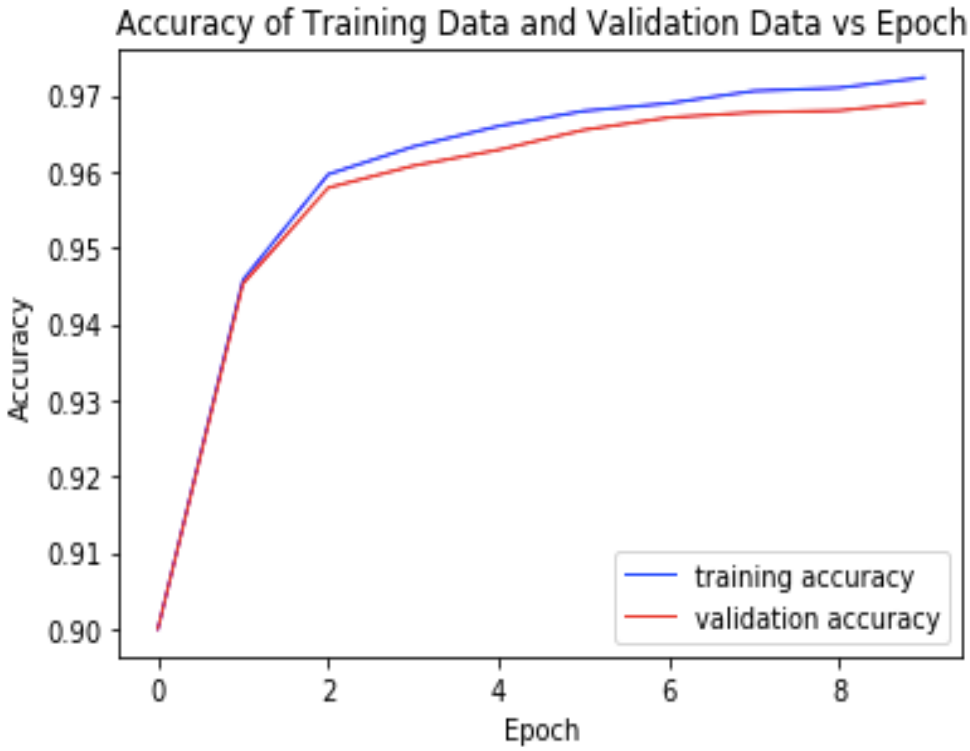
The prediction of my testdata set goes as follows:

a. Accuracy = 0.97 (approx.)
b. Precision = 0.99 (approx.)
c. Recall = 0.98 (approx.)

For testing the correctness of the values of hyperparameters, I have plotted a Training error function V/S epoch:



The second graph plots the accuracy of Training data and Validation data against epoch to check how similar my predictions are with respect to the training dataset.



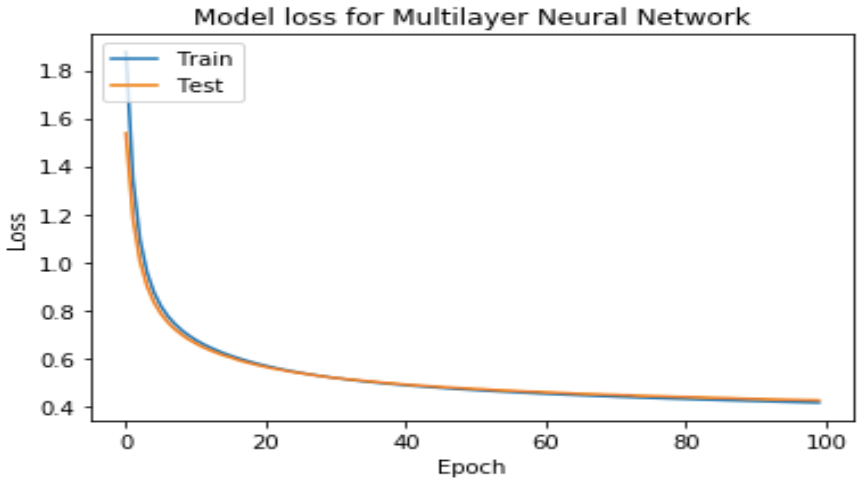
TASK 2:

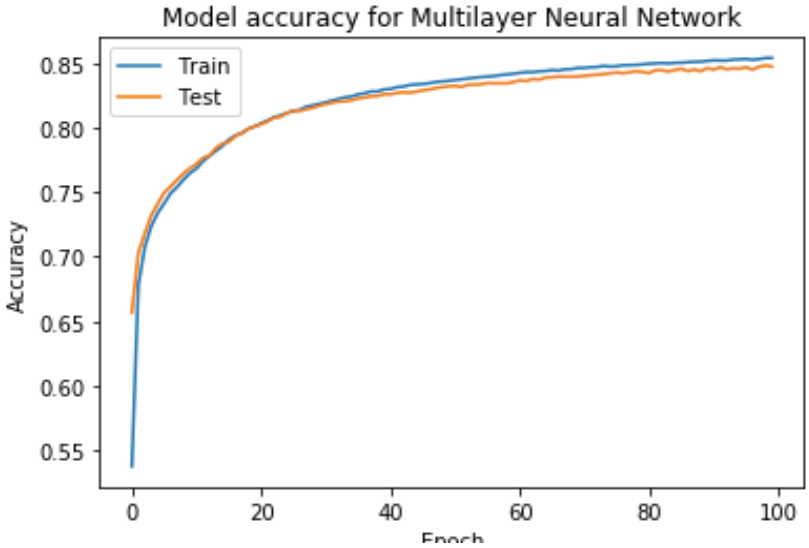
For a multilayered neural network, I have tested using a three, four and five layered network model of which the three-layered network produced the best accuracy measure: 0.84 (approx.)

```
TEST LOSS:0.4533223389148712
TEST ACCURACY: 0.8371999859809875
```

