Deep Learning- CSE641

Assignment - 2 [Part 2]

Name: Akanksha Shrimal **Roll No:** MT20055
Name: Vaibhav Goswami **Roll No:** MT20018
Name: Shivam Sharma **Roll No**: MT20121

FILES SUBMITTED : submitted .py file , .ipynb file and readme.pdf and output.pdf

ASSUMPTIONS :

- The fit function of the network class takes one hot encoded labels and preprocessed data.
- To display the Training and Testing accuracy during each epoch we have scored training dataset only on the first 1000 samples, to reduce the lag for each epoch.
- The Dropout passed to the neural network must be an array stating the drop out probability of each layer in the network (including input and output layers) As we want a classification for 10 classes so dropout probability for output must always be passed 1 and that of input layer must be very close to 0.

UTILITY FUNCTIONS :

1. Save and Load models using Pickle

```
# Saving and Loading models using pickle
def save(filename, obj):
 with open(filename, 'wb') as handle:
     pickle.dump(obj, handle, protocol=pickle.HIGHEST_PROTOCOL)

def load(filename):
 with open(filename, 'rb') as handle:
     return pickle.load(handle)
```

2. Pre Processing data - Normalization and One Hot Encoding

```
# Utility function to normalize the data and one hot encode the
labels
def pre_process_data(train_x, train_y, test_x, test_y):
   # Normalize
   train_x = train_x / 255.
   test_x = test_x / 255.
   enc = OneHotEncoder(sparse=False, categories='auto')
   train_y = enc.fit_transform(train_y.reshape(len(train_y), -1))
   test_y = enc.transform(test_y.reshape(len(test_y), -1))
   return train_x, train_y, test_x, test_y
```

3. Plotting functions

```
# function to plot double line graph
# Plot double line using X1 , Y1 and X2 , Y2
def plot_double_line_graph(X1,Y1,label1 ,X2 ,Y2,label2
,title,y_name):
 fig = plt.figure(figsize=(7,5))
 plt.subplot(111)
  plt.plot(X1,Y1 ,label=label1 ,marker = "x" , color="blue")
 plt.plot(X2, Y2 , label=label2 ,marker = "x" , color="red")
 plt.title(title)
```

```
 plt.ylabel(y_name)
 plt.xlabel('Epochs')
 plt.legend( loc='upper left',prop={'size': 13})
 plt.show()


# Plot simgle line using X1 , Y1
def plot_single_line_graph(X1,Y1,label1, title,name_y):
 fig = plt.figure(figsize=(7,5))
 plt.subplot(111)
  plt.plot(X1,Y1 ,label=label1 ,marker = "x" , color="blue")
 plt.title(title)

 plt.ylabel(name_y)
 plt.xlabel('Epochs')
 plt.legend( loc='lower right',prop={'size': 13})
 plt.show()
```

4. Plotting ROC Curve

```
# (7,7)
#https://www.dlology.com/blog/simple-guide-on-how-to-generate-roc-
plot-for-keras-classifier/
def plot_roc(classes, y_test, y_score, figsize=(7,7)):
 n_classes = len(classes)
 # Plot linewidth.
 lw = 2

 # Compute ROC curve and ROC area for each class
 fpr = dict()
 tpr = dict()
 roc_auc = dict()
 for i in range(n_classes):
     fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
     roc_auc[i] = auc(fpr[i], tpr[i])

 # Compute micro-average ROC curve and ROC area
 fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(),
y_score.ravel())
```

```
 roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

 # Compute macro-average ROC curve and ROC area

 # First aggregate all false positive rates
 all_fpr = np.unique(np.concatenate([fpr[i] for i in
range(n_classes)]))

 # Then interpolate all ROC curves at this points
 mean_tpr = np.zeros_like(all_fpr)
 for i in range(n_classes):
     mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])

 # Finally average it and compute AUC
 mean_tpr /= n_classes

 fpr["macro"] = all_fpr
 tpr["macro"] = mean_tpr
 roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

 # Plot all ROC curves
 plt.figure(1)
 plt.figure(figsize=figsize)
 plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)

 plt.plot(fpr["macro"], tpr["macro"],
         label='macro-average ROC curve (area = {0:0.2f})'
               ''.format(roc_auc["macro"]),
         color='navy', linestyle=':', linewidth=4)

 colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
 for i, color in zip(range(n_classes), colors):
     plt.plot(fpr[i], tpr[i], color=color, lw=lw,
              label='ROC curve of class {0} (area = {1:0.2f})'
              ''.format(i, roc_auc[i]))

 plt.plot([0, 1], [0, 1], 'k--', lw=lw)
```

```
 plt.xlim([0.0, 1.0])
 plt.ylim([0.0, 1.05])
 plt.xlabel('False Positive Rate')
 plt.ylabel('True Positive Rate')
 plt.title('Some extension of Receiver operating characteristic to
multi-class')
 plt.legend(loc="lower right")
 plt.show()
```

