ML Assignment 3

Daksh Thapar
2018137

Q1.
All parts are present in the code itself.

Q2.
a)

Given architecture and parameters:

```
l=[784,256,128,64,10]
nn=MyNeuralNetwork(n_layers=len(l),layers_sizes=l,activation="sigmoid",wei
ght_init="normal",batch_size=200,learning_rate=0.1, num_epochs=100 )
```

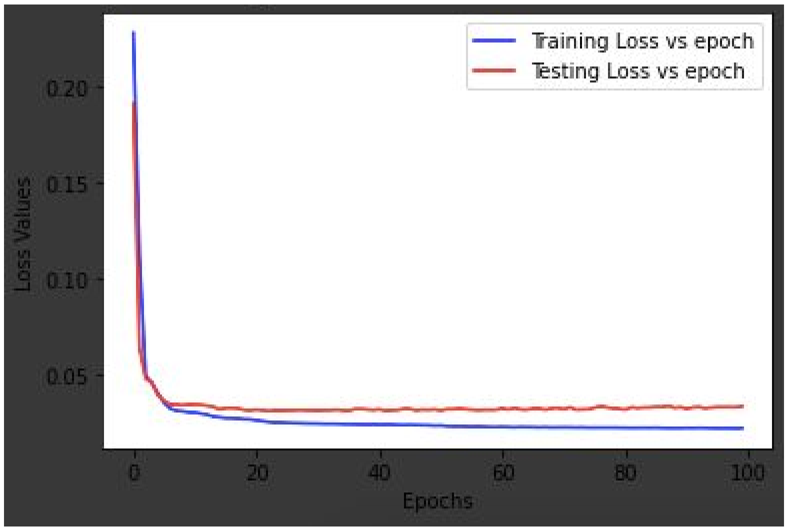
- As asked, I have used the Normal Initialization here with scaling factor=0.01
- I have saved the class objects of weights for all activations separately in the "**Weights"** folder.

- Test Accuracies for all activations (for the given parameters)-

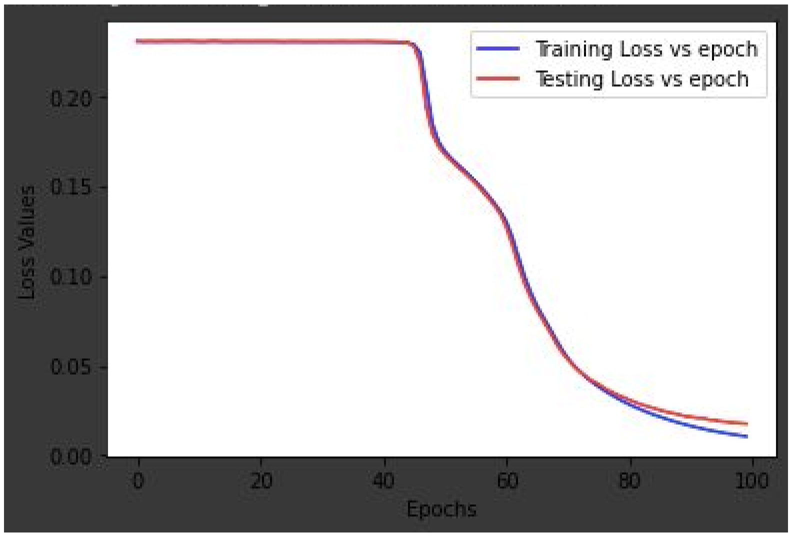
Activation Function	**Test Accuracy**
Linear	91.40%
Sigmoid	95.35%
Relu	96.88%
Tanh	96.84%

