Please submit a PDF of your notebook with all the outputs, and separately the source code on Gradescope. You grade will be primarily based on the outputs in the PDF submission. Please refer to the introductory part of any previous problem set in this class regarding general homework standards and procedures.

```
#importing libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, losses, optimizers
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.utils import plot_model, to_categorical
from tensorflow.keras.callbacks import ModelCheckpoint
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


from google.colab import drive
drive.mount('/content/drive')
```

```
⊢→    Mounted at /content/drive
```

```
import sys
sys.path.append('/content/drive/My Drive/Colab Notebooks/mad_class/hw5/')
```

```
!ls
```

```
    drive  sample_data
```

```
def plot_history(history, filename, model_name):
    best_epoch = history.history['val_loss'].index(min(history.history['val_loss']))
    fig, ax1 = plt.subplots(figsize=(12,8))
    plt.title(' '.join([model_name, 'model learning curve - Max accuracy on test is %1.4
    ax1.set_xlabel('Epochs')
    ax1.set_xticks(range(Epochs))
    ax1.set_ylabel('Loss')
    ax1.plot(range(Epochs), history.history['loss'], 'r', label='Train Loss')
    ax1.plot(range(Epochs), history.history['val_loss'], 'orange', label='Test Loss')
    ax1.axvline(best_epoch, color='m', lw=4, alpha=0.5, label='Best epoch')
    ax1.legend()

    ax2 = ax1.twinx()
    ax2.set_ylabel('Accuracy')
    ax2.plot(range(Epochs), history.history['accuracy'], 'g', label='Train Accuracy')
    ax2.plot(range(Epochs), history.history['val_accuracy'], 'b', label='Test Accuracy')
    ax2.legend()

    plt.savefig(filename)
```

```python
        plt.show()

def plot_first25labels(test_x, test_y, test_y_hat):
    #plotting first 25 samples with labels
    plt.figure(figsize=(12,12))
    plt.suptitle('First 25 samples of MNIST test dataset and their estimated labels\nActu
    for i in range(25):
        plt.subplot(5,5,i+1)
        plt.title('%d -> %d' % (test_y[i], test_y_hat[i]))
        plt.imshow(test_x[i, :, :, 0], cmap='gray')
        plt.axis('off')

def plot_mislabeled(test_x, test_y, test_y_hat):
    #plotting first 25 samples with mislabeled
    rows = np.where(test_y_hat != test_y)[0]

    if len(rows) < 25:
        raise Exception('Mislabeled samples are less than 25 (%d). Perfect model!' % len

    plt.figure(figsize=(12,12))
    plt.suptitle('First 25 samples of MNIST test dataset that the model mislabeled\nActua
    for i in range(25):
        index = rows[i]
        plt.subplot(5,5,i+1)
        plt.title('%d -> %d' % (test_y[index], test_y_hat[index]))
        plt.imshow(test_x[index, :, :, 0], cmap='gray')
        plt.axis('off')

def plot_confusion(test_y, test_y_hat):
    # Show the confusion matrix
    cm = confusion_matrix(test_y, test_y_hat, normalize='true')

    df_cm = pd.DataFrame(cm, index = [i for i in range(10)],
                                  columns = [i for i in range(10)])
    plt.figure(figsize = (10,7))
    plt.title(' '.join(['Confusion matrix of', model_name]))
    sn.heatmap(df_cm, annot=True, fmt='.2%')
    plt.xlabel('Predicted label')
    plt.ylabel('True label')
    plt.show()


#loading data
mnist = tf.keras.datasets.mnist

(train_x, train_y), (test_x, test_y) = mnist.load_data()

print('Shape of train_x is : %s (min= %1.2f, max= %1.2f)' % (str(train_x.shape), train_
print('Shape of train_y is : %s (min= %d, max= %d)' % (str(train_y.shape), train_y.min(
```

```python
#preprocessing input
train_x = train_x.astype('float') / 255.0
test_x = test_x.astype('float') / 255.0

train_x = np.expand_dims(train_x, -1)
test_x = np.expand_dims(test_x, -1)

print('Shape of train_x is : %s (min= %1.2f, max= %1.2f)' % (str(train_x.shape), train_
print('Shape of train_y is : %s (min= %d, max= %d)' % (str(train_y.shape), train_y.min(
```

```
Shape of train_x is : (60000, 28, 28, 1) (min= 0.00, max= 1.00)
Shape of train_y is : (60000,) (min= 0, max= 9)
```

```python
#plotting first 25 samples
plt.figure(figsize=(12,12))
plt.suptitle('First 25 samples of MNIST train dataset')
for i in range(25):
        plt.subplot(5,5,i+1)
        plt.title('Label is %d' % train_y[i])
        plt.imshow(train_x[i, :, :, 0], cmap='gray')
        plt.axis('off')
```

