CSE574-D: Introduction to Machine Learning, Fall 2023

Assignment 2

Building Neural Networks and Convolutional Neural Networks

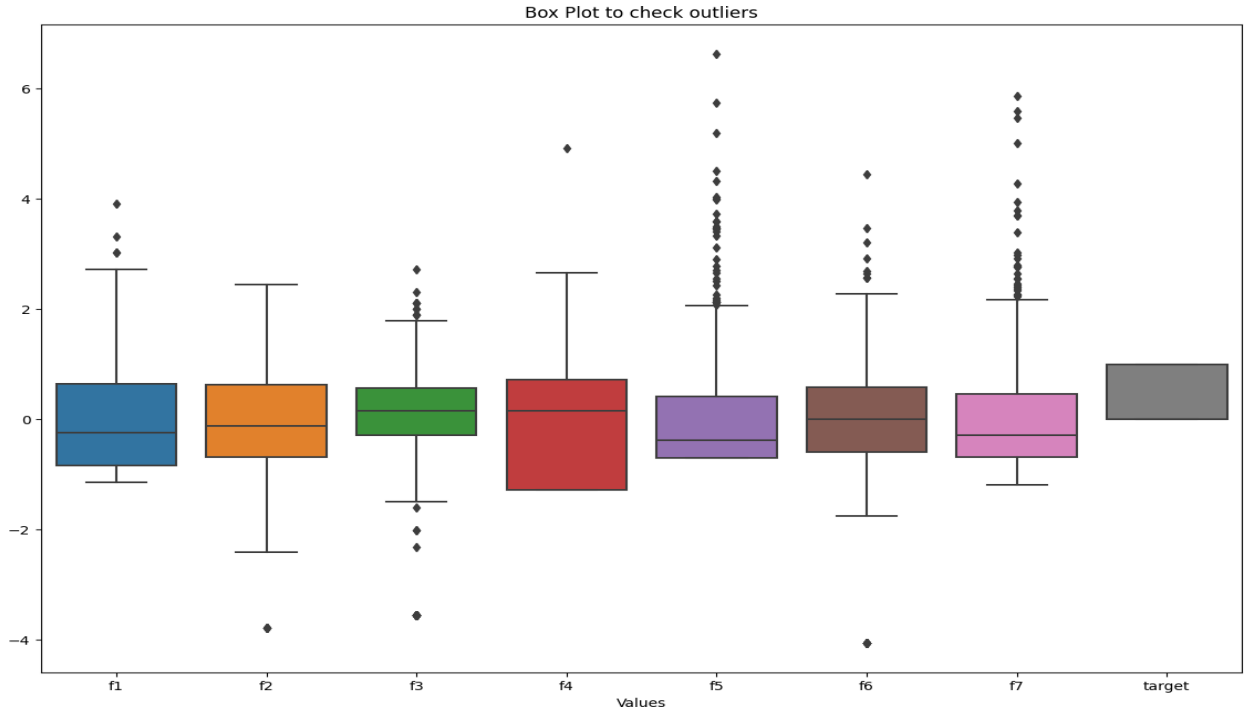
PART 1 :

1. Dataset and Main Statistics

- The dataset set given is a numerical dataset that contain 760 entries with 7 features.
- It contains binary classification data (0 or 1) as target.

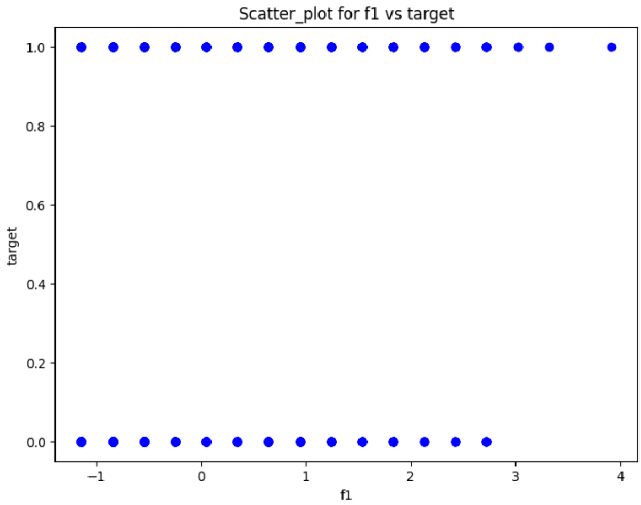
	f1	f2	f3	f4	f5	f6	f7	target
count	760.000000	760.000000	760.000000	760.000000	760.000000	760.000000	760.000000	760.000000
mean	3.834211	120.969737	69.119737	20.507895	80.234211	31.998684	0.473250	0.350000
std	3.364762	32.023301	19.446088	15.958029	115.581444	7.899724	0.332277	0.477284
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	0.000000
25%	1.000000	99.000000	63.500000	0.000000	0.000000	27.300000	0.243750	0.000000
50%	3.000000	117.000000	72.000000	23.000000	36.000000	32.000000	0.375500	0.000000
75%	6.000000	141.000000	80.000000	32.000000	128.250000	36.600000	0.627500	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	1.000000