b) **Training error vs Epoch curve, Testing error vs Epoch curve**

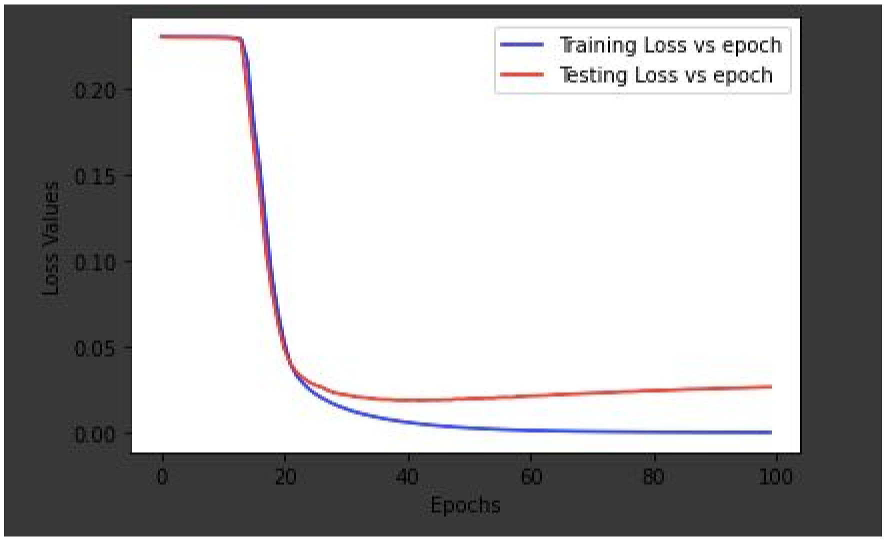
1. **Linear** activation function



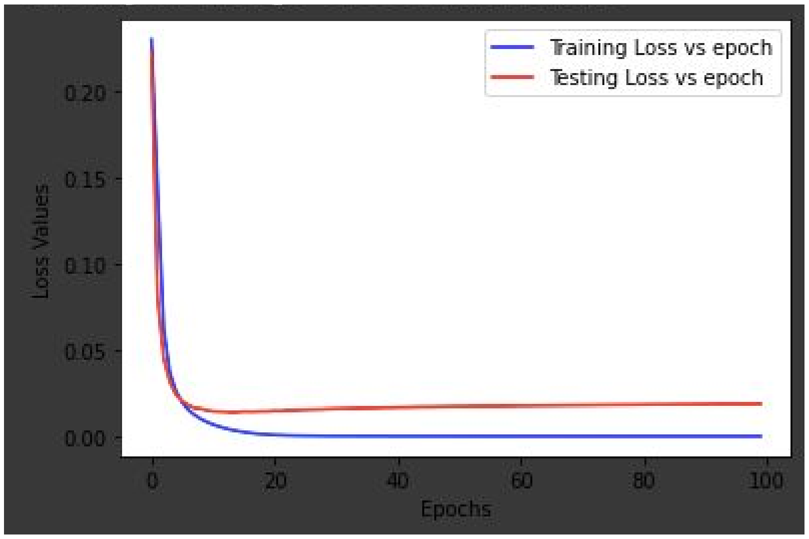
2. **Sigmoid** activation function



3. **Relu** activation function



4. **Tanh** activation function

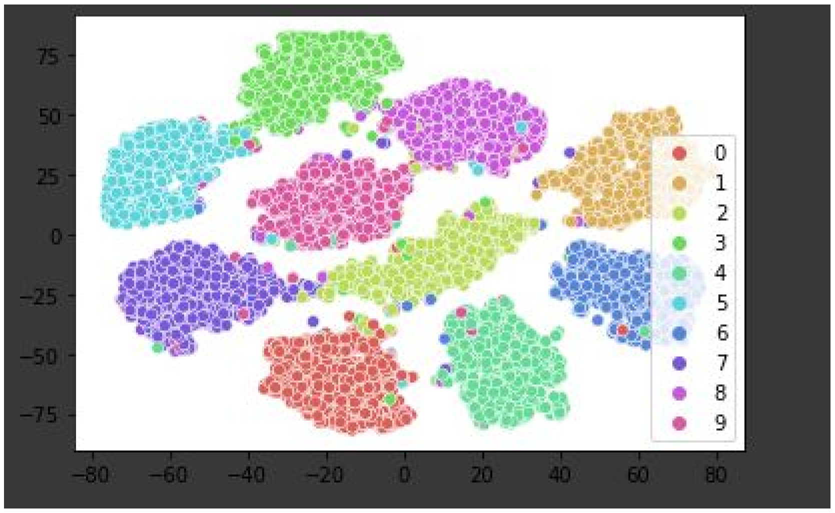


c)
Softmax activation function should be the activation function for the output layer. For the given question we have used cross entropy loss function and for that we need meaningful outputs in the range 0-1. Softmax function calculates the “probability distribution” of the event over 10 different events for our question. The main advantage of using Softmax is the output probabilities range which is 0-1 and the sum of all the probabilities will be equal to 1. If the softmax function used for a multi-classification model it returns the probabilities of each class and the target class will have the high probability.