| Label is 5 | Label is 0 | Label is 4 | Label is 1 | Label is 9 |
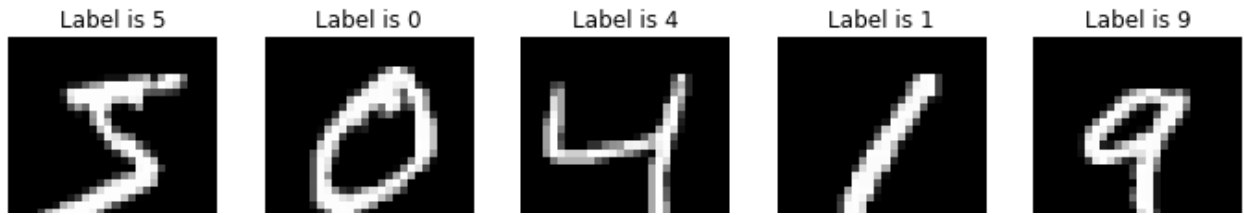
```
#counting number of samples for each class and plotting them
(_, train_count) = np.unique(train_y, return_counts=True)
(_, test_count) = np.unique(test_y, return_counts=True)

plt.figure(figsize=(12,8))
plt.title('MNIST data distribution')
plt.xticks(range(10), labels=range(10))
plt.xlabel('Number')
plt.ylabel('Number of samples')
plt.bar(range(10), train_count, label='Train')
plt.bar(range(10), test_count, label='Test', bottom=train_count)
plt.legend()
plt.savefig('DataDistribution.png')
plt.show()
```

MNIST data distribution
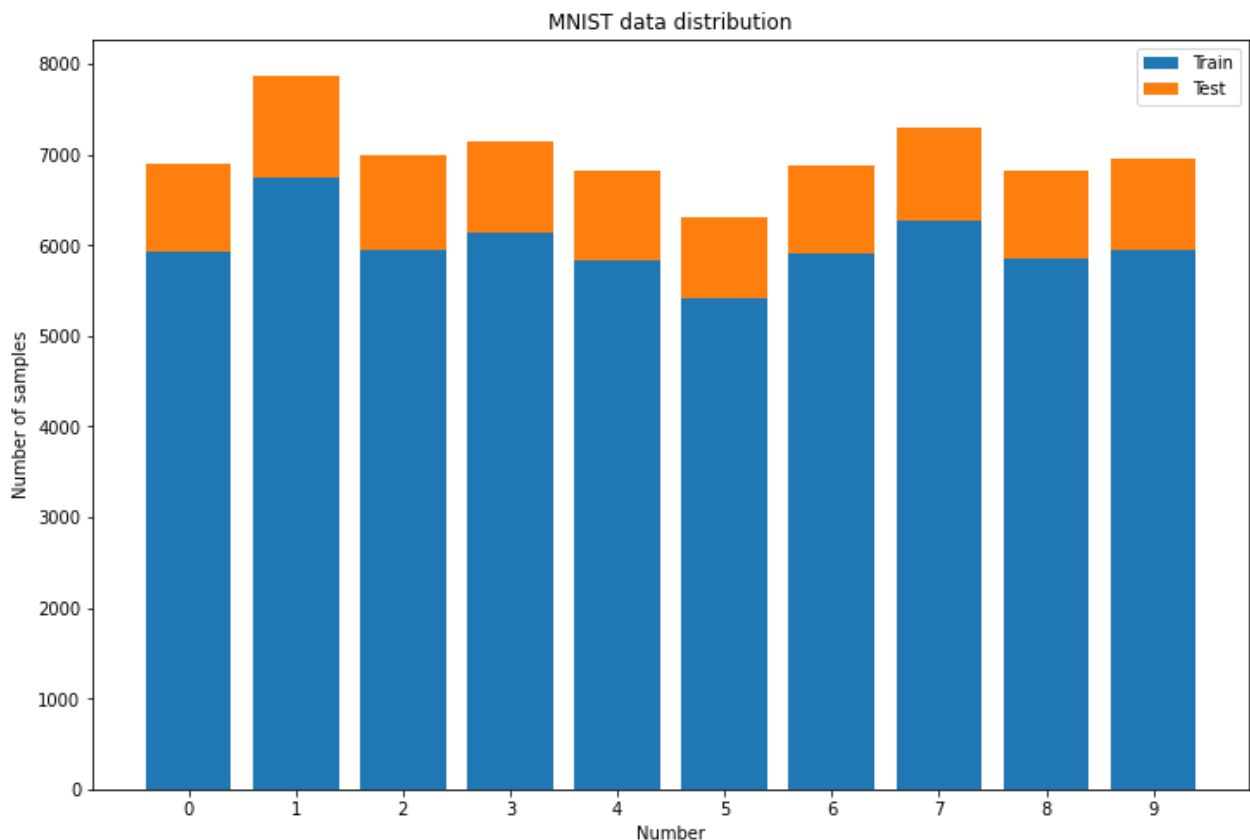
```
Epochs = 10 #hyperparameter
BatchSize = 32 #hyperparameter
```

```
#one hot encoding to match loss function expectation
one_hot_train_y = to_categorical(train_y, num_classes=10)
one_hot_test_y = to_categorical(test_y, num_classes=10)
print('Shape of one_hot_train_y is : ' + str(one_hot_train_y.shape))
print('Shape of one_hot_test_y is : ' + str(one_hot_test_y.shape))

      Shape of one_hot_train_y is : (60000, 10)
      Shape of one_hot_test_y is : (10000, 10)
```

