REPORT : ASSIGNMENT 3
CSL7620: Machine Learning

NAME: DIPAN MANDAL
ROLL NO.: M23CSA009
M.TECH in ARTIFICIAL INTELLIGENCE

Google Colab notebook link:
https://colab.research.google.com/drive/12xp2-dljaW5NtHUzsXii5YDD2-w-hEEl?usp=sharing

In this assignment we were asked to design a neural network with multiple hidden layers from scratch. The neural network would perform a multi-class classification on the **fashion_mnist dataset** which contains 10 different classes.

The design of the neural network is given below:

Input layer : 784 (28*28), the dimension of the images in the dataset after flattening **Hidden layer 1:** 128 neurons
Hidden layer 2: 64 neurons
Hidden layer 3: 32 neurons
Output layer: 10 neurons i.e. the number of classes present in the dataset

All the hidden layers have **sigmoid** function as their activation function. The output layer contains **softmax** as its activation function.

We have randomly initialized the weights using the `np.random.randn` function and the biases are set to 1.

Q.9) The total trainable and non-trainable parameters.

The total number of trainable parameters are the weights and biases between each layer of the network.
i.e **784 * 128** weights and **128** biases between input and hidden layer 1 **128 * 64** weights and **64** biases between hidden layers 1 and 2
64 * 32 weights and **32** biases between hidden layers 2 and 3
32*10 weights and **10** biases between hidden layer 3 and output

layer. The total number comes out to be **111,146.**

The non-trainable parameters will be the hyperparameters such as l**earning rate, batch size, number of classes** etc.
The dataset had 70,000 images of size 28*28 pixels having values ranging from 0 to 255. As a part of data preprocessing we divided each value by the max value i.e 255

to **normalize the pixel values**. So, now the values range between 0 to 1.

Also we have encoded the y values (outputs) using **one-hot encoding** which is necessary while applying the categorical cross-entropy loss function on the neural network.

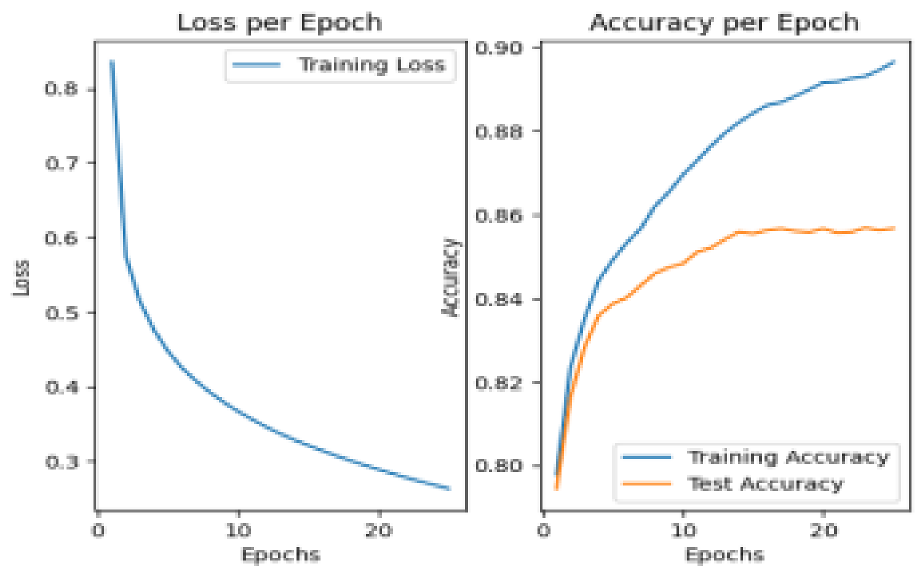
The whole training process revolves around 3 functions:
`forward_prop`, which generates an output when an input is provided `back_prop` which calculates the derivatives of the weights and biases in each layer with respect to the loss function
`gradient_descent` which we have used to update the parameters depending on the loss.