6. Plotting Confusion Matrix

```
def confusion_matrix_find(y, y_hat, nclasses):

   y = y.astype(np.int64)
   y_hat = y_hat.astype(np.int64)

   conf_mat = np.zeros((nclasses, nclasses))

   for i in range(y_hat.shape[0]):
       true, pred = y[i], y_hat[i]
       conf_mat[true, pred] += 1
   return conf_mat
# Plotting confusion matrix
def confusion_matrix_plot(cm, classes, title='Confusion matrix',
cmap=plt.cm.Blues, figsize=(7,7), path=None, filename=None):

   cm = cm.astype(np.int64)
   plt.figure(figsize=figsize)
   plt.imshow(cm, interpolation='nearest', cmap=cmap)
   plt.title(title)
   plt.colorbar()
   tick_marks = np.arange(len(classes))
   plt.xticks(tick_marks, classes, rotation=45)
   plt.yticks(tick_marks, classes)

   fmt = 'd'
```

```
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

Best Model from Part-1:

Network = [784,128,24,10]
Learning Rate = 0.01
Epochs = 100
Activation = Tanh
Initialization = Random
Optimizer = Adam

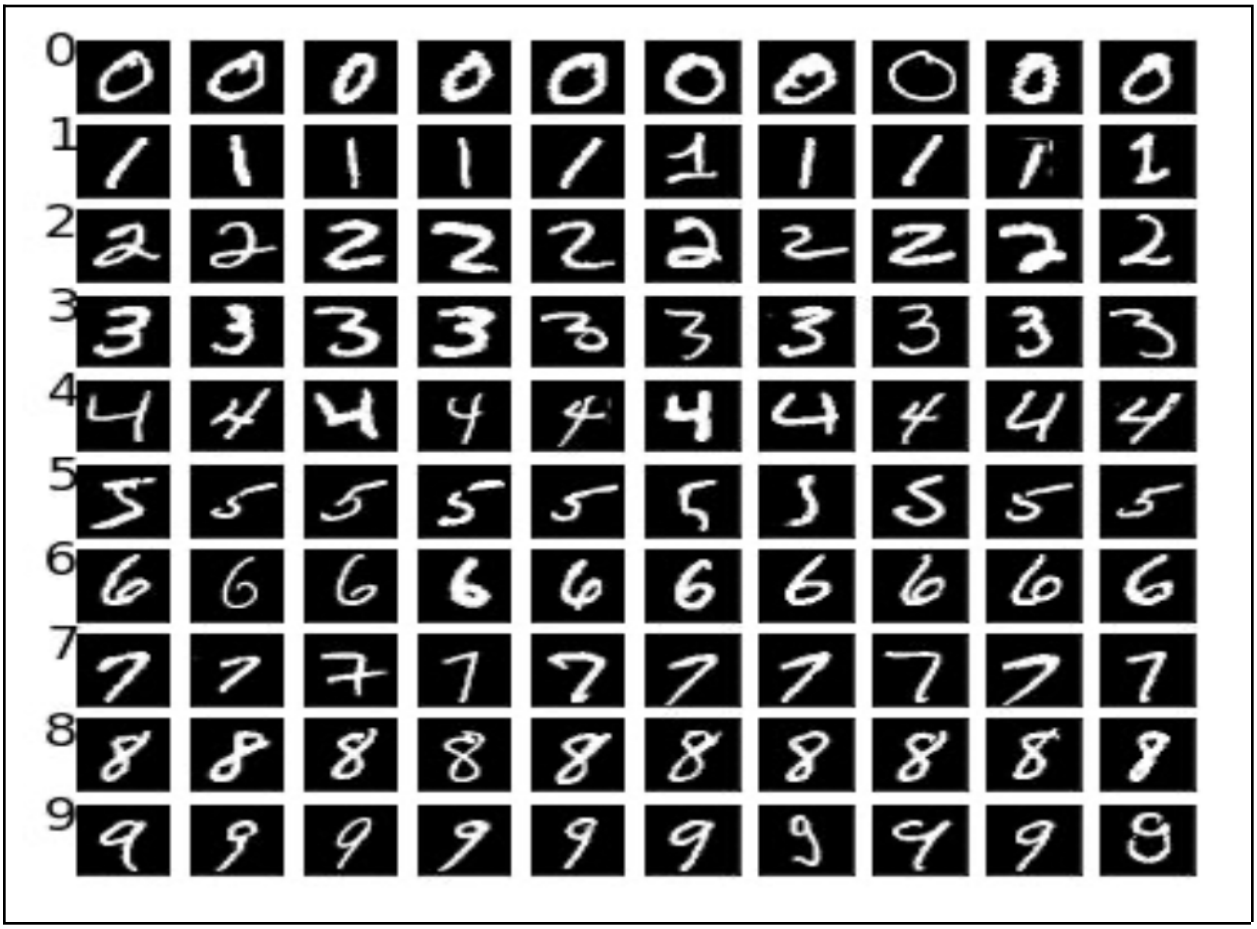
This model is further used in part too. Though originally Adam was trained for 450 epochs but it was getting overfitted so for convenience we used 100 epocs.

2. Weight Initialization:

1.1 About dataset

A brief description of the dataset : MNIST

No of samples	10,000 (Training) , 2000 (Testing)
dimensions of each sample	(28,28)
Unique Labels	0,1,2,3,4,5,6,7,8,9



1.2 Preprocessing dataset

Pre Processing Data

1. PIL Image is converted to np array
2. Image data is normalised using standard scalar form numpy.
3. Labels are one-hot encoded for multi-classification.

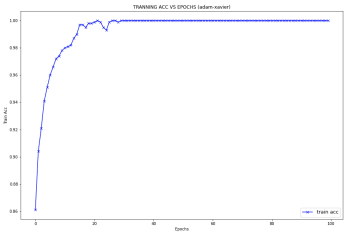
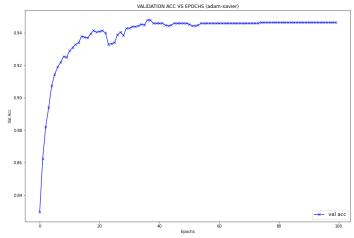
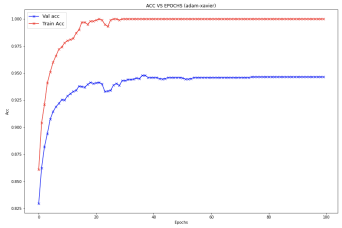
1.3 Results and Observation from Weight Initialization

1. Xavier

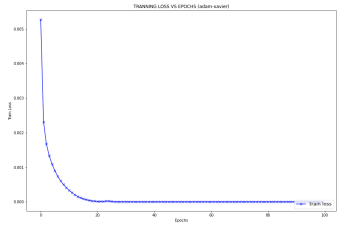
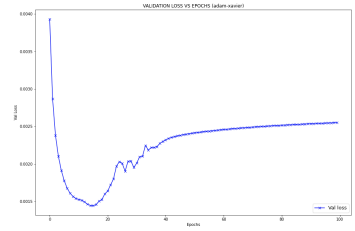
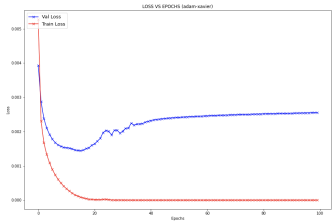
Plots

From the plots it can be observed that after approximately 19 epochs model starts overfitting and learning the training and and doing less generalization. At this time Xavier gives the accuracy which is training 98% and testing 94% (approx). Also we know Xavier initialization works very well for tanh and He is preferred for Relu.

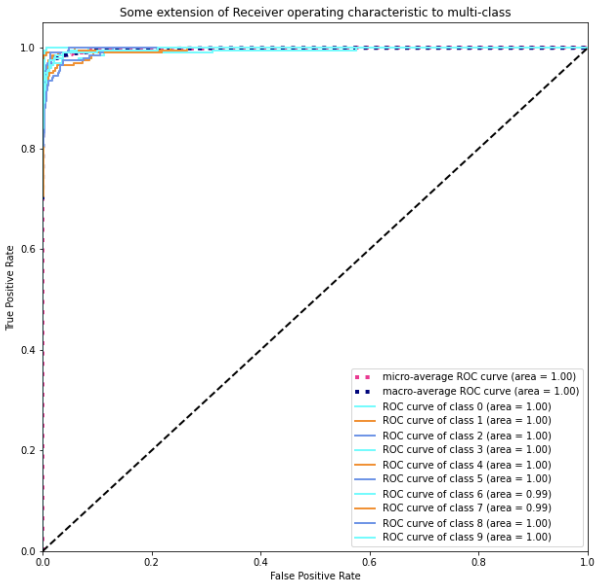
Accuracy vs Epoch

Train	Test	Combined
		

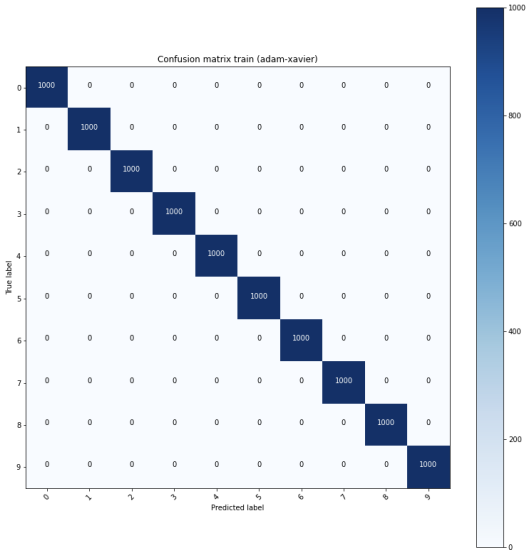
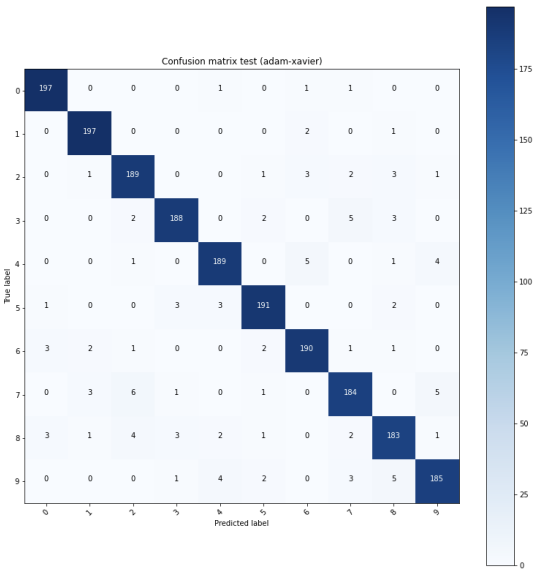
Loss vs Epoch

Train	Test	Combined
		

ROC curves (Val set)



Confusion Matrix of Validation set

Train	Test
	

Observation

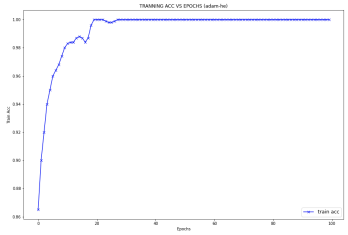
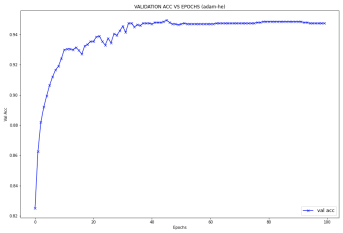
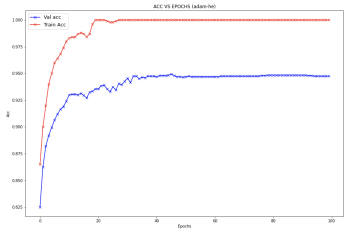
```
================================================================================================
Final Train Accuracy: 100.0
Final Test Accuracy: 94.65

================================================================================================
Optimizer: "adam"
------------------------------------------------------------------------------------------------
Epochs: 100
------------------------------------------------------------------------------------------------
Activation Fn(Hidden Layers): "tanh"
------------------------------------------------------------------------------------------------
Activation Fn(Output Layer): "softmax"
------------------------------------------------------------------------------------------------
Step size: 0.01
------------------------------------------------------------------------------------------------
Weight initialization strategy: "xavier"
------------------------------------------------------------------------------------------------
Regularization: "None" Lambda: "None"
------------------------------------------------------------------------------------------------
Dropout: None
------------------------------------------------------------------------------------------------
Batch size: 64

------------------------------------------------------------------------------------------------
Layer 1: (128, 784)

------------------------------------------------------------------------------------------------
Layer 2: (24, 128)

------------------------------------------------------------------------------------------------
Layer 3: (10, 24)