(a) Using the model framework below in Question (b), define and train multinomial logistic regression, i.e., you will only have an input layer, a flattening layer, and an outer layer with softmax activation). You should get accuracy comparable to that of the model in Question (b) (~90%)

```
inp = layers.Input(shape=(28, 28, 1), name='InputLayer')
x = layers.Flatten(name='FlattenLayer')(inp)
outp = layers.Dense(10, activation='softmax', name='OutputLayer')(x)

model = Model(inp, outp, name='Model')
model.compile(loss=losses.CategoricalCrossentropy(), optimizer=optimizers.SGD(), metrics=['accura

model.summary()
plot_model(model, show_shapes=True, to_file='Model.png')
```

```
Model: "Model"
```

_____

```
 Layer (type)                 Output Shape              Param #
```
===

```
!mkdir  LinearModelCheckPoints
```

~~FlattenLayer (Flatten)   (None, 784)   0~~

```
#training  model
history  =  model.fit(train_x,  one_hot_train_y,
                               validation_data=(test_x,  one_hot_test_y),
                               epochs=Epochs,
                               batch_size=BatchSize,
                               callbacks=[
                                       ModelCheckpoint(filepath='LinearModelCheckPoints/be
                                       ModelCheckpoint(filepath='LinearModelCheckPoints/de
```

```
Epoch 1/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.7814 - accuracy: 0.8120 -
Epoch 2/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.4574 - accuracy: 0.8807 -
Epoch 3/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.4041 - accuracy: 0.8911 -
Epoch 4/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3773 - accuracy: 0.8965 -
Epoch 5/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.3605 - accuracy: 0.9003 -
Epoch 6/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.3486 - accuracy: 0.9031 -
Epoch 7/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3394 - accuracy: 0.9053 -
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3323 - accuracy: 0.9075 -
Epoch 9/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.3263 - accuracy: 0.9087 -
Epoch 10/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.3214 - accuracy: 0.9104 -
```

```
filename  =  'LinearModelLearningCurve.png'
model_name  =  'Multinomial  Logistic  Regression'
plot_history(history,  filename,  model_name)
```

Multinomial Logistic Regression model learning curve - Max accuracy on test is 0.9153
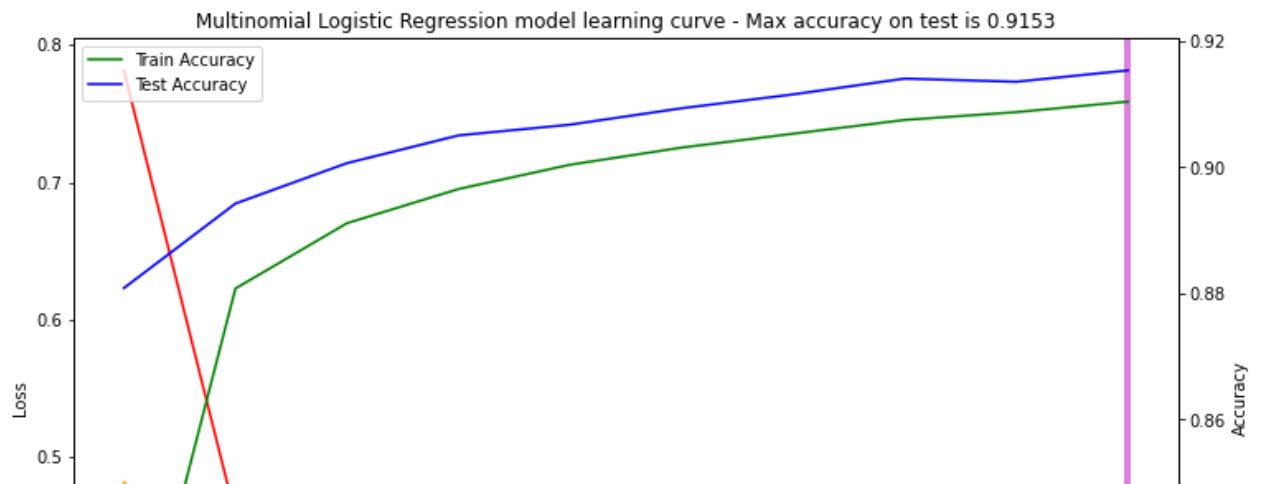
```
#loading best model (least validation loss)
model = load_model('LinearModelCheckPoints/best_dense_model.h5')
#evaluating model on test set
test_y_hat = model.predict(test_x)
print('test_y_hat shape is : ' + str(test_y_hat.shape))
#finding max and min of predictions
test_y_hat = np.argmax(test_y_hat, axis=1)
print('Now test_y_hat shape is : %s (min = %d, max = %d)' % (str(test_y_hat.shape), t
plot_first25labels(test_x, test_y, test_y_hat)
```

```
313/313 [==============================] - 0s 1ms/step
test_y_hat shape is : (10000, 10)
Now test_y_hat shape is : (10000,) (min = 0, max = 9)
```