d)

Total Layers of Neural network= 5 [784,256,128,64,10]
Hidden Layers of Neural network= 3 [256,128,10]

e)
Model with Highest Test Accuracy= **Relu**

Visualization for the features of the final hidden layer
Using tSNE-



10 colors denote the 10 class labels

f)
Comparing our test accuracies with Sklearn for the same parameters,

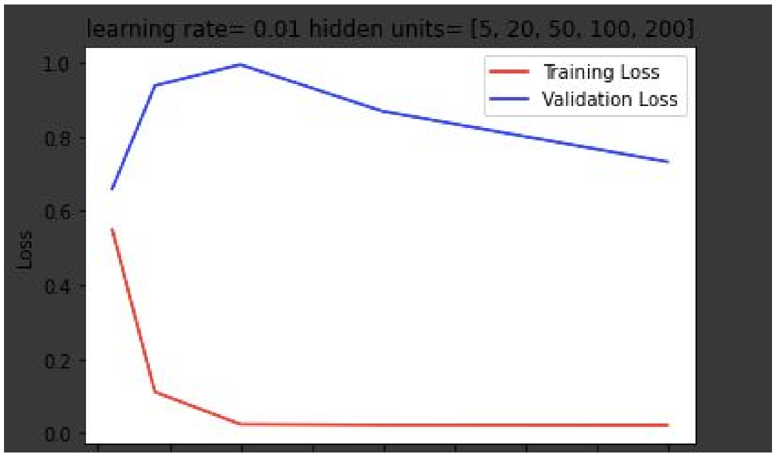
Activation Function	Test Accuracy (my implementation)	Test Accuracy (Sklearn)
Linear	90.40%	83.79%
Sigmoid	95.35%	96.35%
Relu	96.88%	96.51%
Tanh	96.83%	96.35%

On observation, we can observe a very high similarity in the testing accuracies when I compare my implementation to Sklearn's implementation. The small difference between accuracies might be due to the effect of seed of 'random_state' attribute. Overall the results are consistent with the implementation.

Q3.

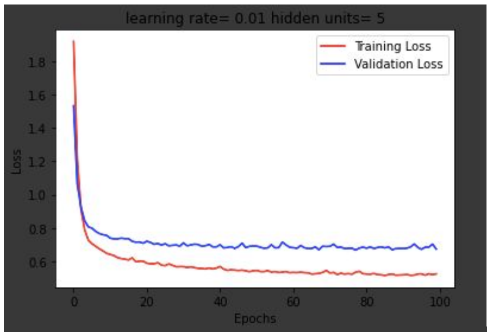
1.