================================================================================================
```

2. He

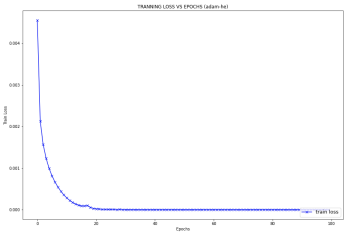
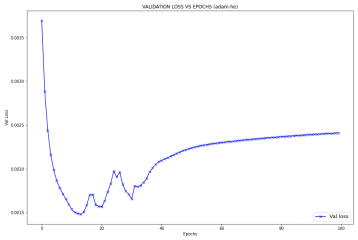
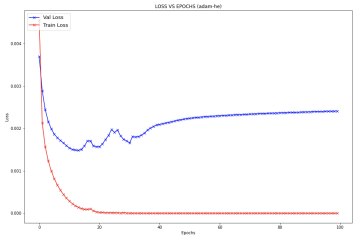
Plots

From the plots it can be observed that after approximately 19 epochs model starts overfitting and learning the training and and doing less generalization. At this time He gives the accuracy which is training 98% and testing 93% (approx). Also we know Xavier initialization works very well for tanh and He is preferred for Relu.

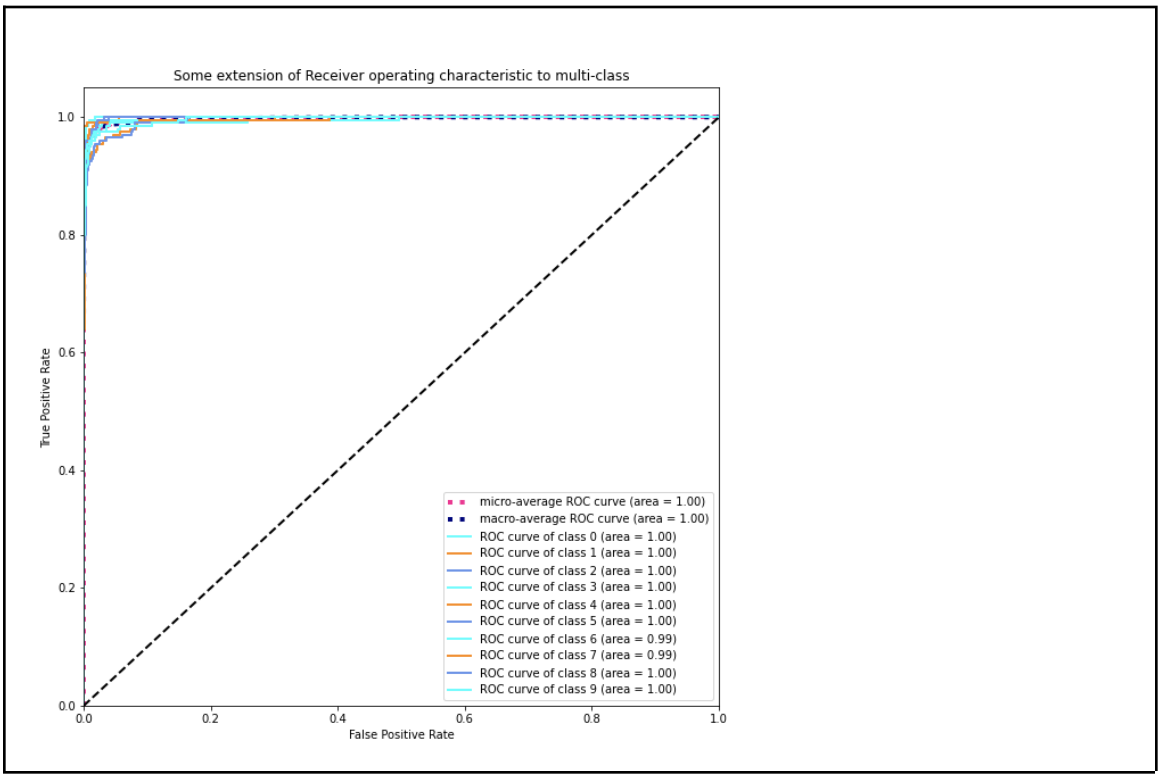
Accuracy vs Epoch

Train	Test	Combined
		

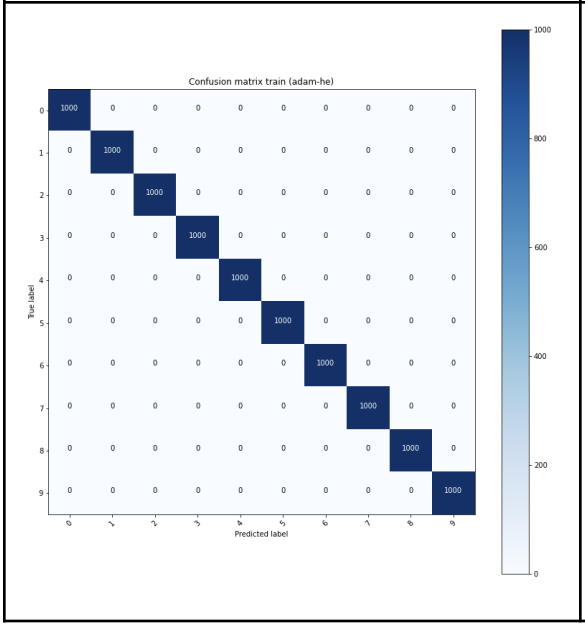
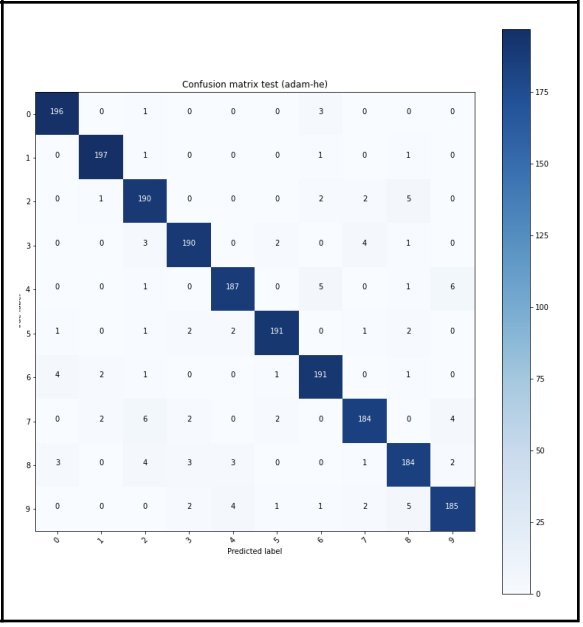
Loss vs Epoch

Train	Test	Combined
		

ROC curves (Val set)



Confusion Matrix of Validation set

Train	Test
	

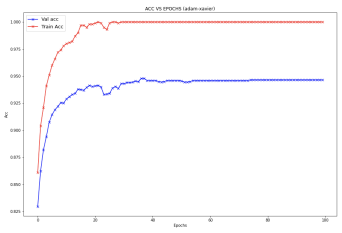
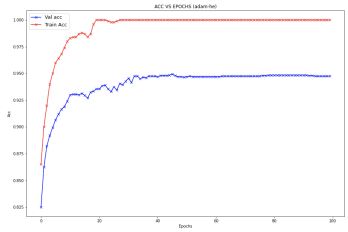
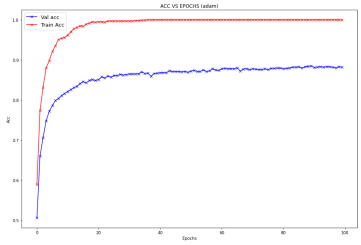
Observation

```
================================================================================================
Final Train Accuracy: 100.0
Final Test Accuracy: 94.75

================================================================================================
Optimizer: "adam"
------------------------------------------------------------------------------------------------
Epochs: 100
------------------------------------------------------------------------------------------------
Activation Fn(Hidden Layers): "tanh"
------------------------------------------------------------------------------------------------
Activation Fn(Output Layer): "softmax"
------------------------------------------------------------------------------------------------
Step size: 0.01
------------------------------------------------------------------------------------------------
Weight initialization strategy: "he"
------------------------------------------------------------------------------------------------
Regularization: "None" Lambda: "None"
------------------------------------------------------------------------------------------------
Dropout: None
------------------------------------------------------------------------------------------------
Batch size: 64

------------------------------------------------------------------------------------------------
Layer 1: (128, 784)

------------------------------------------------------------------------------------------------
Layer 2: (24, 128)

------------------------------------------------------------------------------------------------
Layer 3: (10, 24)

================================================================================================
```

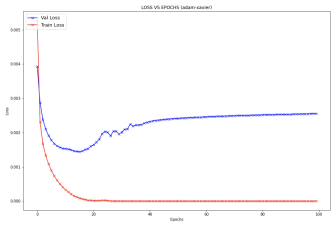
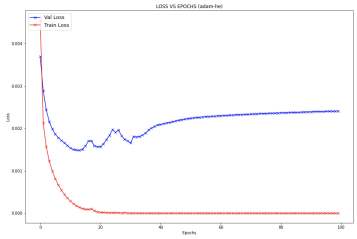
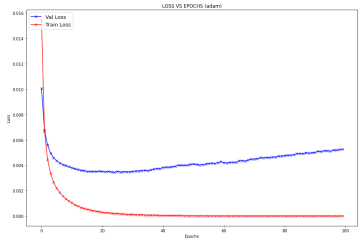
- ***Comparison of Random Initialization , Xavier , He***