First 25 samples of MNIST test dataset and their estimated labels
Actual lable -> Estimated label

| 7 -> 7 | 2 -> 2 | 1 -> 1 | 0 -> 0 | 4 -> 4 |

```
plot_mislabeled(test_x,   test_y,   test_y_hat)
```
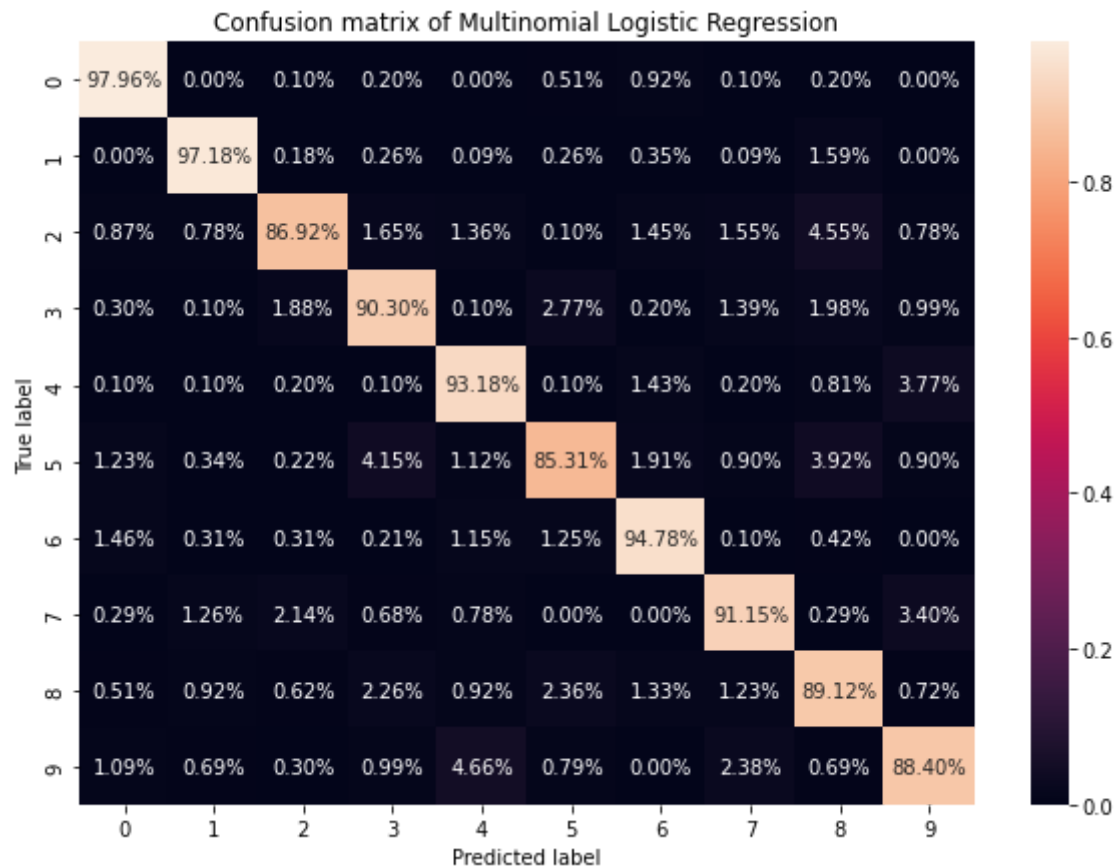
First 25 samples of MNIST test dataset that the model mislabeled
Actual lable -> Estimated label

5 -> 6	4 -> 6	3 -> 2	2 -> 7	9 -> 4
7 -> 4	2 -> 9	9 -> 4	3 -> 8	6 -> 5
8 -> 7	9 -> 8	3 -> 5	4 -> 6	6 -> 0
8 -> 4	4 -> 6	7 -> 9	3 -> 5	2 -> 3
9 -> 7	2 -> 7	5 -> 3	6 -> 4	5 -> 0

```
plot_confusion(test_y,  test_y_hat)
```