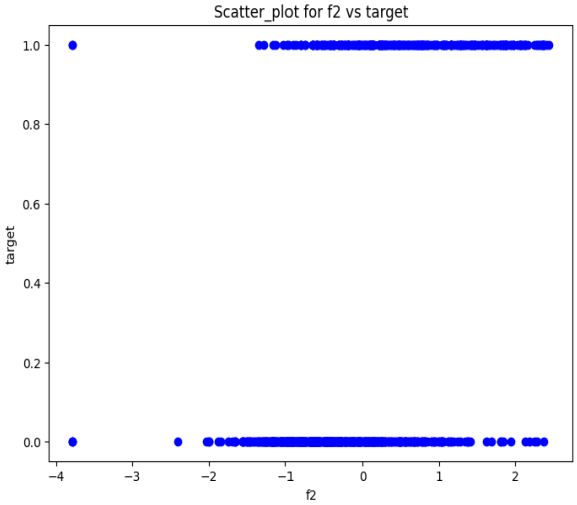
2. Visualization graphs

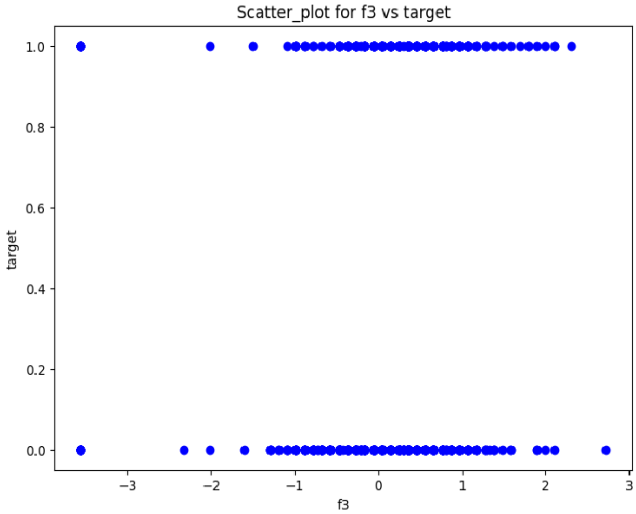


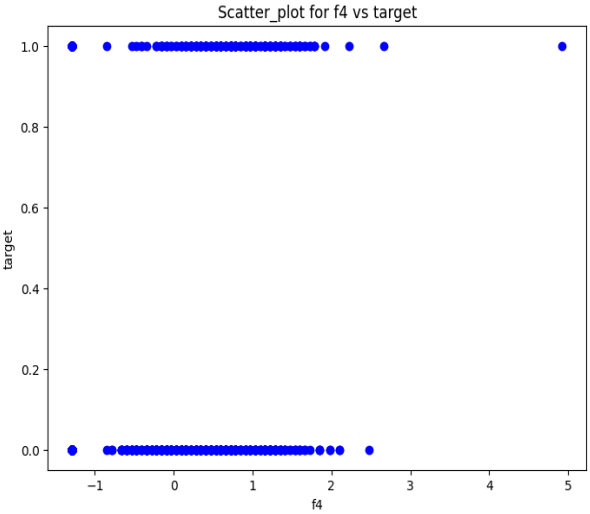
- In the **boxplot** visualization graph, as we pointed the datapoints we could see the outliers that are present in it, that lies out of the interquartile range.
- Here f5, f6,f7 feature have more outliers and all outliers in the features needs to be removed for better accuracy.

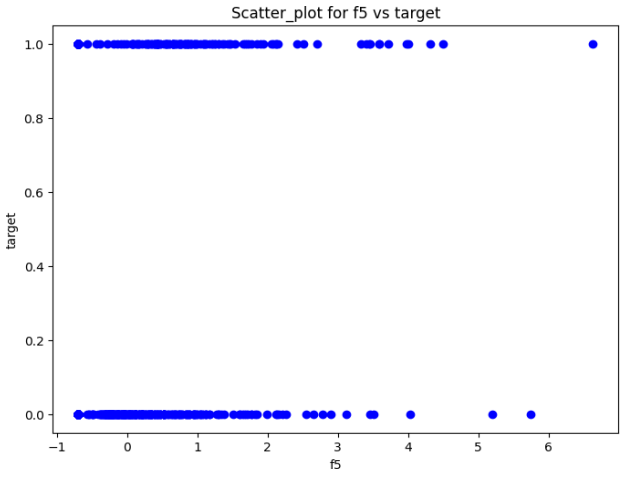
- On plotting the **correlation matrix**, we could see the closely correlated data with the target value. Here f2 is more corelated with the target than other features.

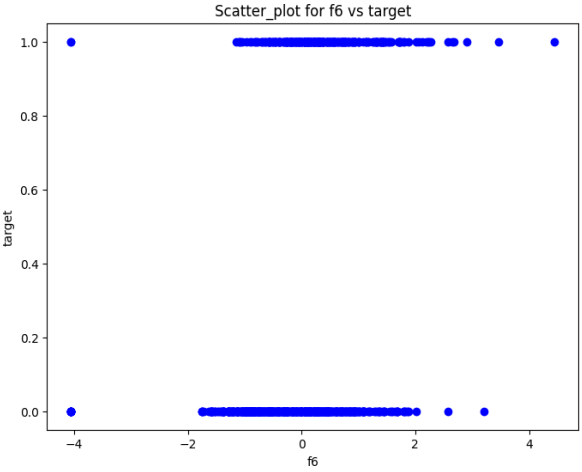

Scatter_plot for f1 vs target
target
f1

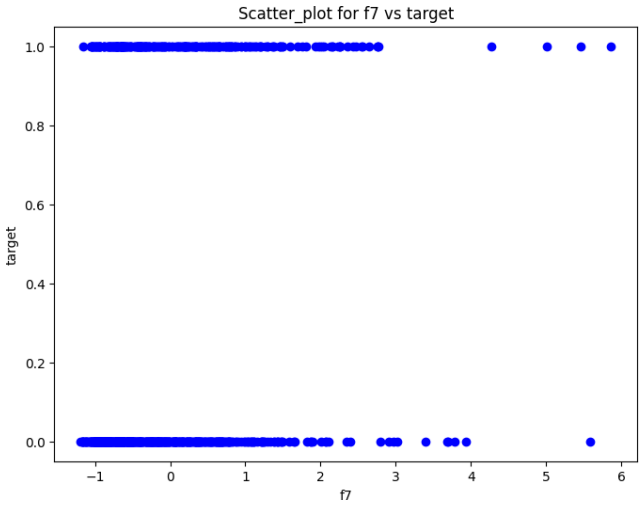

Scatter_plot for f2 vs target
target
f2


Scatter_plot for f3 vs target
target
f3


Scatter_plot for f4 vs target
target
f4


Scatter_plot for f5 vs target
target
f5


Scatter_plot for f6 vs target
target
f6



- On plotting the **scatterplot** with respect to the target variable, we could see how the data is scattered with respect to the target value.
- In the given dataset, f1 feature is more scattered than other features.
- And f4 and f6 are less scattered than any other features.

3. Preprocessing Techniques

- We dropped the non-numerical data in numerical features using the below code where alpha contains the upper and lower case alphabets

  df=df[~df.isin(alpha)]
- The outliers are removed using Interquartile range outlier removal technique.
- Used StandardScaler() to scale the data, so that the data wont vary much with outliers.

These above techniques helped to increase the accuracy of the model

4. Summary of Model

```
==========================================================================
Layer (type:depth-idx)                   Param #
==========================================================================
NeuralNetwork                            --
├─Sequential: 1-1                        --
│    └─Linear: 2-1                       1,024
│    └─ReLU: 2-2                         --
│    └─Linear: 2-3                       8,256
│    └─ReLU: 2-4                         --
│    └─Dropout: 2-5                      --
│    └─Linear: 2-6                       65
│    └─Sigmoid: 2-7                      --
==========================================================================
Total params: 9,345
Trainable params: 9,345
Non-trainable params: 0
==========================================================================
```

- We used 3 layered Neural Networks,
- In the first layer the input is the number of features(7) and output of this layer is 128 with RELU activation function
- In the second layer the input is 128, ie,(the output of 1st layer) and the output is 64 with RELU activation function
- Then we used the dropout of probability 0.4 in the second layer before passing to the activation function.
- In the third layer the input is 64,(output of 2nd layer) and the output is 1 with sigmoid activation function (as it's a binary classification)
- Then we are training the dataset in batches, where each batch contains 102 entries.
- We use Binary Cross Entropy Loss to predict the loss and Adam function for optimization