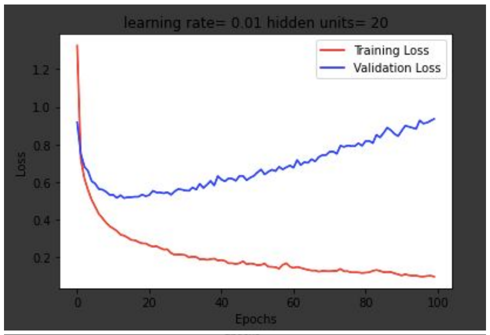
a) **Hidden units on X axis [5,20,50,100,200]**
Training and validation losses on Y axis

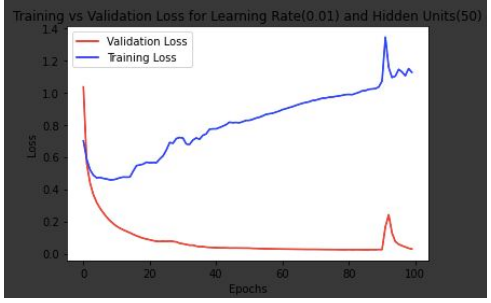


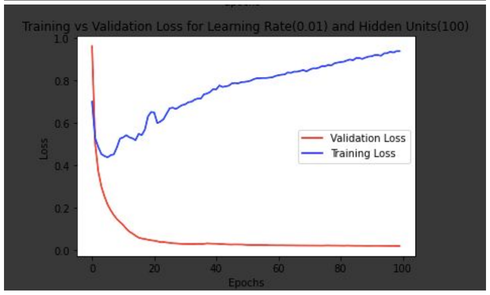
5 20 20 100 200

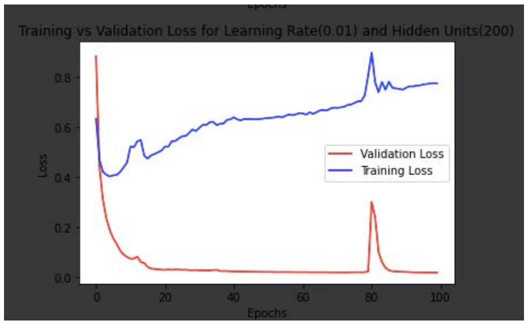
Epochs on X axis (5 cases corresponding to Hidden Units)











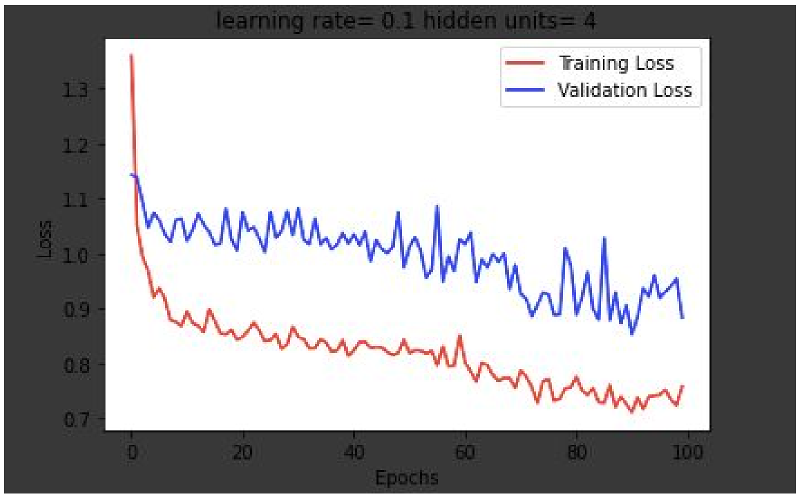
b) From the 1st figure, we can see that as we increase our epochs, the Validation loss increases to some epoch and then becomes constant (slightly decreasing) and our Training loss decreases steeply till some epoch and then becomes constant.

   We can also observe from the next 5 diagrams (loss vs epoch corresponding to each hidden units value) that as epochs increase, the Validation loss increases to some epoch and then stabilises whatsoever, while the Training loss decreases to some epoch and then doesn't change much if we increase the number of hidden units.