```
Network = [784,128,24,10]
Learning Rate = 0.01
Epochs = 100
Activation = Tanh
Optimizer = Adam
Regularization = None
Dropout = None
Initialization = Variable
```

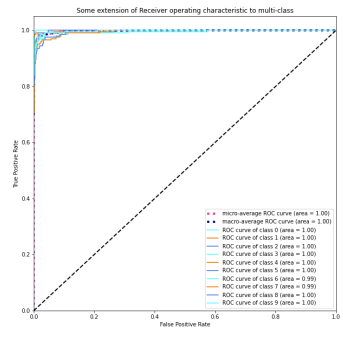
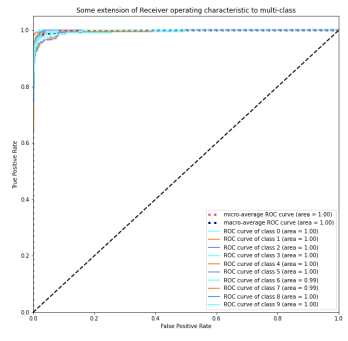
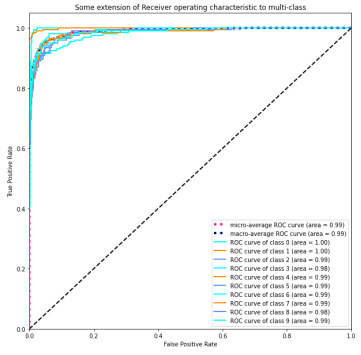
Accuracy vs Epoch

Xavier	He	Random
		

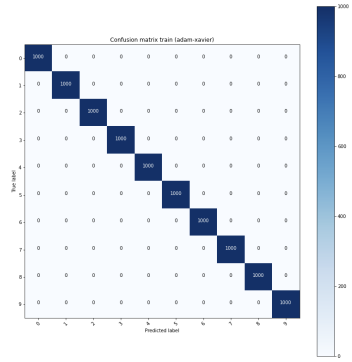
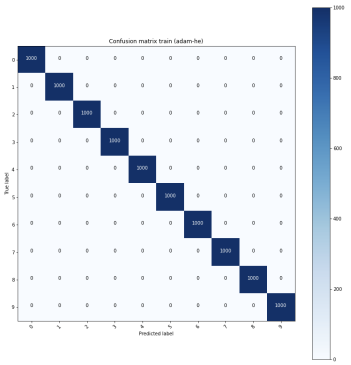
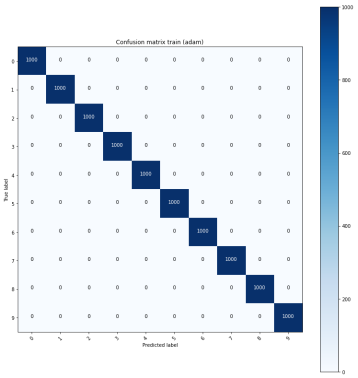
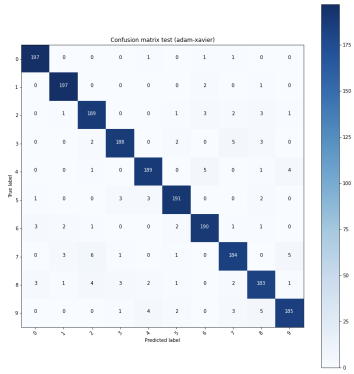
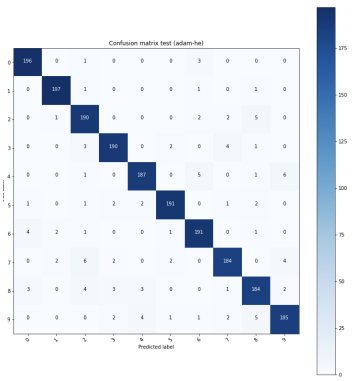
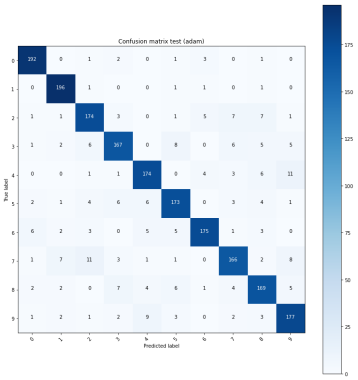
Loss vs Epoch

Xavier	He	Random
		

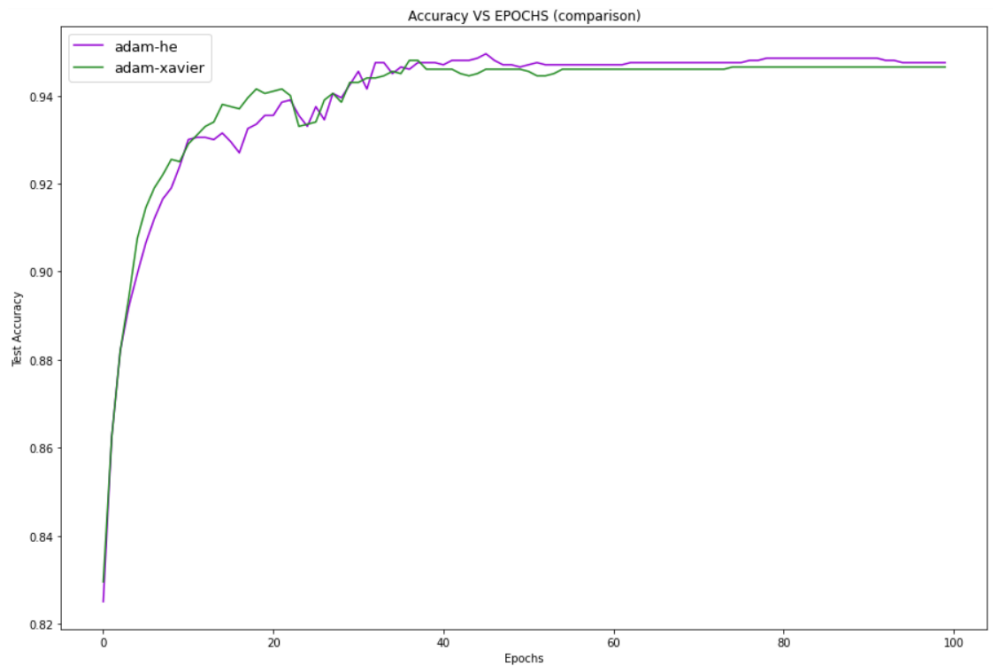
ROC curves

Xavier	He	Random
Test		
		

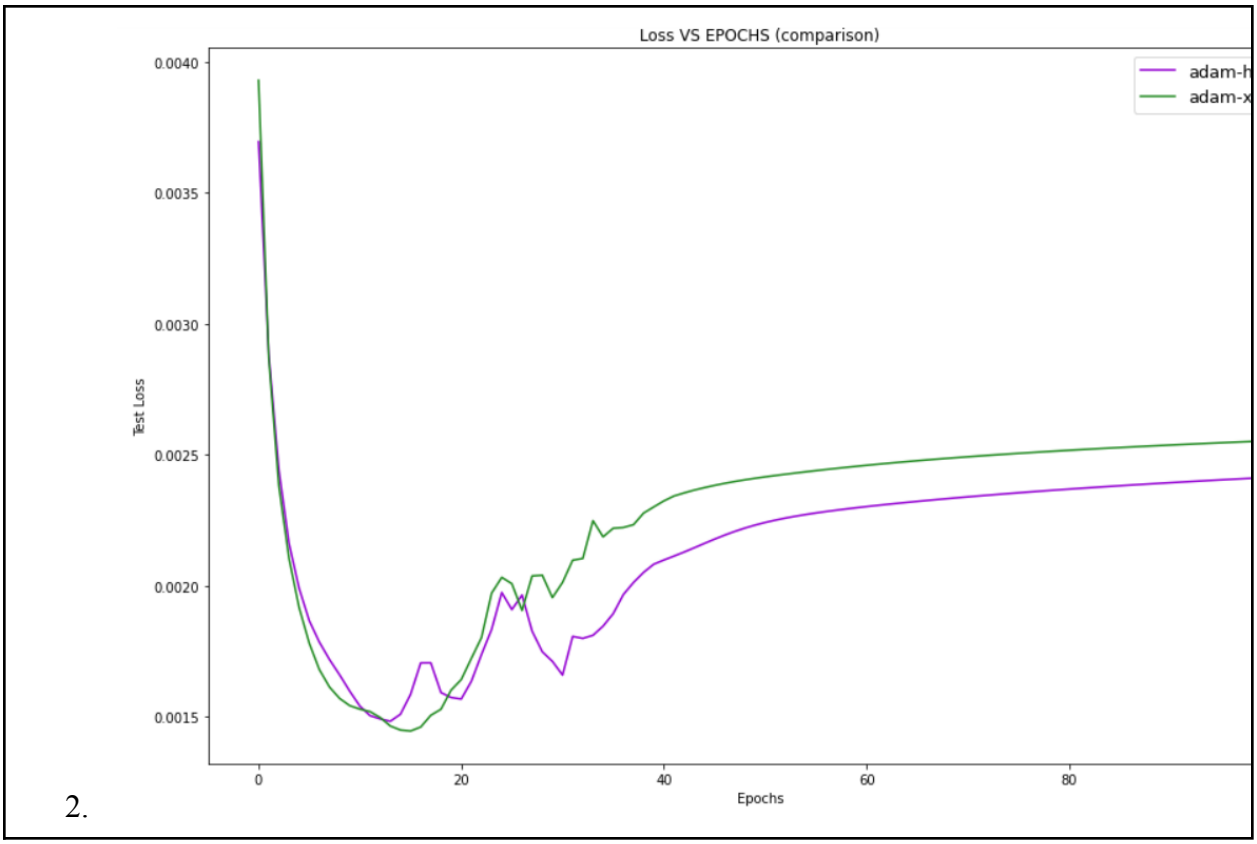
Confusion Matrix of Validation set

Xavier	He	Random
Train		
		
Test		
		

1.



2.



Observations :-

From the plots and analysis above the following observations are made.

- All the models using Adam whatever be the initialization are , are overfitting after approximately 20 epochs and loss is increasing.
- So for analysis we consider the first 20 epochs.
- Random Initialization initialises the weights randomly and thus may not get a good start for training, thus with random initialization the accuracy for first epoch is approximately 50 percent while for Xavier and He it is above 80%. With random weights we may not be close to minima also.
- For Xavier and He, both give good start to learning and achieve a good accuracy from the first epoch only that is above 80%. This may happen because they may begin reducing loss close to minima.
- Also we can observe that at 20th epoch, Xavier outperforms He and thus we have chosen Xavier initialization.
- Also it is proved theoretically that Xavier out performs He when the activation function is tanh.