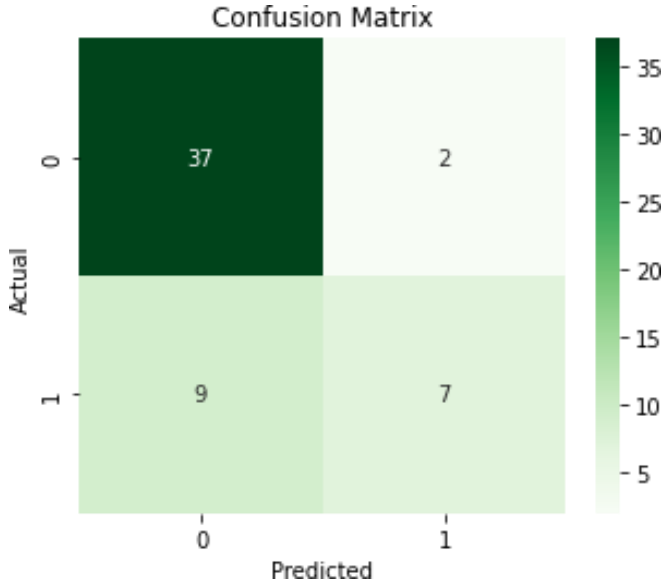
5. Performance metrics

- Time to Train : 22.124 seconds
- Accuracy : 80 %
- Precision : 0.797
- Recall : 0.800
- Fscore : 0.780

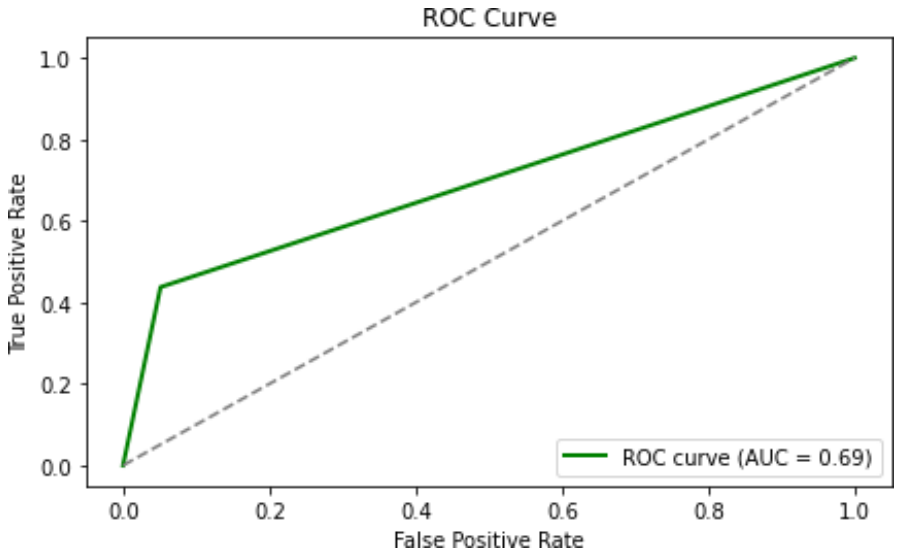
Analysis :

- As we analyzed, even though if we get high accuracy, if there are false positive or true negative predictions we were able to find it out in Precision, Recall and Fscore.
- Accuracy is the proportion of correct predictions.
- Precision is the proportion of true positives among predicted positives.
- Recall is the proportion of true positives among actual positives.
- Fscore is the combined measure of precision and recall.
- Although initially we were unable to achieve the expected accuracy, after making the slight changes we were able to achieve the accuracy of 80%.
- We tried with different hyperparameters and epochs such that the model is not overfitted.
- We used dropout to randomly drop some columns that helps to increase the accuracy.

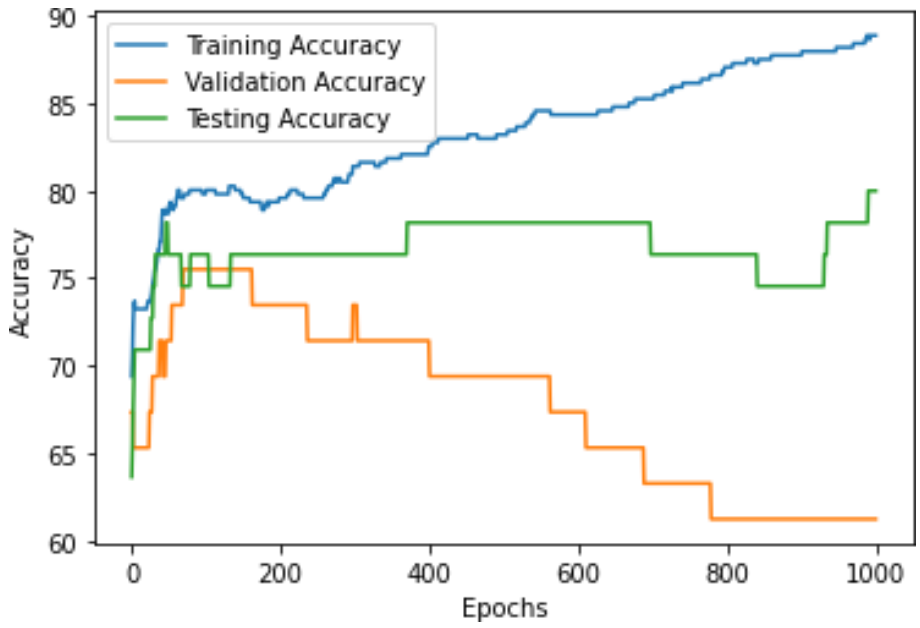
6. Analysis graphs



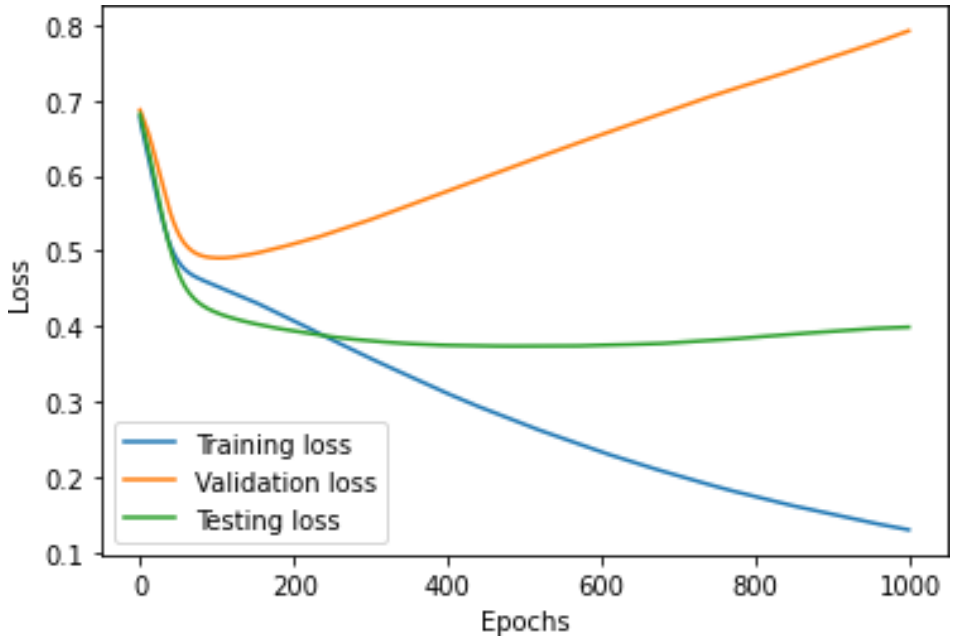
- On plotting the confusion matrix, we could determine the false negative, false positive, true positive and true negative values.
- Here we got the false positive value as 1 and false negative value as 38, true negative is 9 and true positive is 7.