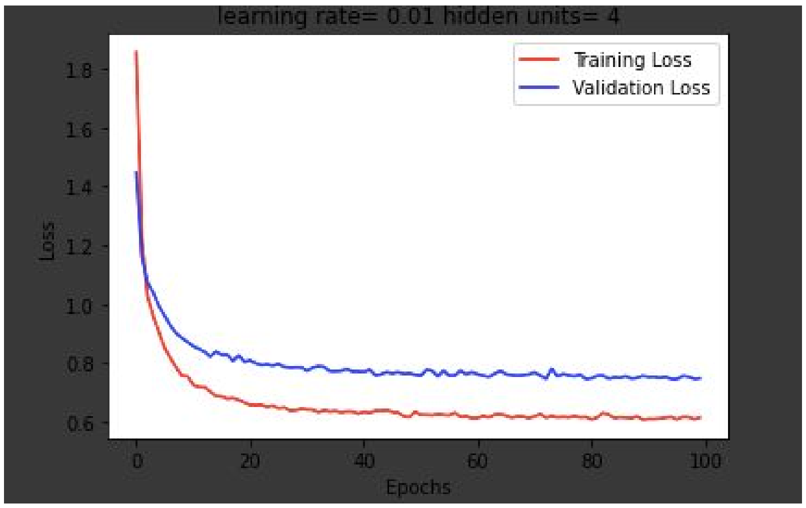
2.

a) **Learning Rate on X axis [0.1,0.01,0.001]**
   Training and validation losses on Y axis

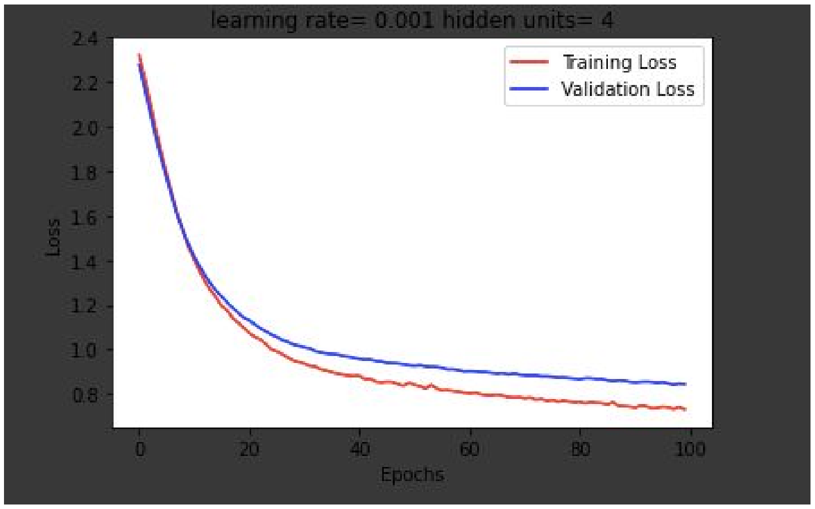
   Learning Rate=0.1



   Learning rate=0.01



Learning rate=0.001



b) We can clearly see that see that as 0.1 is too large a learning rate and hence the gradient descent algorithm works in such a way that there is an oscillation and the minima isn't reached as the value of the learning rate, or the step is too big that it doesn't quite arrive at the minima, hence the loss dances up and down.