After getting the updated parameters, we also test the network on the test set to get the accuracy of the network.
We have used a set of **3 train-test splits** which we implemented using the `keras` library.
Also we have used **mini-batch gradient descent** with a batch size of 23 to make the training process more efficient.

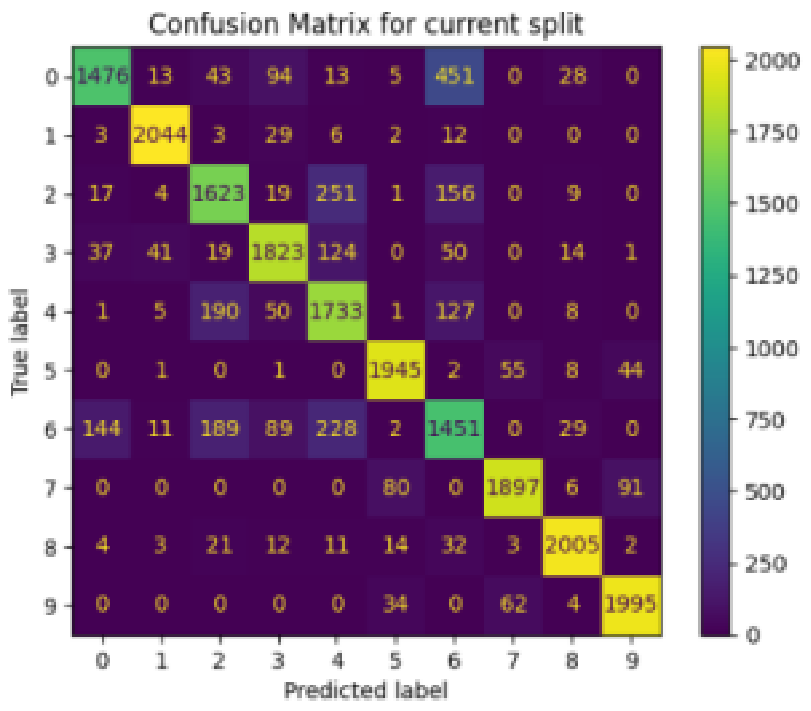
Here are the results that we have received after training the neural network for **25 epochs** with a **learning rate of 0.01** on 3 different train-test splits:

Train: 70% , Test: 30%:

Loss and accuracy per epoch:



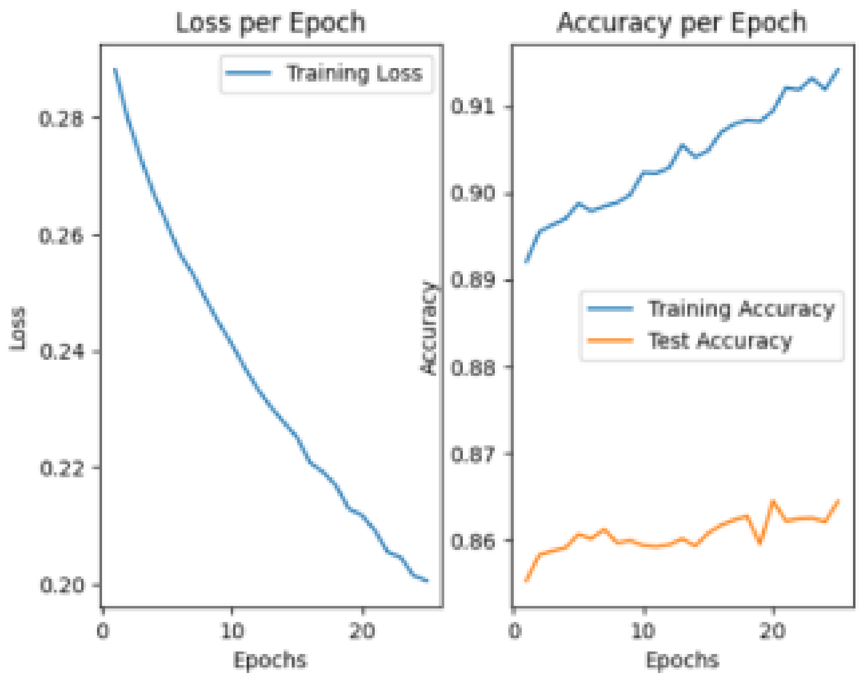
Confusion matrix:



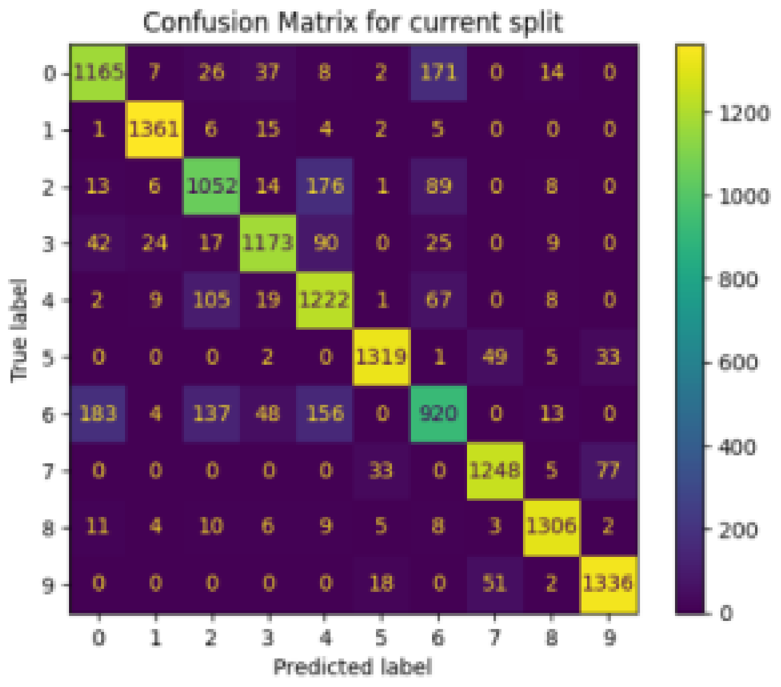
For 70-30 split, the model has training accuracy: 89.65% test accuracy: 85.67%

Train: 80%, test: 20%

Loss and accuracy per epoch:



Confusion matrix:

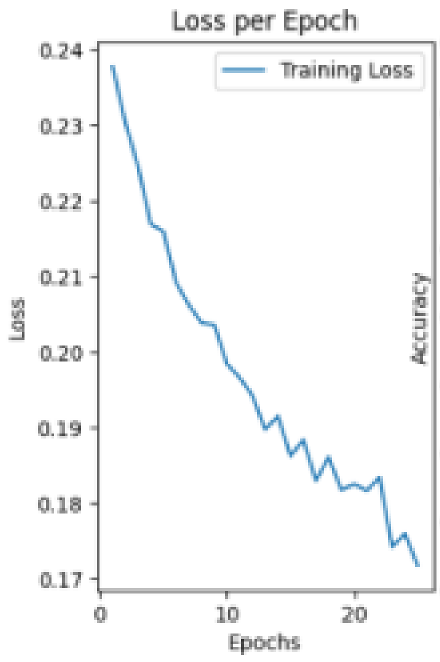


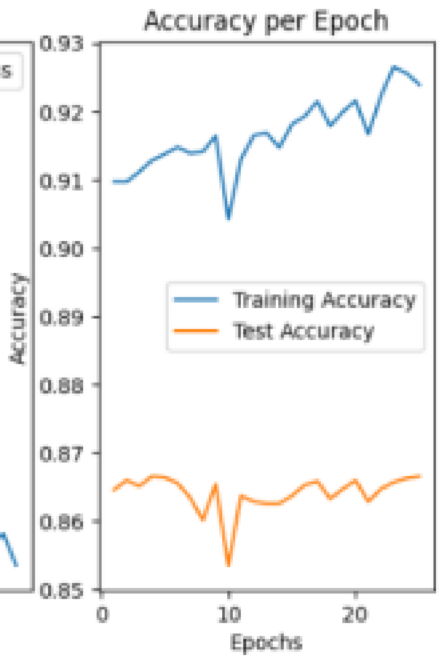
True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	1165	7	26	37	8	2	171	0	14	0
1	1	1361	6	15	4	2	5	0	0	0
2	13	6	1052	14	176	1	89	0	8	0
3	42	24	17	1173	90	0	25	0	9	0
4	2	9	105	19	1222	1	67	0	8	0
5	0	0	0	2	0	1319	1	49	5	33
6	183	4	137	48	156	0	920	0	13	0
7	0	0	0	0	0	33	0	1248	5	77
8	11	4	10	6	9	5	8	3	1306	2
9	0	0	0	0	0	18	0	51	2	1336

For the 80-20 split: training accuracy: 91.41% test accuracy: 86.44%

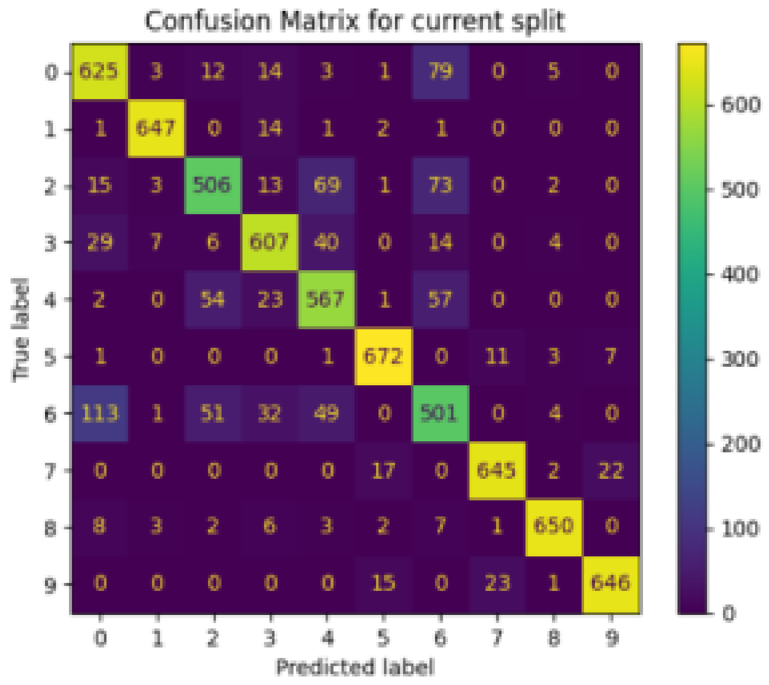
Train: 90%, test: 10%

Loss and accuracy per epoch:





Confusion matrix:



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	625	3	12	14	3	1	79	0	5	0
1	1	647	0	14	1	2	1	0	0	0
2	15	3	506	13	69	1	73	0	2	0
3	29	7	6	607	40	0	14	0	4	0
4	2	0	54	23	567	1	57	0	0	0
5	1	0	0	0	1	672	0	11	3	7
6	113	1	51	32	49	0	501	0	4	0
7	0	0	0	0	0	17	0	645	2	22
8	8	3	2	6	3	2	7	1	650	0
9	0	0	0	0	0	15	0	23	1	646

For the 90-10 split, the training accuracy: 92.39%, test accuracy: 86.66%

From the given data, it is evident that the model with **90-10** split performed the best.

To get better results we can experiment with the learning rate value, and increase the number of epochs. Also we can make the neural network deeper with more number of neurons per layer.