- ROC curve helps to identify the trade off between true positive and false positive instances from the ground data and the predicted data.
- Here, for our predictions, we got the AUC as 0.69. It means the area under the curve is 69%



- In the graph we could see that the accuracy increases with increase in number of epochs.
- We are able to get the better accuracy for training data, where the training accuracy after 1000 epochs reached around 92% and the testing accuracy is 80%



- In the graph we are able to interpret that the loss decreases with increase in epochs.
- We could also infer that the training loss decreases much more than testing and validation losses. This could be due to little bit overfitting of the model to the training data.

PART 2

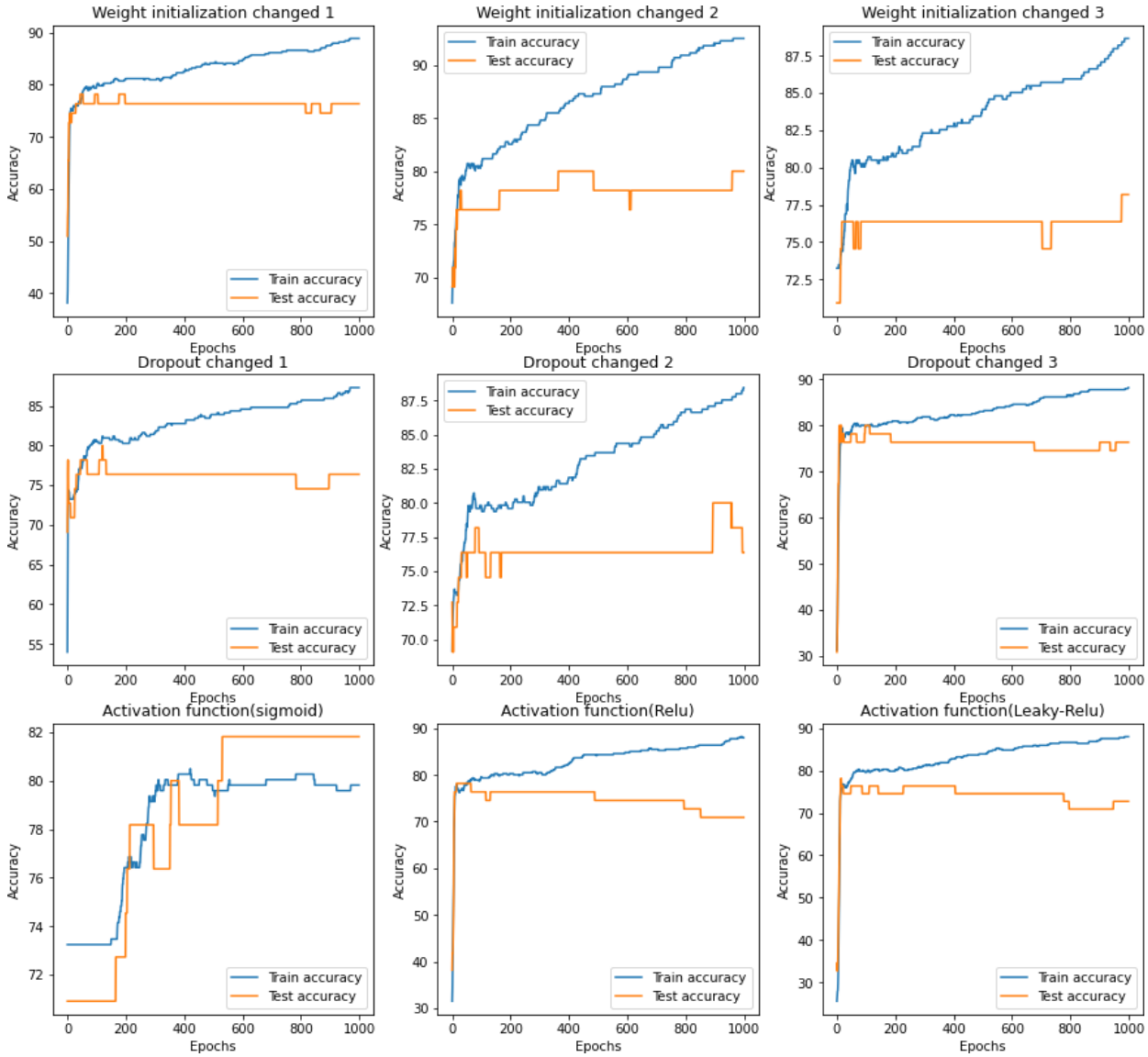
1.

	Setup 1	Test Accuracy	Setup 2	Test Accuracy	Setup 3	Test Accuracy
Dropout	0.3	76.36%	0.3	80%	0.3	78.18%
Optimizer	Adam		Adam		Adam	
Activation Function	ReLU		ReLU		ReLU	
Initializer	Uniform a=0.2 b=-0.2		Kaiming Mode=fan_in Non-linearity = relu		Xavier	

	Setup 1	Test Accuracy	Setup 2	Test Accuracy	Setup 3	Test Accuracy
Dropout	0.45	76.36%	0.5	76.36%	0.55	76.36%
Optimizer	Adam		Adam		Adam	
Activation Function	ReLU		ReLU		ReLU	
Initializer	Uniform a=0.2 b=-0.2		Uniform a=0.2 b=-0.2		Uniform a=0.2 b=-0.2	

	Setup 1	Test Accuracy	Setup 2	Test Accuracy	Setup 3	Test Accuracy
Dropout	0.3	81.81%	0.3	70.90%	0.3	72.72 %
Optimizer	Adam		Adam		Adam	
Activation Function	Sigmoid		ReLU		Leaky ReLU	
Initializer	Uniform a=0.2 b=-0.2		Uniform a=0.2 b=-0.2		Uniform a=0.2 b=-0.2	

2.



In graph 1, the training and testing accuracy is plotted for setup1 (parameters mentioned in Qn1), here we could observe that the training accuracy reached around 88% and testing accuracy is around 76%

In graph 2, the training and testing accuracy is plotted for setup2 (parameters mentioned in Qn2) , here we could observe that the training accuracy reached around 95% and testing accuracy is around 80%

In graph 3, the training and testing accuracy is plotted for setup3 (parameters mentioned in Qn3) , here we could observe that the training accuracy reached around 89% and testing accuracy is around 78%

In graph 4, the training and testing accuracy is plotted for setup4 (parameters mentioned in Qn4) , here we could observe that the training accuracy reached around 8% and testing accuracy is around 76%

In graph 5, the training and testing accuracy is plotted for setup5 (parameters mentioned in Qn5) , here we could observe that the training accuracy reached around 88% and testing accuracy is around 76%

In graph 6, the training and testing accuracy is plotted for setup6 (parameters mentioned in Qn6) , here we could observe that the training accuracy reached around 88% and testing accuracy is around 76%

In graph 7, the training and testing accuracy is plotted for setup7 (parameters mentioned in Qn7) , here we could observe that the training accuracy reached around 80% and testing accuracy is around 82%

In graph 8, the training and testing accuracy is plotted for setup8 (parameters mentioned in Qn8) , here we could observe that the training accuracy reached around 89% and testing accuracy is around 70%

In graph 9, the training and testing accuracy is plotted for setup9 (parameters mentioned in Qn9) , here we could observe that the training accuracy reached around 89% and testing accuracy is around 72%

3.