1.5 Best Configuration form Part-2 (b)

To be used in later parts

```
Network = [784,128,24,10]
Learning Rate = 0.01
Epochs = 100
Activation = Tanh
Initialization = 'Xavier'
Regularization = None
Dropout = 'None'
```

Note :- The model though overfitting cannot be said as best configuration at 100 epochs but it may be helpful in doing a thorough analysis for the different types of regularization techniques in the question part 2

2. Regularization:

3.3 L1 - Methodology, Plots and Observations

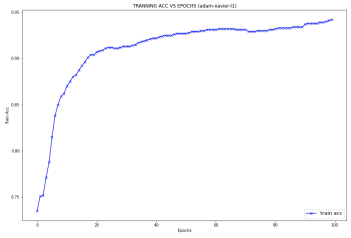
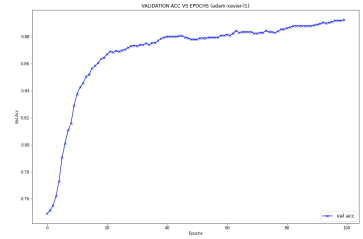
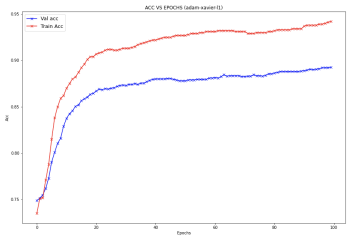
L1 Regularization :-

Plots

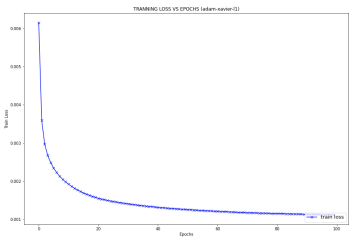
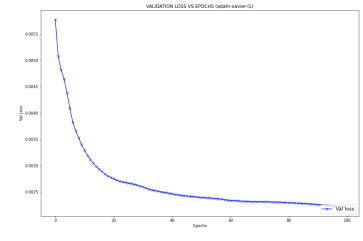
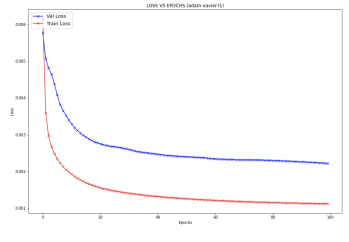
It can be clearly observed from the graph that the model is not overfitting as both training and testing loss are decreasing together and similarly the accuracies are increasing. So L1 regularization is definitely working here, where without it model was overfitting earlier at 20th epoch, Here we can clearly see that there is no overfitting happening. Both training and testing accuracies are close by in a range of 5% and this proves that model is able to generalise well.

A grid search was applied to find the optimal value of lambda among [0.1,0.01,0.001] and it was observed that this model required less regularization effect and so 0.001 works.

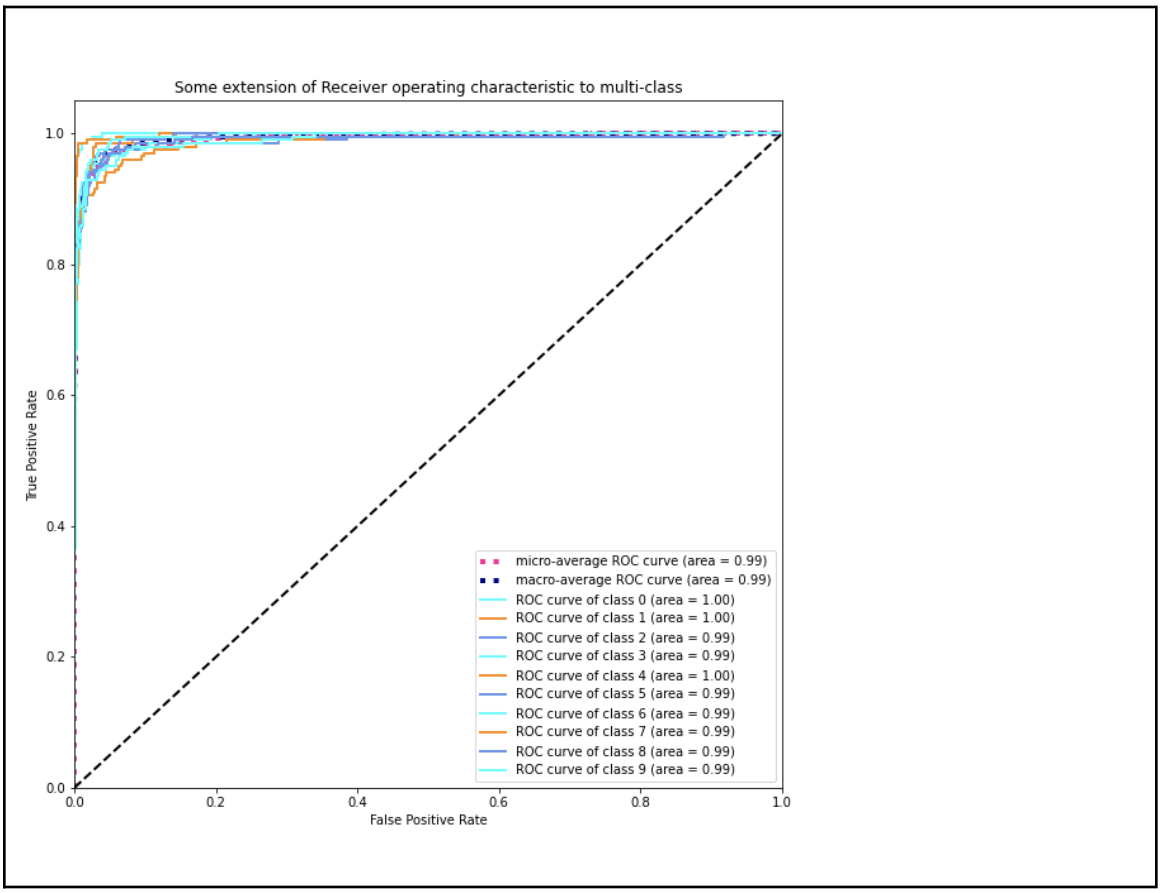
Accuracy vs Epoch

Train	Test	Combined
		

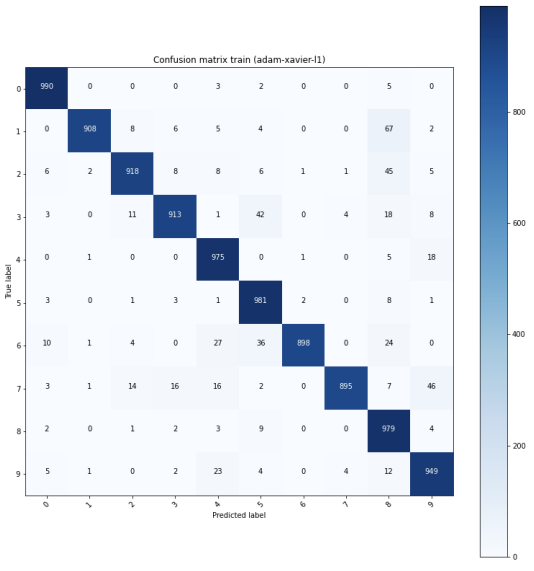
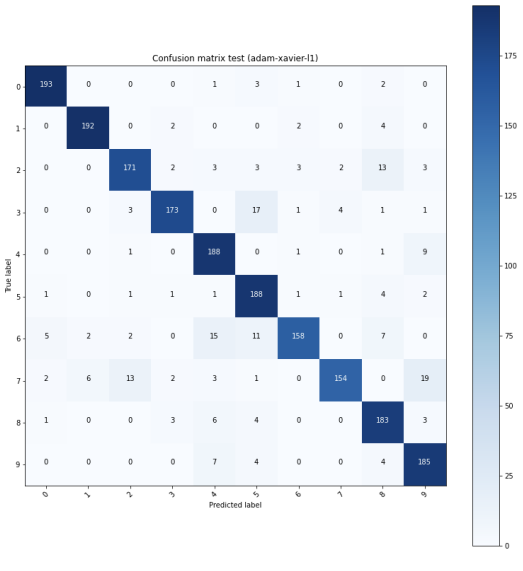
Loss vs Epoch

Train	Test	Combined
		

ROC curves (Val set)



Confusion Matrix of Validation set

Train	Test
	

Observation