Confusion matrix of Multinomial Logistic Regression

```
#printing  classification  report
print(classification_report(test_y,  test_y_hat,  labels=[i  for  i  in  range(10)]))
```
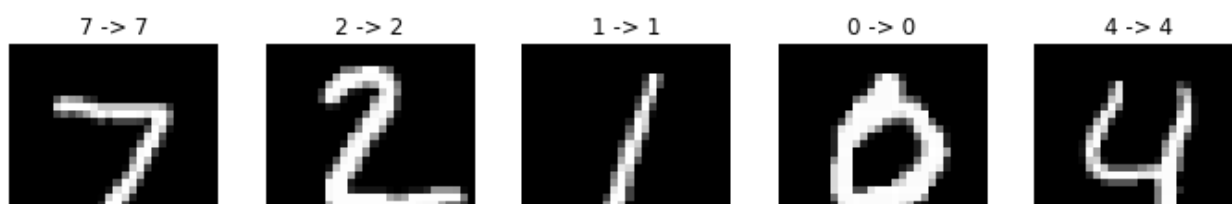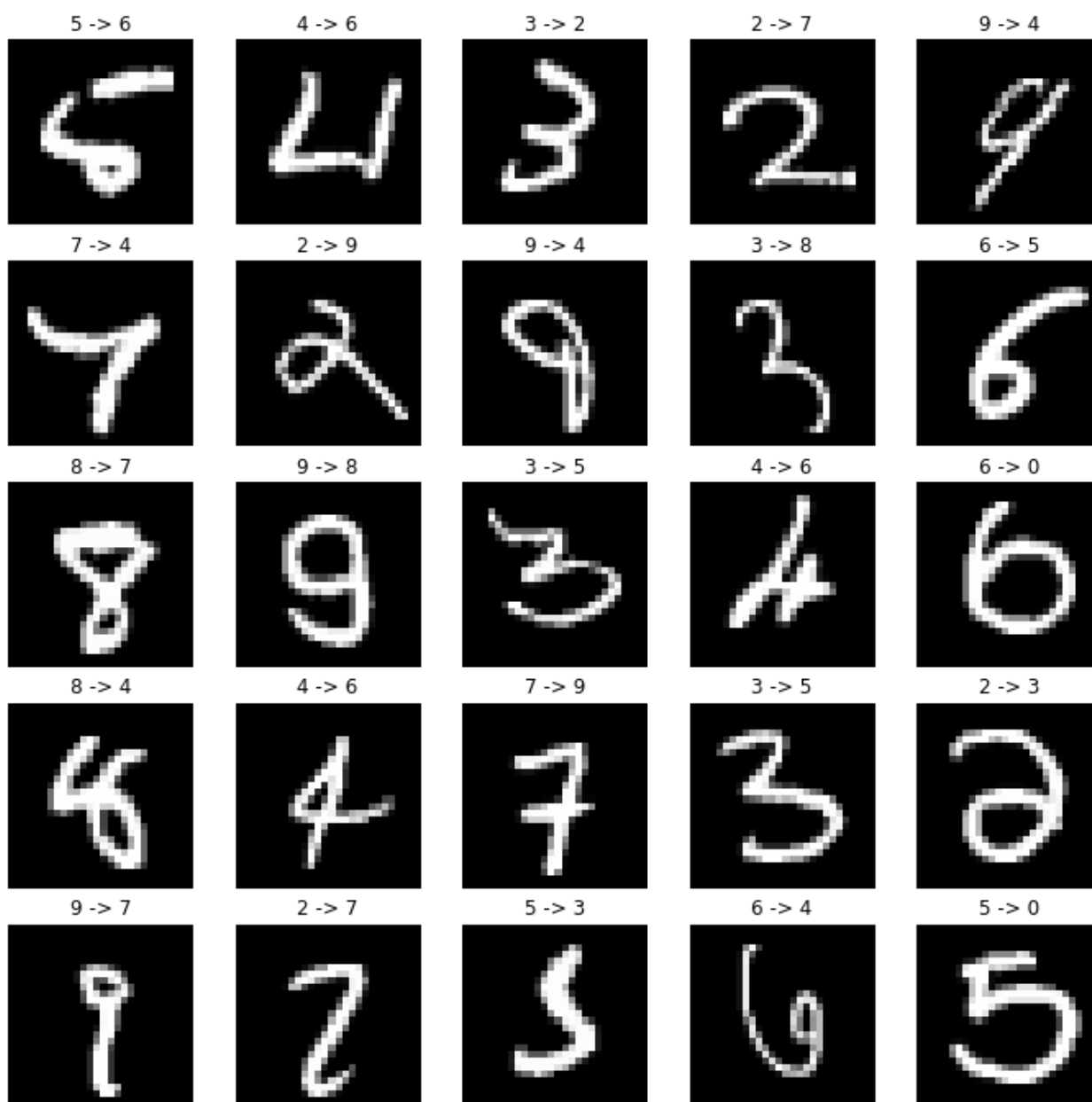
	precision	recall	f1-score	support
0	0.94	0.98	0.96	980
1	0.96	0.97	0.97	1135
2	0.94	0.87	0.90	1032
3	0.90	0.90	0.90	1010
4	0.90	0.93	0.92	982
5	0.90	0.85	0.88	892
6	0.92	0.95	0.94	958
7	0.92	0.91	0.92	1028
8	0.86	0.89	0.87	974
9	0.89	0.88	0.89	1009
accuracy			0.92	10000
macro avg	0.91	0.91	0.91	10000
weighted avg	0.92	0.92	0.92	10000

Question (b)

(i) Experiment with the architecture of the following model to improve its accuracy (it can be increased to as high as ~95%). You can use the relu activation function in the internal layers and

reduce the size of those layers. Please include the outputs of the original and modified models in your submission.

```
inp  =  layers.Input(shape=(28,28,1),   name='InputLayer')
x  =  layers.Flatten(name='FlattenLayer')(inp)
x  =  layers.Dense(128,  activation='relu'  ,name='DenseLayer1')(x)
x  =  layers.Dense(64,  activation='relu'  ,name='DenseLayer2')(x)
outp  =  layers.Dense(10,  activation='softmax'  ,name='OutputLayer')(x)

model  =  Model(inp,  outp,  name='DenseModel')
model.compile(loss=losses.CategoricalCrossentropy(),  optimizer=optimizers.SGD(),  metrics=['accura

model.summary()
plot_model(model,  show_shapes=True,  to_file='DenseModel.png')
```

```
Model: "DenseModel"
_____
 Layer (type)                Output Shape              Param #
=================================================================
 InputLayer (InputLayer)     [(None, 28, 28, 1)]       0

 FlattenLayer (Flatten)      (None, 784)               0
```