We have tried with dropout rate of 0.3 in order to drop the random nodes to reduce the computation complexity, with Adam optimizer and used ReLU as it takes only positive values and clips all negative values. Different weight initialization

In Setup1, we used uniform initializer with a=0.2 and b = -0.2 and got the accuracy of 76.36%

In Setup2, we used Kaiming initializer with mode=fan_in and Non linearity = relu and got the accuracy of 80%

In Setup3, we used Xavier initializer and got the accuracy of 78.18%

Then we have tried with different dropout rates in order to drop the random nodes to reduce the computation complexity, with Adam optimizer and used ReLU as it takes only positive values and clips only negative values and uniform initializer for all the different dropout rates.

In setup4, we used dropout rate of 0.45 and got the accuracy of 76.36%

In setup5, we used dropout rate of 0.5 and got the accuracy of 76.36%

In setup6, we used dropout rate of 0.55 and got the accuracy of 76.36%

Then we have tried with dropout rate of 0.3 in order to drop the random nodes to reduce the computation complexity, with Adam optimizer and uniform initializer for all the different dropout rates. Then we used different activation functions for the model

In setup7, we used Sigmoid and got the accuracy of 81.81%

In setup8, we used ReLU and got the accuracy of 70.90%

In setup9, we used Leaky ReLU and got the accuracy of 72.72%

4.

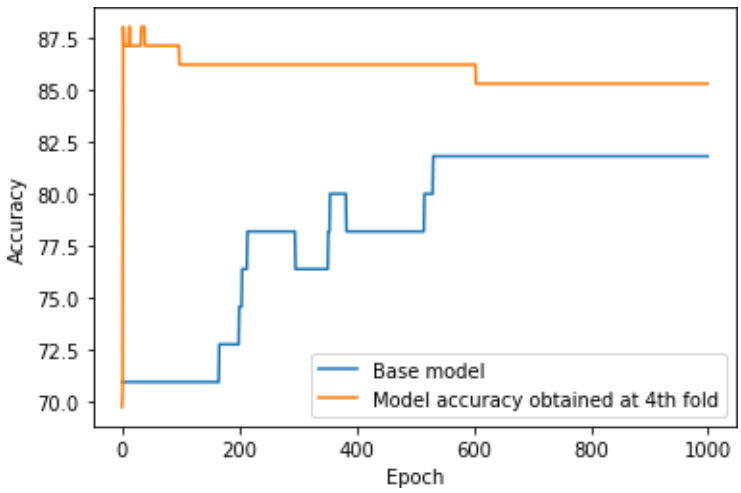
To improve the accuracy and training time we used four different methods

1. Early stopping

   This method mainly focuses in preventing overfitting, As it prevents training the model to a higher number of iterations once it finds there is a increase in testing loss for decrease in testing loss

   - We got the accuracy of 78.18
   - Time taken for training is 1.735 seconds

2. K-fold
   It helps to reduce the overfitting and training time. It splits the data into k folds and its not shuffled. Then the prediction is is learned till the previous (ie, k-1) model and the remaining folds that are left are used for testing.



   - We got the accuracy of 85.32 % in 4th fold
   - Time taken for training is 13.91 seconds

3. Learning rate scheduler

   The main motive is to make the learning rate adaptive to the gradient descended algorithm, ie, it decreases the learning rate with respect to number of epochs, this increasing the accuracy of the model and decreases the training time.

   - We got the accuracy of 80.0%
   - Time taken for training is 33.35 seconds

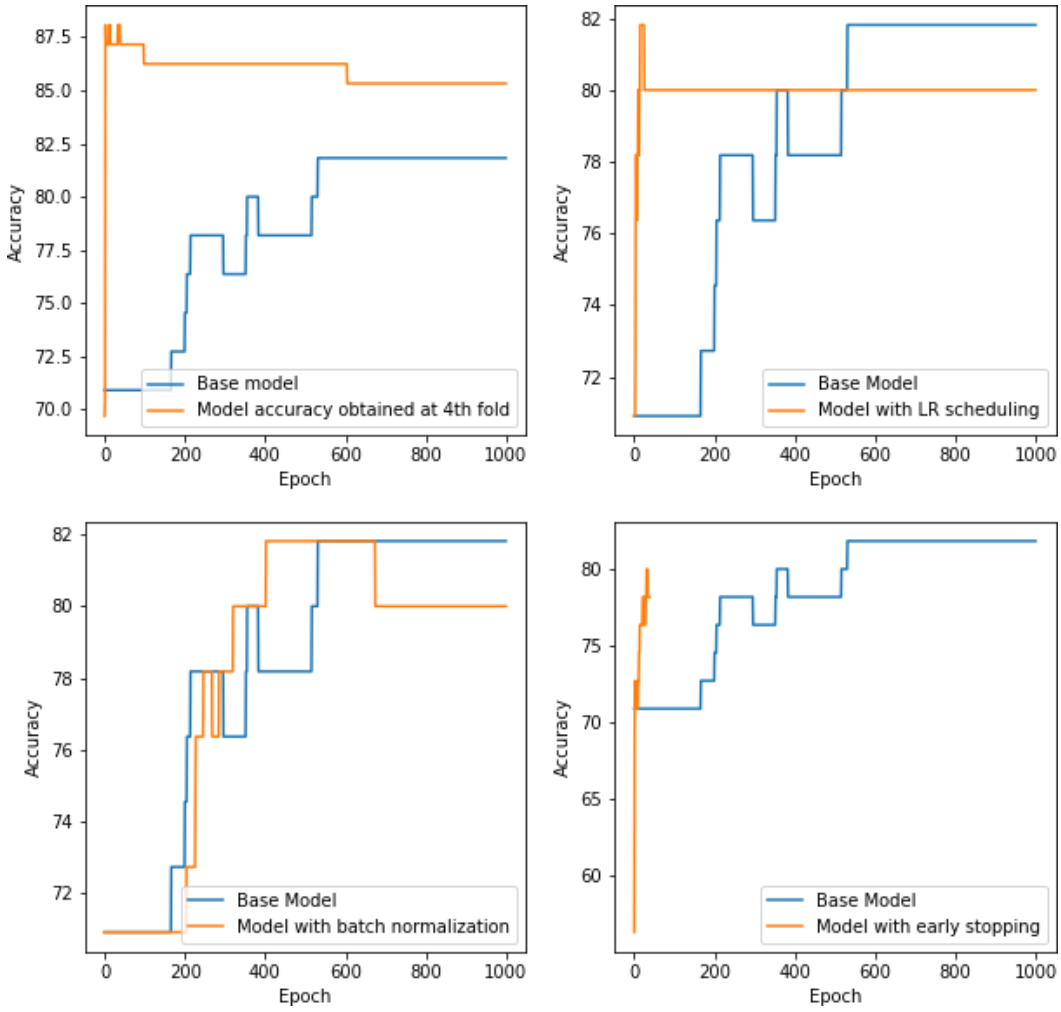
4. Batch normalization

   It helps in internal covariance shift by adding noise to the inputs of each layer.