```
================================================================================================
Final Train Accuracy: 94.06
Final Test Accuracy: 89.25

================================================================================================
Optimizer: "adam"
------------------------------------------------------------------------------------------------
Epochs: 100
------------------------------------------------------------------------------------------------
Activation Fn(Hidden Layers): "tanh"
------------------------------------------------------------------------------------------------
Activation Fn(Output Layer): "softmax"
------------------------------------------------------------------------------------------------
Step size: 0.01
------------------------------------------------------------------------------------------------
Weight initialization strategy: "xavier"
------------------------------------------------------------------------------------------------
Regularization: "l1" Lambda: "0.001"
------------------------------------------------------------------------------------------------
Dropout: None
------------------------------------------------------------------------------------------------
Batch size: 64

------------------------------------------------------------------------------------------------
Layer 1: (128, 784)

------------------------------------------------------------------------------------------------
Layer 2: (24, 128)

------------------------------------------------------------------------------------------------
Layer 3: (10, 24)

================================================================================================
```

3.2 L2 - Methodology, Plots and Observations

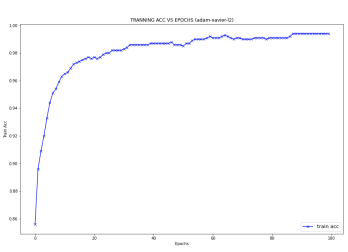
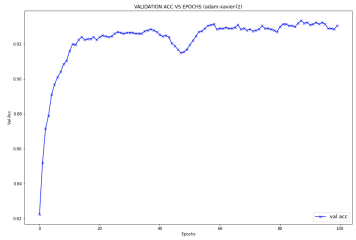
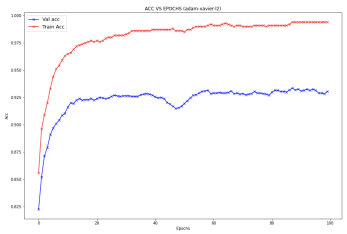
L2 Regularization :-

Plots

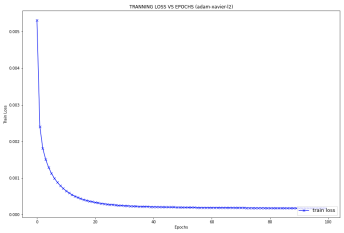
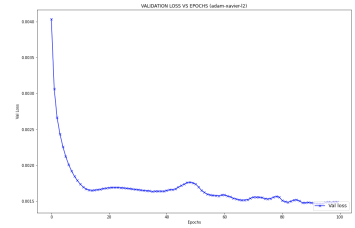
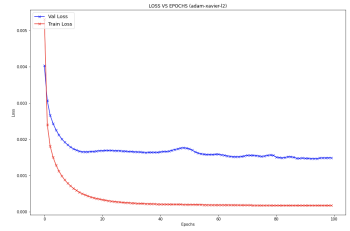
It can be clearly observed from the graph that the model is not overfitting as both training and testing loss are decreasing together and similarly the accuracies are increasing. So L2 regularization is definitely working here, where without it model was overfitting earlier at 20th epoch, Here we can clearly see that there is no overfitting happening. Both training and testing accuracies are close by in a range of 5% and this proves that model is able to generalise well.

A grid search was applied to find the optimal value of lambda among [0.1,0.01,0.001] and it was observed that this model required less regularization effect and so 0.001 works.

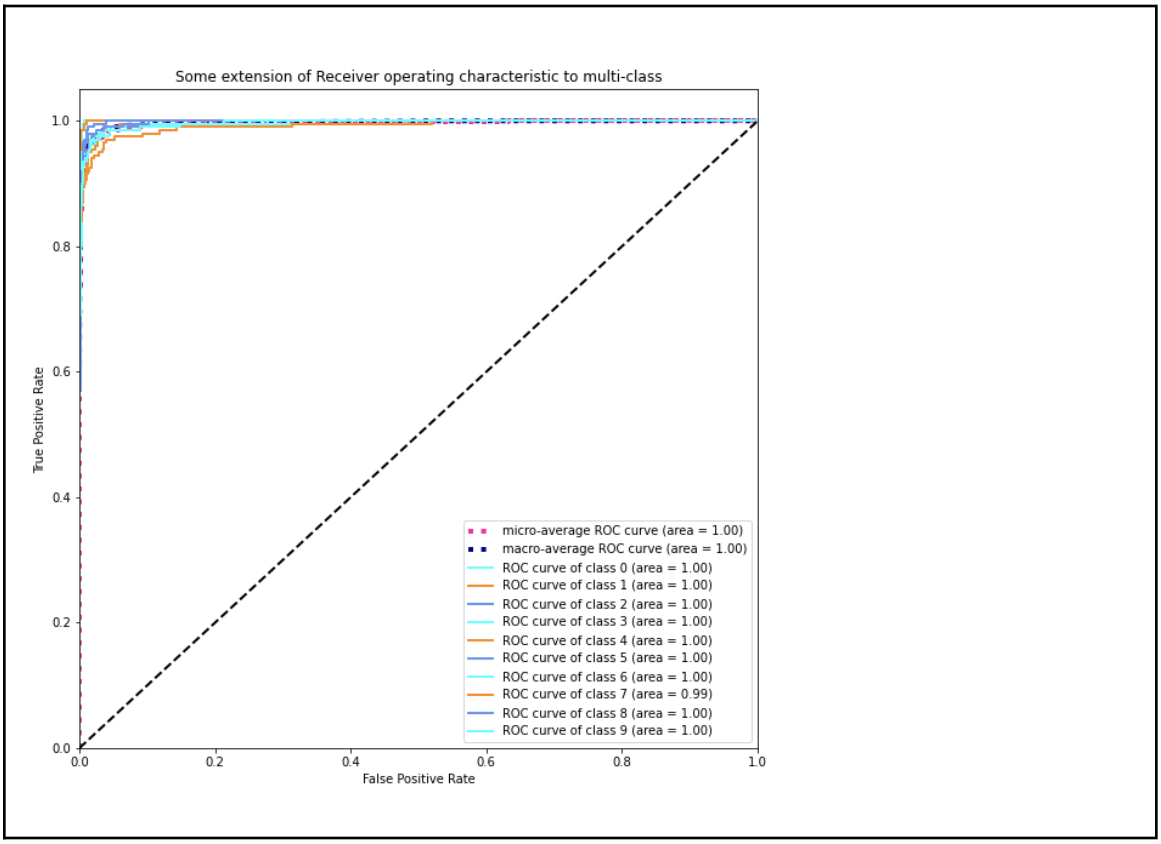
Accuracy vs Epoch

Train	Test	Combined
		

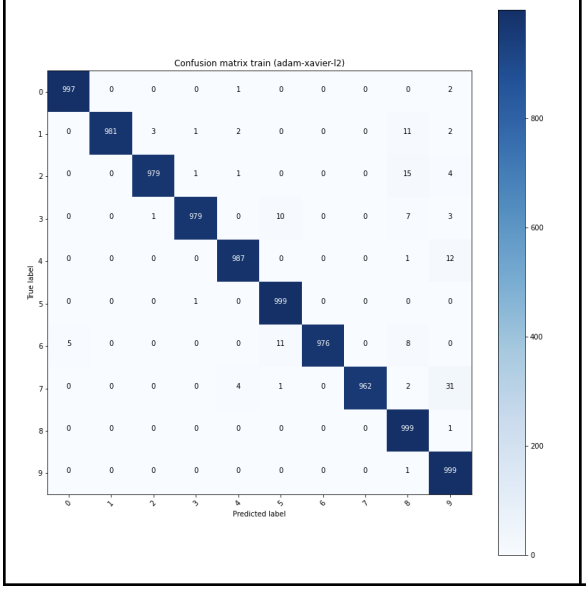
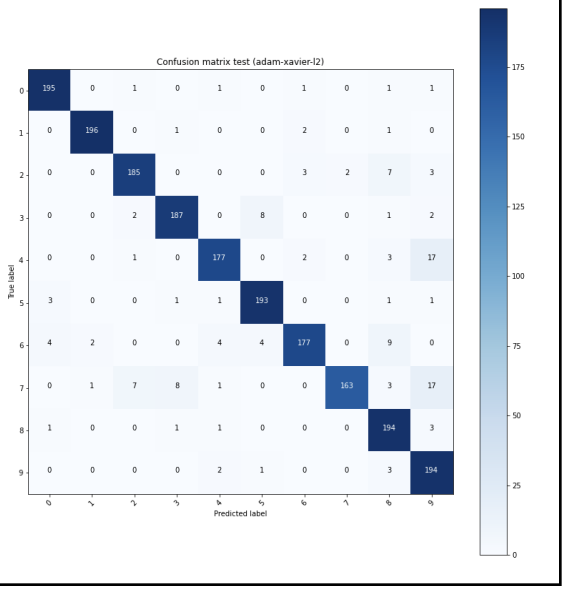
Loss vs Epoch

Train	Test	Combined
		

ROC curves (Val set)



Confusion Matrix of Validation set

Train	Test
	

Observations