!mkdir DenseModelCheckPoints

```
#training model
history = model.fit(train_x, one_hot_train_y,
                               validation_data=(test_x, one_hot_test_y),
                               epochs=Epochs,
                               batch_size=BatchSize,
                               callbacks=[
                                        ModelCheckpoint(filepath='DenseModelCheckPoints/bes
                                        ModelCheckpoint(filepath='DenseModelCheckPoints/den
```

```
Epoch 1/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.6221 - accuracy: 0.8314 -
Epoch 2/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.2941 - accuracy: 0.9155 -
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2402 - accuracy: 0.9312 -
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2054 - accuracy: 0.9414 -
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1807 - accuracy: 0.9480 -
Epoch 6/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1608 - accuracy: 0.9539 -
Epoch 7/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.1458 - accuracy: 0.9589 -
Epoch 8/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.1334 - accuracy: 0.9624 -
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1224 - accuracy: 0.9654 -
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1130 - accuracy: 0.9678 -
```
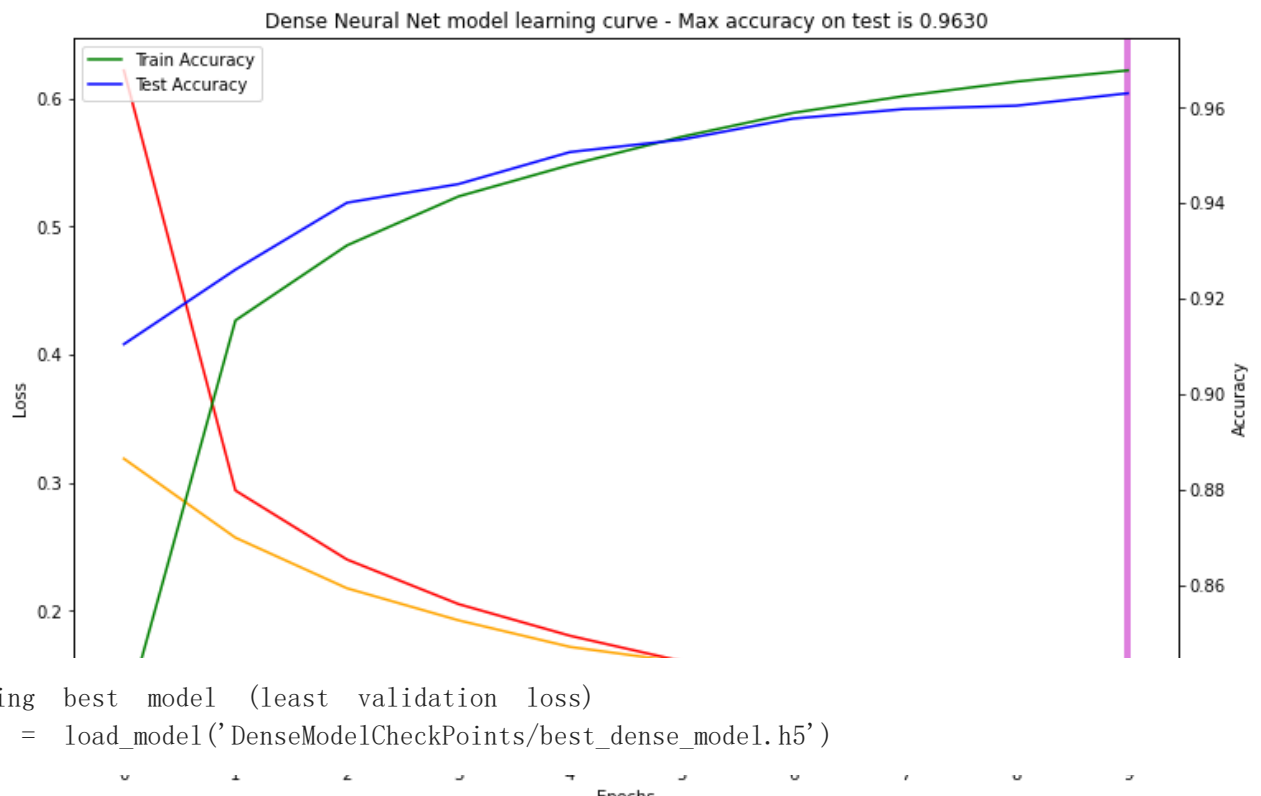
```
#plotting learning curves and labelling them
filename='DenseModelLearningCurve.png'
model_name = 'Dense Neural Net'
plot_history(history, filename, model_name)
```