   As it helps in preventing overfitting it increases the accuracy of model and consequently the training time is also decreased.

   - We got the accuracy of 80.0%
   - Time taken for training is 35.86 seconds

The graphs for all the methods are given below



Among all the methods tried k-fold cross validation method gave a good test accuracy at 4^{th} split.

PART 3

1. The given dataset is image classification dataset. It contains images of a to z and 0 to 9, as a total of 36 classes. We are encountering images of size 28 x 28.
classes=(0,1,2,3,4,5,6,7,8,9,'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z')
and each class have 2800 images with a total of 1,00,800 images

Total classes:36
Image size:torch.Size([1, 28, 28])
Class-0,count:2800
Class-1,count:2800
Class-2,count:2800
Class-3,count:2800
Class-4,count:2800
Class-5,count:2800
Class-6,count:2800
Class-7,count:2800
Class-8,count:2800
Class-9,count:2800
Class-A,count:2800
Class-B,count:2800
Class-C,count:2800
Class-D,count:2800
Class-E,count:2800
Class-F,count:2800
Class-G,count:2800
Class-H,count:2800
Class-I,count:2800
Class-J,count:2800
Class-K,count:2800
Class-L,count:2800
Class-M,count:2800
Class-N,count:2800
Class-O,count:2800
Class-P,count:2800
Class-Q,count:2800
Class-R,count:2800
Class-S,count:2800
Class-T,count:2800
Class-U,count:2800
Class-V,count:2800
Class-W,count:2800
Class-X,count:2800
Class-Y,count:2800
Class-Z,count:2800

3.

```
CNN( (C1): Conv2d(1, 24, kernel_size=(3, 3), stride=(1, 1)) (C2): Conv2d(24,
36, kernel_size=(3, 3), stride=(1, 1)) (fc1): Linear(in_features=900,
out_features=128, bias=True) (fc2): Linear(in_features=128, out_features=64,
bias=True) (fc3): Linear(in_features=64, out_features=36, bias=True) )
```

4. In order to improve the model's accuracy, have used the learning rate scheduler. In the due process converted the input images to gray scale image to yield better output. But we didn't not just stop with the convertion, we also went ahead and normalized the image.