```
================================================================================================
Final Train Accuracy: 98.58
Final Test Accuracy: 93.05

================================================================================================
Optimizer: "adam"
------------------------------------------------------------------------------------------------
Epochs: 100
------------------------------------------------------------------------------------------------
Activation Fn(Hidden Layers): "tanh"
------------------------------------------------------------------------------------------------
Activation Fn(Output Layer): "softmax"
------------------------------------------------------------------------------------------------
Step size: 0.01
------------------------------------------------------------------------------------------------
Weight initialization strategy: "xavier"
------------------------------------------------------------------------------------------------
Regularization: "l2" Lambda: "0.001"
------------------------------------------------------------------------------------------------
Dropout: None
------------------------------------------------------------------------------------------------
Batch size: 64

------------------------------------------------------------------------------------------------
Layer 1: (128, 784)

------------------------------------------------------------------------------------------------
Layer 2: (24, 128)

------------------------------------------------------------------------------------------------
Layer 3: (10, 24)

================================================================================================
```

3.1 Dropout- Methodology, Plots and Observations

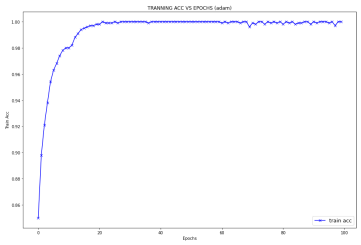
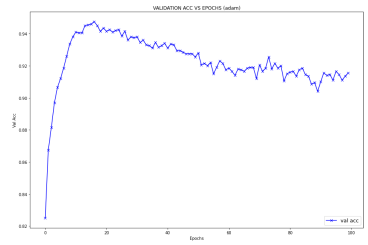
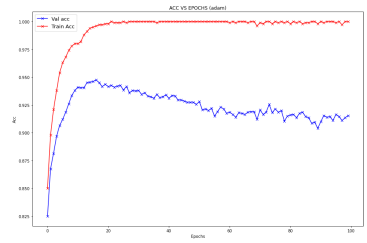
Plots

By changing the dropout probability of different hidden layers for the same configuration as Part 2 , Q1 we found the following observations. A thorough analysis of the accuracy by different dropout probabilities is shown in the later part.

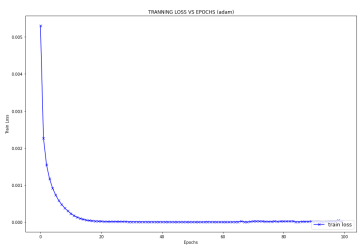
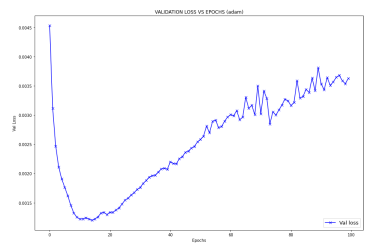
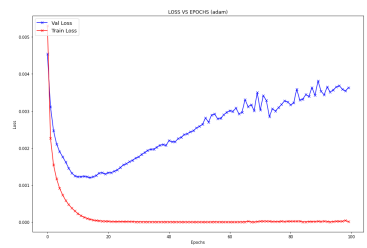
It was noticed that keeping the drop probability high model was underfitting so We preferred to keep drop probability low. (Explained Later part)

```
Network = [784,128,24,10]
Learning Rate = 0.01
Epochs = 100
Activation = Tanh
Initialization = Xavier
Dropout (layers) = [0,0.1,0.1,0]
Optimizer = Adam
```

Accuracy vs Epoch

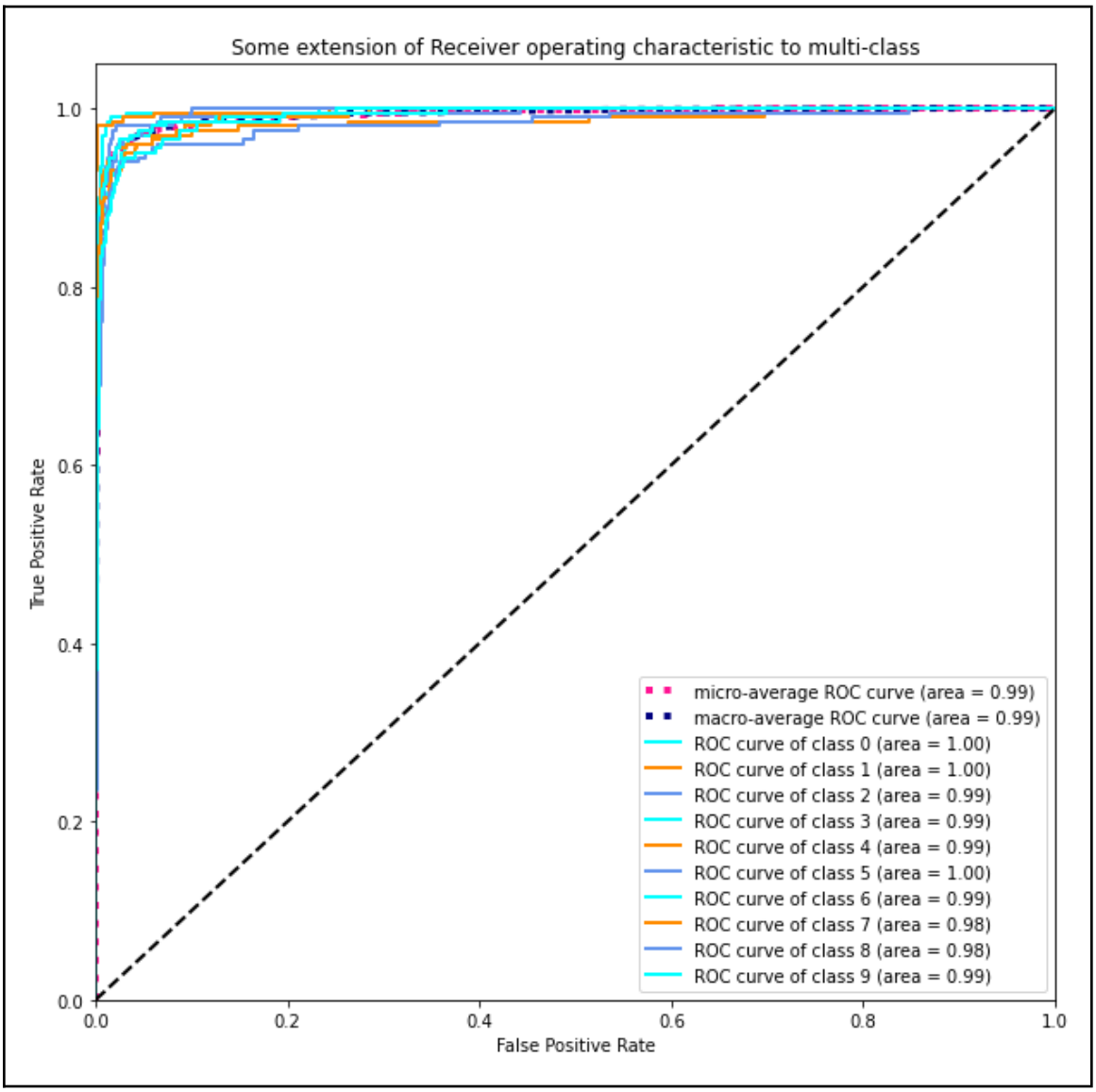
Train	Test	Combined
		

Loss vs Epoch

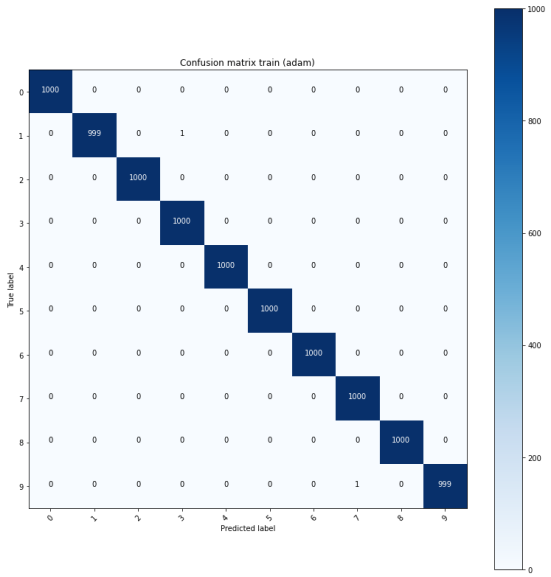
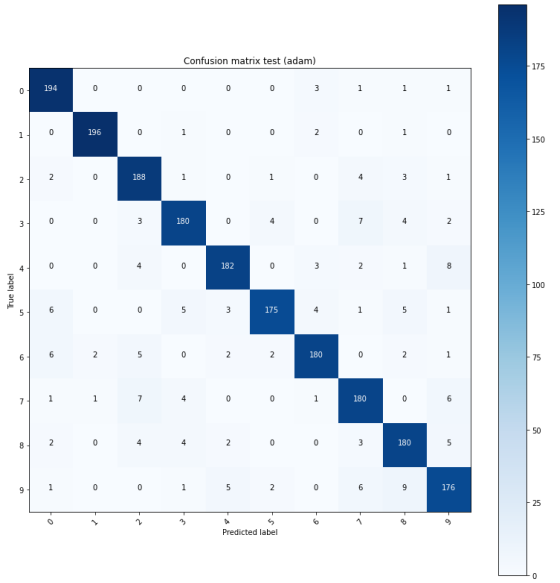
Train	Test	Combined
		

From these plots it can be seen that with more number of epochs along with dropout model starts getting simpler and accuracies start decreasing. So the right time to stop will be epoch-10 where maximum accuracy reached is 94%

ROC curves (Val set)



Confusion Matrix of Validation set

Train	Test
	

Observations

TESTING ACCURACY

91.55

TRAINING ACCURACY

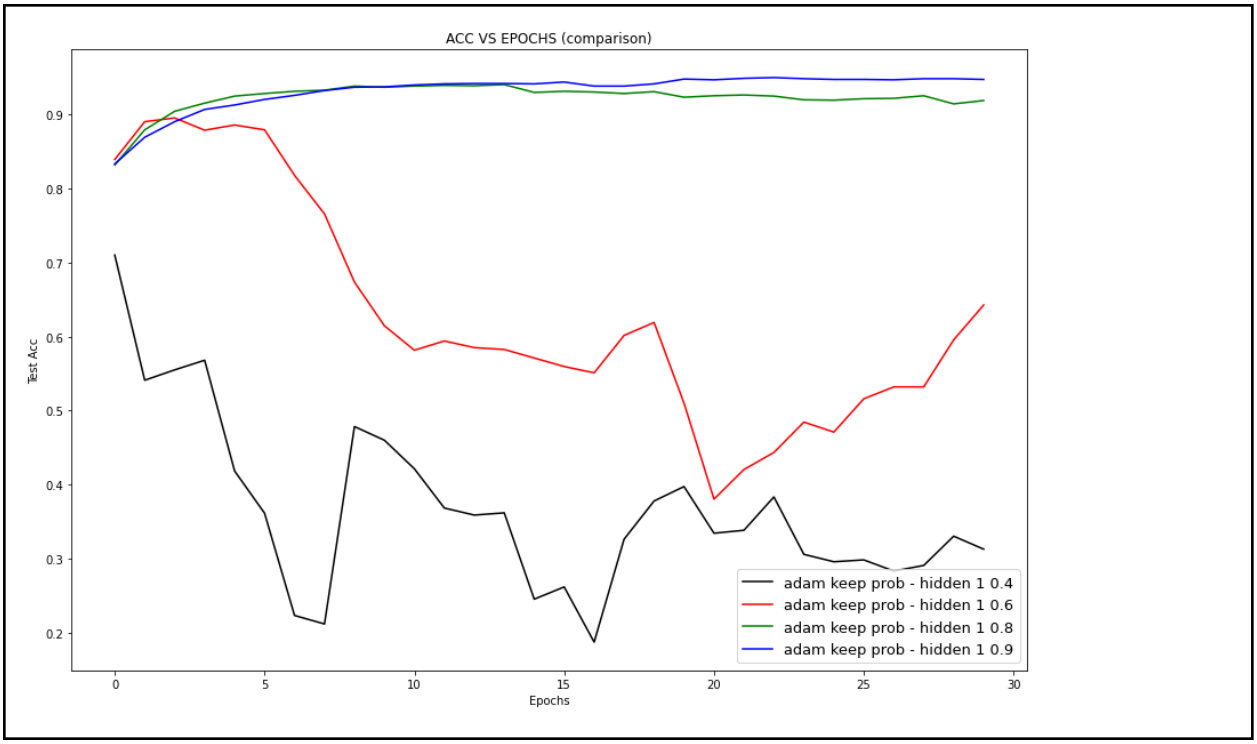
99.98