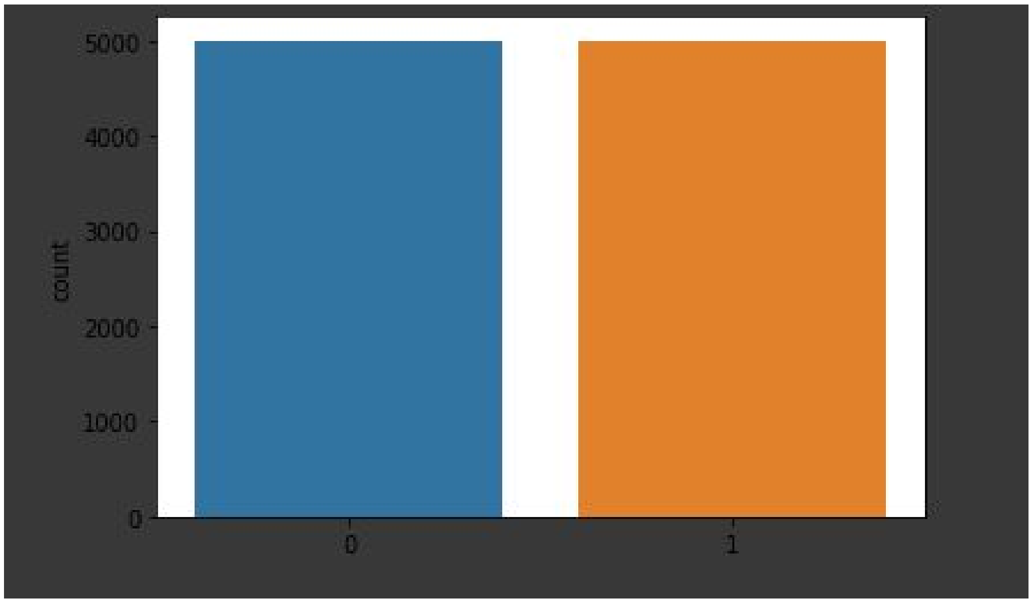
   For 0.01, we observe that the graph smoothens by a huge amount and this shows that 0.01 works much better than 0.1. Also, the cross entropy loss is able to converge better for 0.01 even though 0.1 takes a much larger step towards the minima but overshoots.

   For 0.001, we observe an even smoother graph but even at the end of 100 epochs the graph doesn't really converge to the minima. This is because 0.001 is too small a value that the loss takes so little steps while decreasing, and hence we observe a smooth but slowly converging plot.

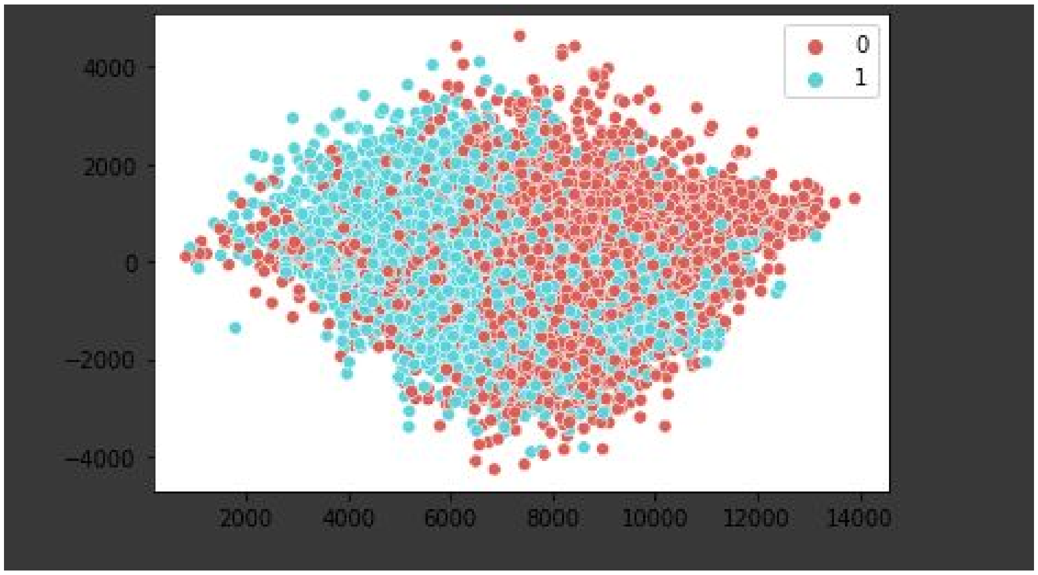
Q4.