For testing the accuracy of the model, I have plotted two graphs:

1. Training error and validation error VS Epochs
2. Training accuracy and validation accuracy VS Epochs





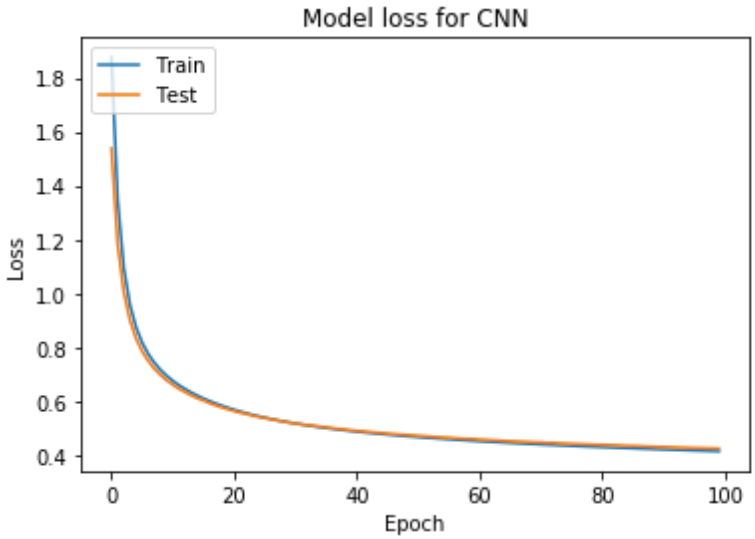
TASK 3:

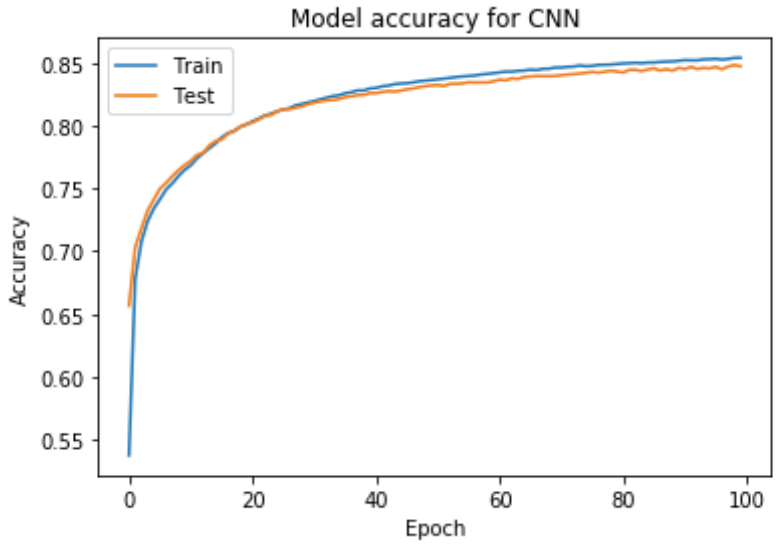
For a CNN, the testing accuracy has been found to be: 0.83 (approx.)

```
CNN TEST LOSS:0.4755798768043518
CNN TEST ACCURACY: 0.8294000029563904
```

For testing the accuracy of the model, I have plotted two graphs:

1. Training error and validation error VS Epochs
2. Training accuracy and validation accuracy VS Epochs





6 CONCLUSION

We have used the Fashion-MNST dataset to model our classification problem using neural networks. There were three tasks to perform: one with a single hidden layer in the neural network, one with multilayers and one with a concept of convolution of the input image features with a kernel of a fixed size. The models are evaluated on the basis of the 'accuracy' metric and the accuracy has been found to be different for each of the models. Accuracy depends on many factors: how well the hyperparameters are tuned, the learning rate and the number of epochs we are using to train our model. For the first task, I have used 1000 epochs, a learning rate of 0.9 and the tuned values of the weights and the biases to get an accuracy of around 97% for our model. While in the other two tasks, I have used around 100 epochs to train our model and have ended up getting an accuracy of around 84% which can be improved if I can train my model with more epochs over a period of time.

ACKNOWLEDGEMENTS:

I am extremely grateful to Prof. Sargur Srihari to have introduced us to the concepts and intricacies of Neural Networks and Convolutional Neural Networks and the content of the slides structured by him has contributed much to the detailed construction of the report. The other sources of information have been listed below:

[1] https://medium.com/

[2] https://towardsdatascience.com/

[3] https://www.wikipedia.org