Dense Neural Net model learning curve - Max accuracy on test is 0.9630

```
#loading  best  model  (least  validation  loss)
model  =  load_model('DenseModelCheckPoints/best_dense_model.h5')
```

```
#evaluating  model  on  test  set
test_y_hat  =  model.predict(test_x)
print('test_y_hat  shape  is  :  '  +  str(test_y_hat.shape))
```

```
    313/313 [==============================] - 0s 1ms/step
    test_y_hat shape is : (10000, 10)
```

```
#finding  max  and  min  of  predictions
test_y_hat  =  np.argmax(test_y_hat,  axis=1)
print('Now  test_y_hat  shape  is  :  %s  (min  =  %d,  max  =  %d)'  %  (str(test_y_hat.shape),  t
```

```
    Now test_y_hat shape is : (10000,) (min = 0, max = 9)
```

```
#plotting  first  25  samples  with  labels
plot_first25labels(test_x,  test_y,  test_y_hat)
```

First 25 samples of MNIST test dataset and their estimated labels
Actual lable -> Estimated label

7 -> 7	2 -> 2	1 -> 1	0 -> 0	4 -> 4

1 -> 1	4 -> 4	9 -> 9	5 -> 6	9 -> 9

0 -> 0	6 -> 6	9 -> 9	0 -> 0	1 -> 1

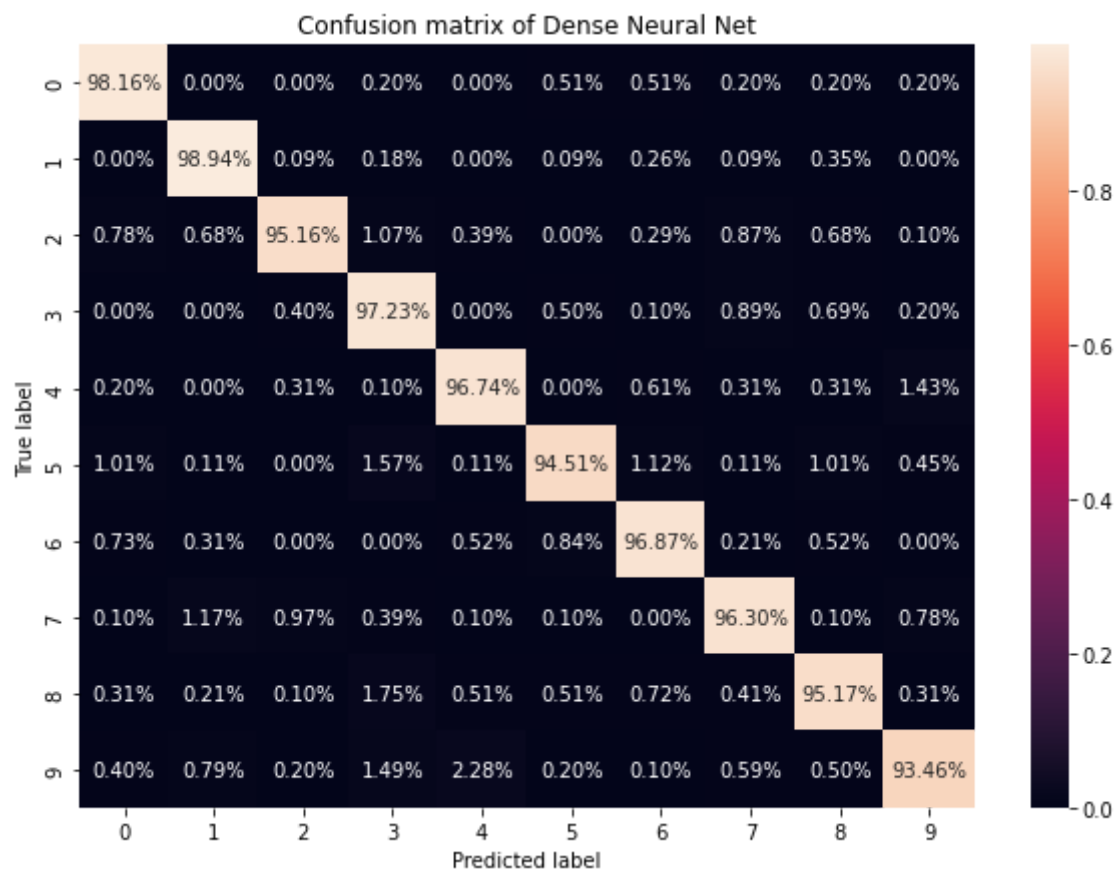| 5 -> 5	9 -> 9	7 -> 7	3 -> 3	4 -> 4

```
plot_mislabeled(test_x,  test_y,  test_y_hat)
```

First 25 samples of MNIST test dataset that the model mislabeled
Actual lable -> Estimated label