5. Below are the performance metrics of our CNN model.

Accuracy : 90.70 %

Precision obtained in test dataset:0.91

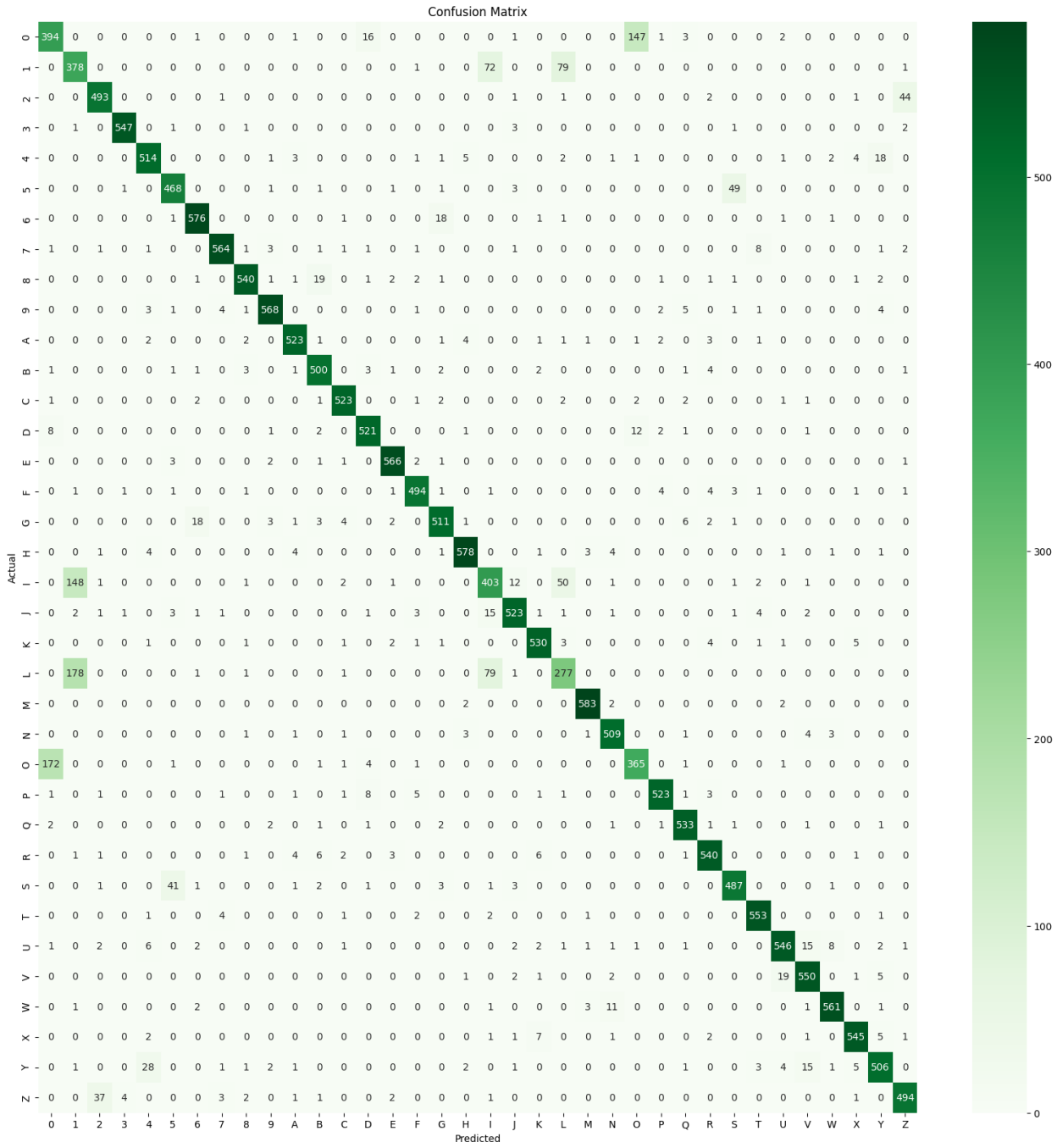
Recall obtained in test dataset:0.91

F_score obtained in test dataset:0.91

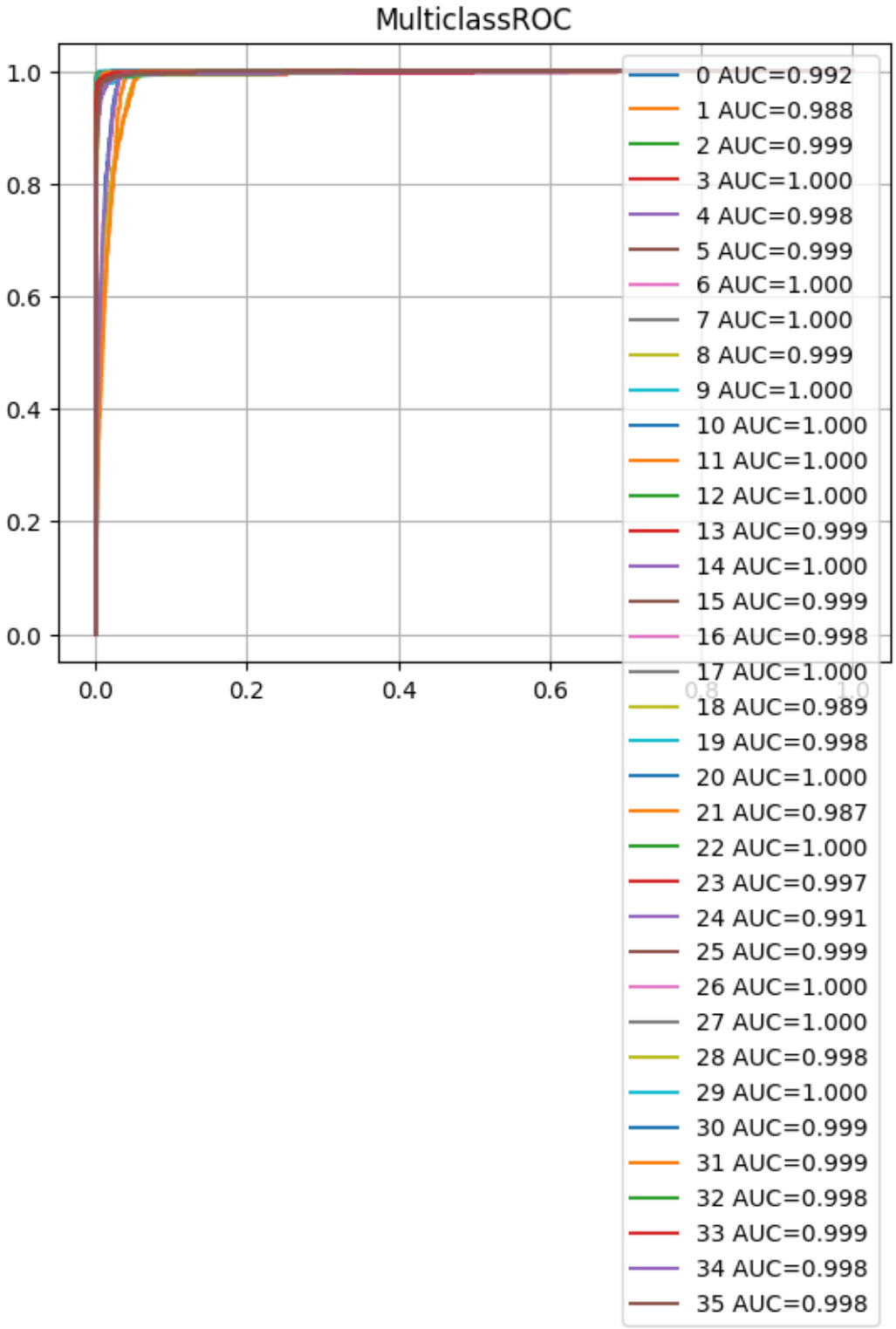
Support obtained in test dataset:None

From this I could infer that this is a good model as it gave an output of accuracy 90.70% followed by other metrics such as precision, recall, F_Score which also resulted in a good performance.

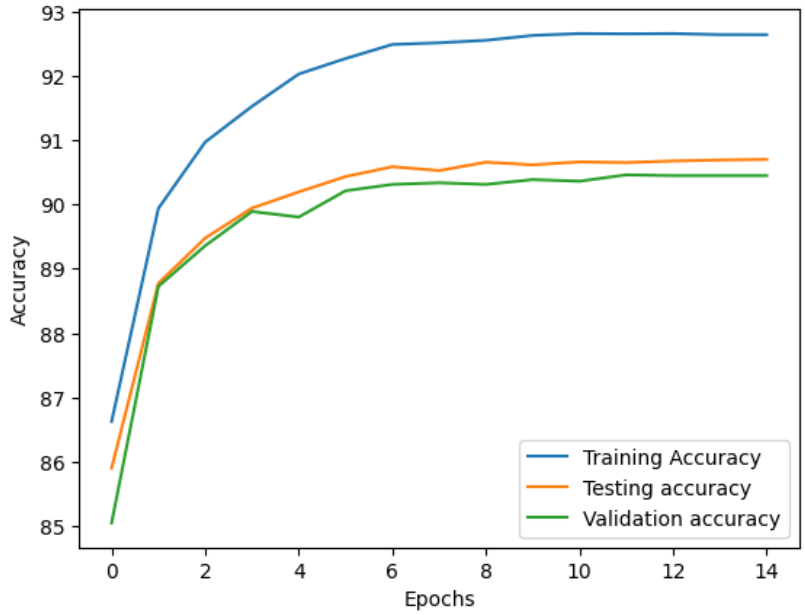
6.



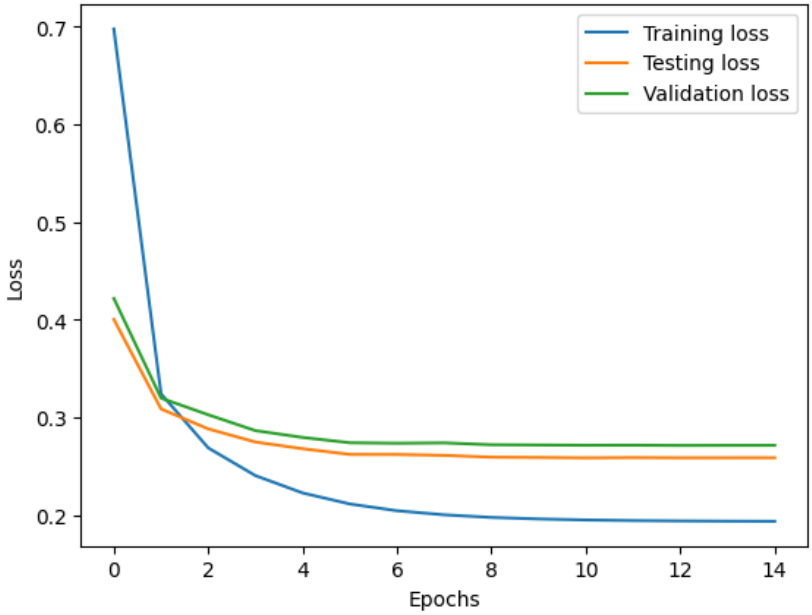
- On plotting the confusion matrix we were able to see the acutal data and the predicted data for each of the classes.
- Here we could see that higher the prediction is accurate, more darker the cell for each of the classes.



- ROC curve helps to identify the trade off between true positive and false positive instances from the ground data and the predicted data.
- Here we have drawn the curve for each classes to find the AUC for each class.