1. Exploratory Data Analysis (EDA) on the CIFAR-10 dataset

   - Histogram of Class Distribution for Training Set (10000 samples)



   - Visualization using SVD (n_components=2) and scatter plot for plotting



2.

- Used AlexNet model from Pytorch (pretrained on ImageNet)
- Performed Preprocessing on the dataset
    - Resizing the data to 3 channels (i.e. 3072->32x32x3)
    - Normalizing the data
    - Resizing the image data, horizontal flip and converting to tensor object
- Further implementation is in the code itself.

3.

Trained my Neural Network Model having the following structure:

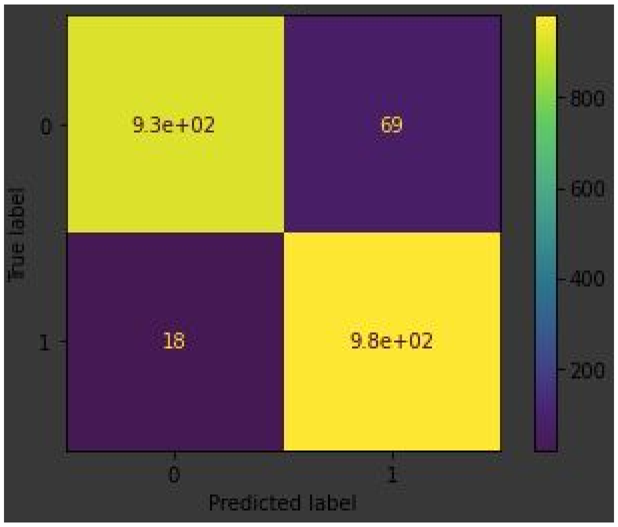
- input_layer_size=1000
- hidden_layer_1_size=512
- hidden_layer_2_size=256
- output_layer_size=2

Trained my model for epoch=10 for a batch size of iterators=64
Training Accuracy= 96.6062%

4. Using Sklearn's methods and metrics,

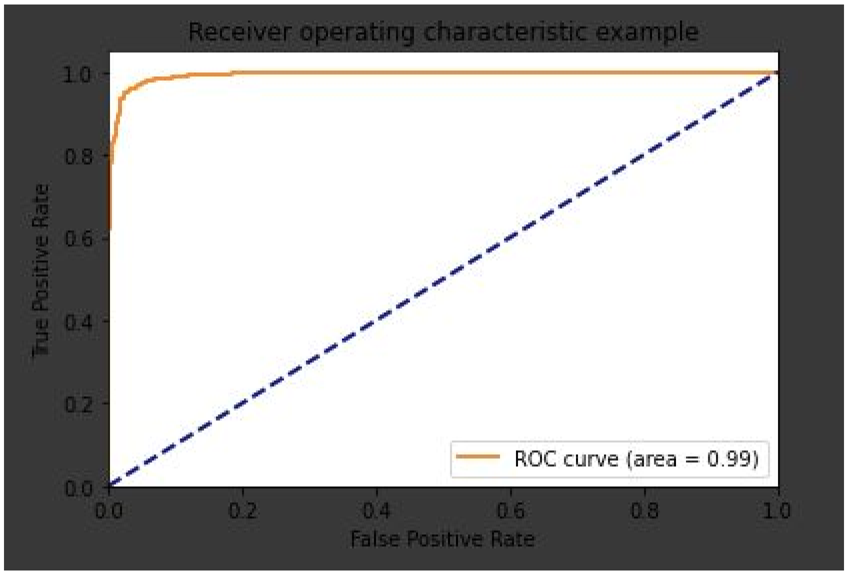
Testing Accuracy=95.712%

Confusion matrix=
[[955 45]
[41 959]]



(visualized using Sklearn ConfusionMatrixDisplay())

ROC Plot-


Receiver operating characteristic example
True Positive Rate
False Positive Rate
ROC curve (area = 0.99)
0.0
0.2
0.4
0.6
0.8
1.0