```
================================================================================================
Optimizer: "adam"
------------------------------------------------------------------------------------------------
Epochs: 100
------------------------------------------------------------------------------------------------
Activation Fn(Hidden Layers): "tanh"
------------------------------------------------------------------------------------------------
Activation Fn(Output Layer): "softmax"
------------------------------------------------------------------------------------------------
Step size: 0.01
------------------------------------------------------------------------------------------------
Weight initialization strategy: "xavier"
------------------------------------------------------------------------------------------------
Regularization: "None"
------------------------------------------------------------------------------------------------
Dropout: [0, 0.1, 0.1, 0]
------------------------------------------------------------------------------------------------
Batch size: 64

------------------------------------------------------------------------------------------------
Layer 1: (128, 784)

------------------------------------------------------------------------------------------------
Layer 2: (24, 128)

------------------------------------------------------------------------------------------------
Layer 3: (10, 24)

================================================================================================
```

Analysis of Effect of Dropout over changing Dropout Probability

```
Network = [784,128,24,10]
Learning Rate = 0.001
Epochs = 30
Activation = Tanh
Initialization = Xavier
Dropout (layers) = [0,variable,0,0]
Optimizer = Adam
```

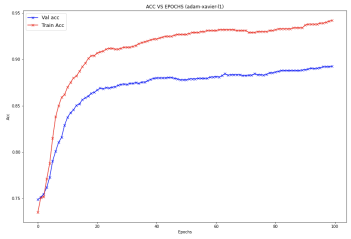
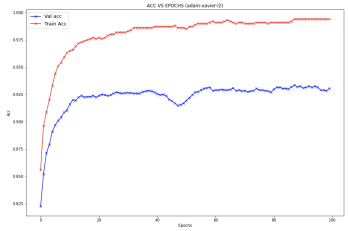
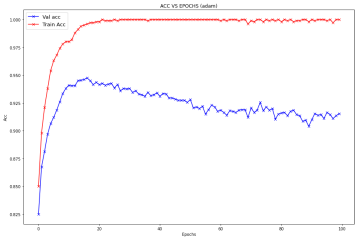
Here for analysis we kept all hyper parameters same and change only the dropout probabilities of first hidden layer containing 128 nodes respectively in this fashion [0.6,0.4,0.2,0.1]



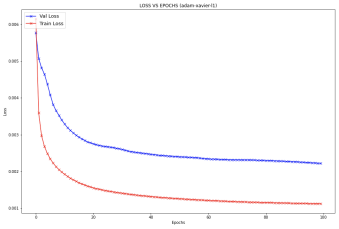
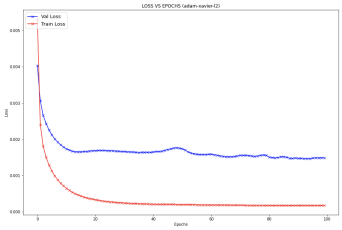
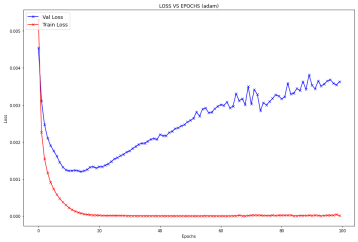
- It can be easily observed from the graph that dropout is actually making the model simpler i.e the more nodes dropped the simpler the model and it may also lead to underfitting if a lot of nodes are dropped.
- From the above graph it can be visualised when only 0.2 percent of nodes are dropped model is able to perform good but as soon as this value goes up to 0.4 the model actually starts underfitting and is not able to give good accuracy.
- From this analysis we chose to keep the dropout probability low so that model does not underfit. Also as hidden layer 2 has only 24 nodes we decided not to drop any nodes from that layer or drop a few only if from that layer.

Analysis of Various Regularization Plots and Observation

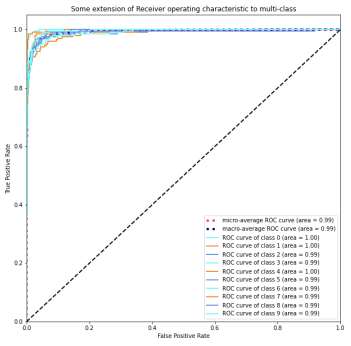
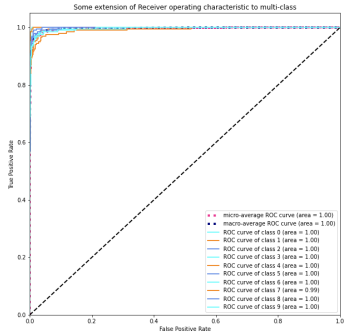
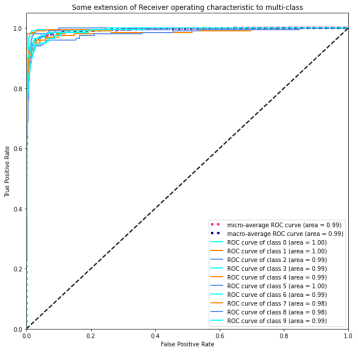
Accuracy vs Epoch

L1	L2	Dropout
		

Loss vs Epoch

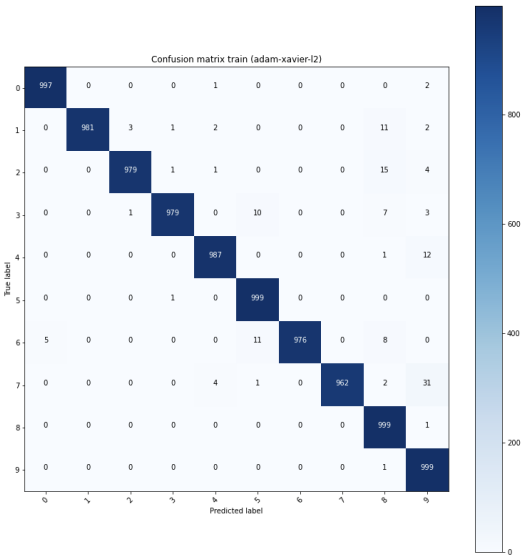
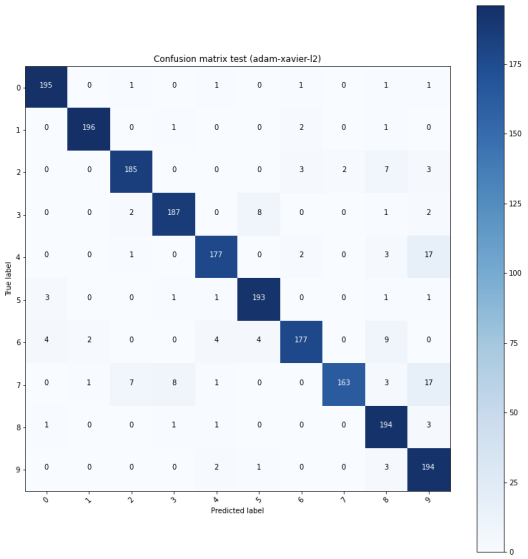
L1	L2	Dropout
		

ROC curves (Val set)

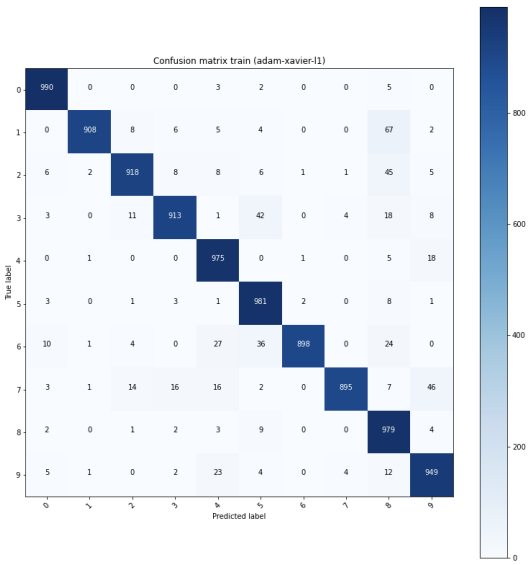
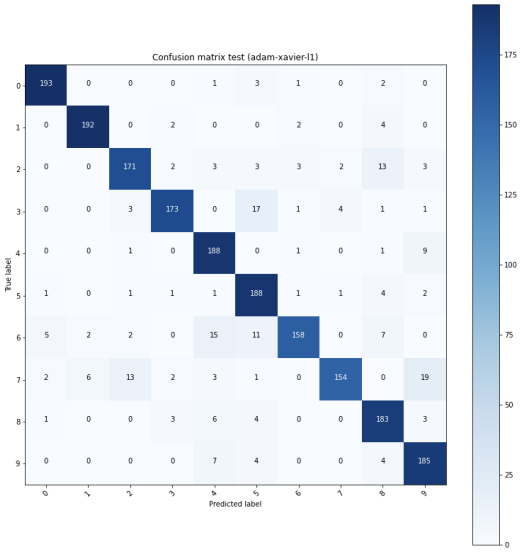
L1	L2	Dropout
		

Confusion Matrix of Validation set

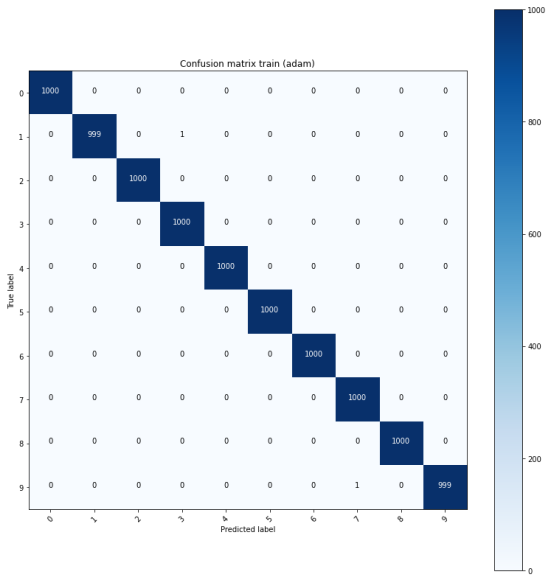
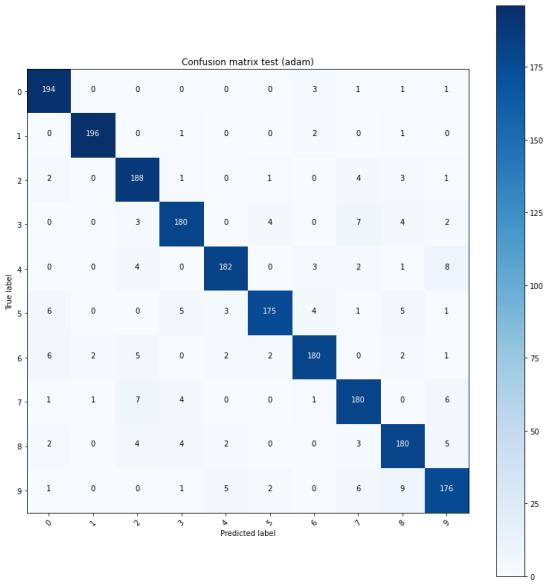
L1

Train	Test
	

L2

Train	Test
	

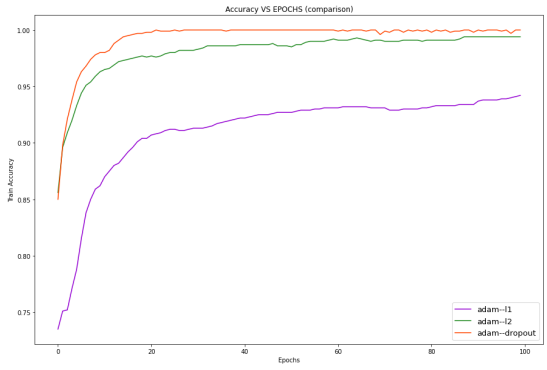
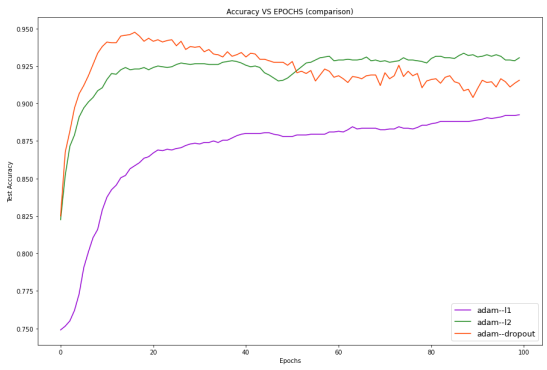
Dropout

Train	Test
	

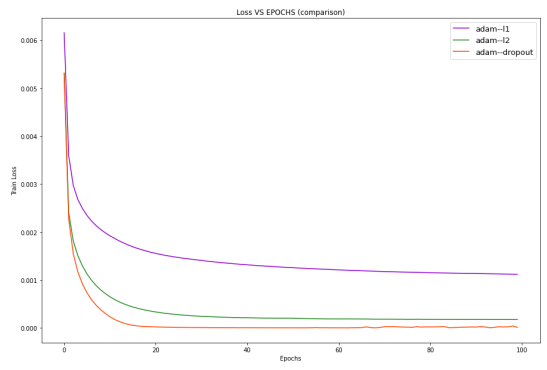
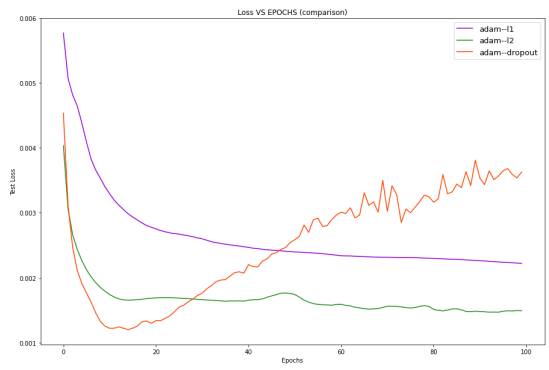
Observations

```
==========================================================================================
Final Train Accuracy: 99.98
Final Test Accuracy: 91.55

==========================================================================================
Optimizer: "adam"
------------------------------------------------------------------------------------------
Epochs: 100
------------------------------------------------------------------------------------------
Activation Fn(Hidden Layers): "tanh"
------------------------------------------------------------------------------------------
Activation Fn(Output Layer): "softmax"
------------------------------------------------------------------------------------------
Step size: 0.01
------------------------------------------------------------------------------------------
Weight initialization strategy: "xavier"
------------------------------------------------------------------------------------------
Regularization: "None" Lambda: "None"
------------------------------------------------------------------------------------------
Dropout: [0, 0.1, 0.1, 0]
------------------------------------------------------------------------------------------
Batch size: 64

------------------------------------------------------------------------------------------
Layer 1: (128, 784)

------------------------------------------------------------------------------------------
Layer 2: (24, 128)

------------------------------------------------------------------------------------------
Layer 3: (10, 24)

==========================================================================================
```

Accuracy vs Epoch (L1, L2, Dropout)

Train	Test
	

Loss

Train	Test
	

Observation :-

- From the above plots it is observed that regularization is working to reduce the overfitting problem and L2,Dropout perform equally good while L1 regularization lags behind.
- The reason why L1 is not able to perform as good as other techniques is that L1 tries to reduce the loss and in this process makes weights zero i.e weight matrix become sparse. This affects a lot as many activations become zero and moving across layers some neurons contribute less due to sparse weights.
- In comparison to L1, L2 and Dropout perform better.
- In Dropout it can be observed that as we increase the number of epochs more loss starts increasing for the validation set, this is because we are regularizing too much. But if we keep epocs to 20, dropout outperforms both L1 and L2 in the case of regularization.

2.2 Best Model

BEST OPTIMIZER AND MODEL :-

```
Network = [784,128,24,10]
Learning Rate = 0.01
Epochs = 30
Activation = Tanh
Initialization = Xavier
Dropout (layers) = [0,0.1,0.1,0]
Optimizer = Adam
```

From the above analysis we can observe that dropout has performed the best with an accuracy around 95% and we can obtain this model using early stopping near epoch 20. Because the model architecture is quite simple we are able to obtain a generalized model with a low dropout factor of just 0.1 in the hidden layers.

Contribution of Each Member

1. **Shivam Sharma**
   Implemented Regularization and plots for each along with regularization part of readme
2. **Akanksha Shrimal**
   Implemented Dropout and report
3. **Vaibhav Goswami**
   Implemented Initialization and report.