- In the graph we could see that the accuracy increases with increase in number of epochs.
- We are able to get the better accuracy for training, testing and validation data, where the training accuracy reached around 92.5% and the testing accuracy is around 90.5%



- In the graph we are able to interpret that the loss decreases with increase in epochs.
- We could also infer that the training loss decreases much and the testing and testing loss decreased to some extent with increase in the number of epochs.
- The training loss decreased to around 0.2 whereas the testing and validation loss had decreased to around 0.3

PART 4

1.Model details of VGG 11 is,

```
VGG11CustomInput(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (11): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (12): ReLU(inplace=True)
    (13): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (14): ReLU(inplace=True)
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (16): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): ReLU(inplace=True)
    (18): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (19): ReLU(inplace=True)
    (20): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=36, bias=True)
  )
)
```

2. The architecture of VGG11 differs from CNN architecture with respect to,

- VGG11 is deep neural network architecture, where as the model in part3 is not deep as compared to the constructed CNN model in part3
- In VGG11 maximum of 1000 classes can be used, whereas 36 classes used in CNN in part3
- In VGG11 the size of the image is resized to 224x224 with 3 channels RGB, whereas in CNN 28x28 image is used with 1 channel as a greyscale image
- In VGG one layer of padding is used ,whereas in cnn architecture in part3, no padding is used
- Adam optimizer is used in CNN network of part3, whereas SGD optimizer is used in VGG11

3. Performance metrics

Accuracy : 88.05 %

Precision: 0.884896319326304

Recall: 0.8824404761904762

F1 Score: 0.8826508364262361

Since the number of layers is less than that of VGG, the first CNN network was able to converge better. In higher layers network model like VGG there might be issue of vanishing gradient which can affect the performance of the model

4. Graphs

From our observations, In VGG11

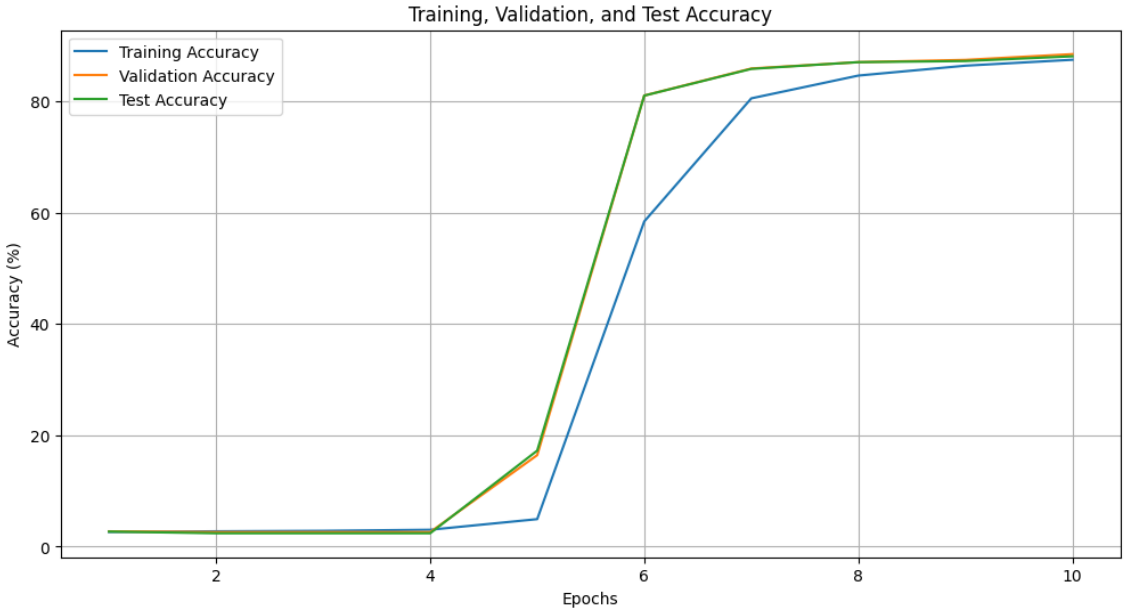
we could see that the accuracy does not increases initially and after certain number of iterations it increases rapidly and attains a saturation state.

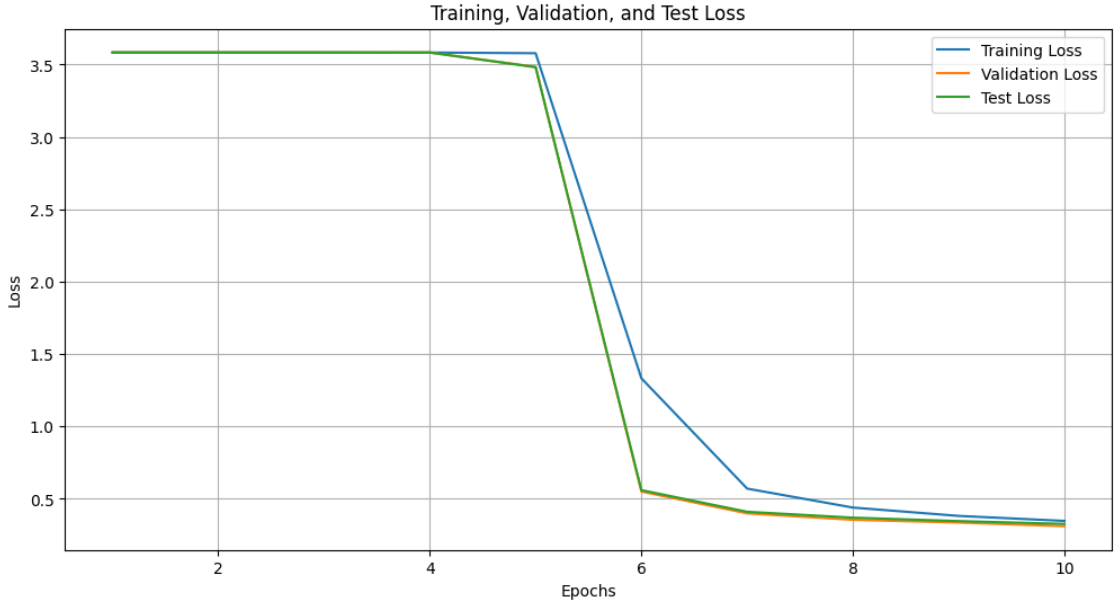
This convergence behavior may be due to the vanishing gradient of VGG.

We could see the same behavior for training, testing and validation data.

In our model in part3, the accuracy increases rapidly initially and reached the saturation state faster than compared to VGG.

And also the accuracy graph of VGG is shown below





From our observations, In VGG11 we could see that the loss does not decreases initially and after certain number of iterations it decreases rapidly and attains a saturation state. We were able to see the decrease in loss of all the training, testing validation data.
In our model in part3, the loss decreases rapidly initially and reached the saturation state faster than compared to VGG.
And also the graph for VGG is shown above

References :

https://scikit-learn.org/0.21/documentation.html
https://pytorch.org/docs/main/
https://stackoverflow.com/questions/58151507/why-pytorch-officially-use-mean-0-485-0-456-0-406-and-std-0-229-0-224-0-2