5 -> 6	7 -> 4	9 -> 8	8 -> 3	9 -> 8
4 -> 2	6 -> 0	4 -> 6	9 -> 1	2 -> 7

```
#  Show  the  confusion  matrix
plot_confusion(test_y,   test_y_hat)
```
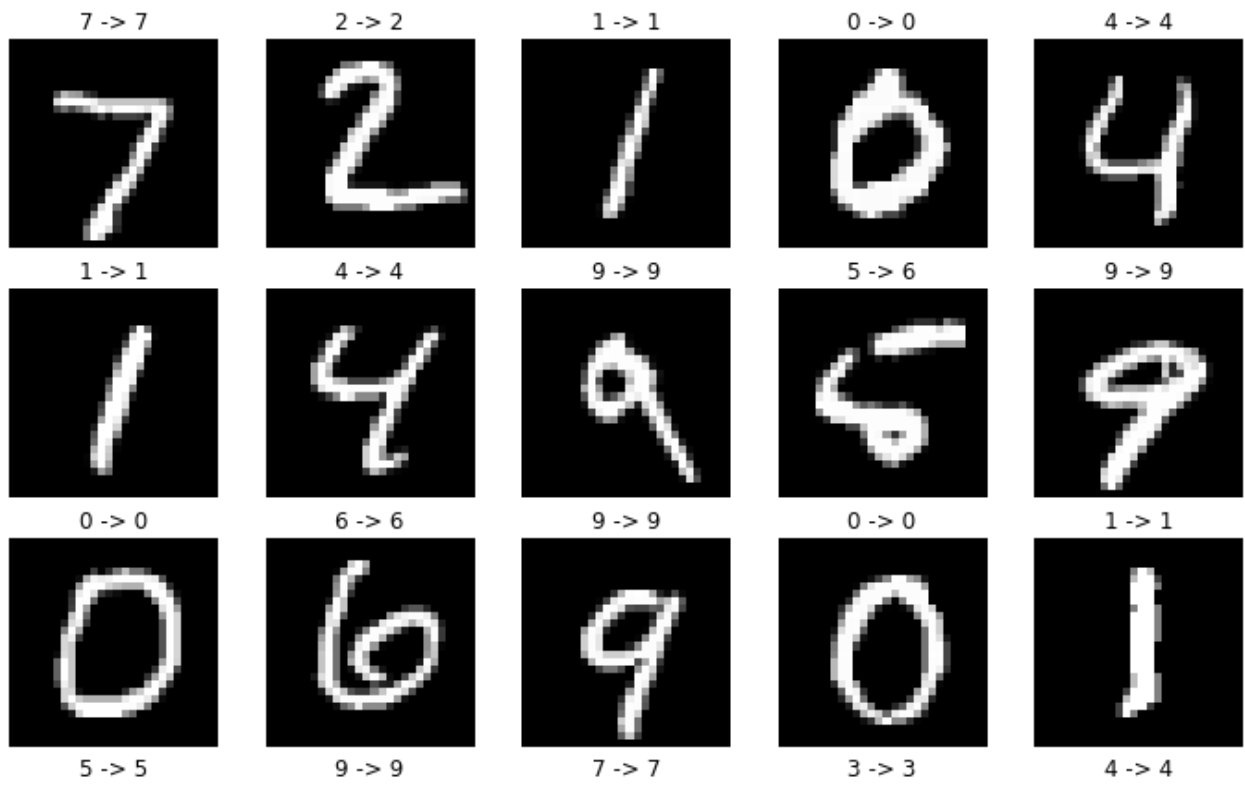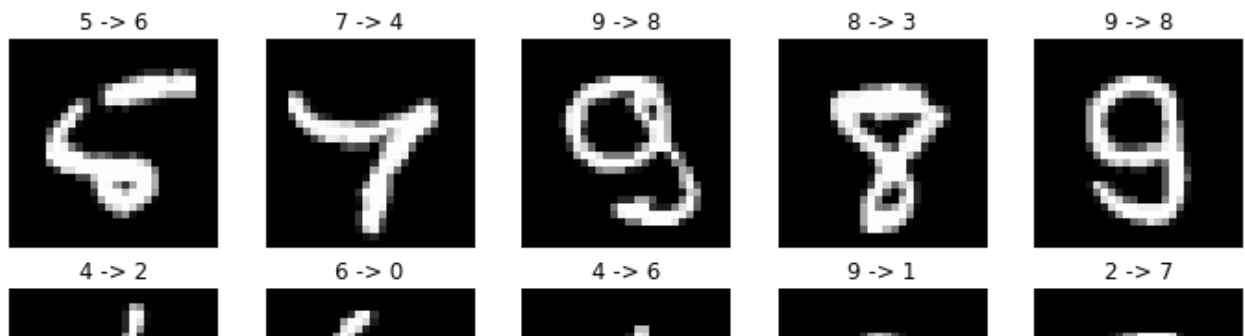

Confusion matrix of Dense Neural Net

```
#printing  classification  report
print(classification_report(test_y,  test_y_hat,  labels=[i  for  i  in  range(10)]))
```

	precision	recall	f1-score	support
0	0.97	0.98	0.97	980
1	0.97	0.99	0.98	1135
2	0.98	0.95	0.97	1032
3	0.94	0.97	0.95	1010
4	0.96	0.97	0.96	982
5	0.97	0.95	0.96	892
6	0.96	0.97	0.97	958
7	0.96	0.96	0.96	1028
8	0.96	0.95	0.95	974

9	0.97	0.93	0.95	1009
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

Question (c)

Develop and train a CNN model for the above classification task. You can use the architecture below in the comments with Relu activation between internal layers and softmax in the output layer. You can get ~98% accuracy. Use/review the keras documentation for, e.g., the following functions

layers.Conv2D(filters=32, kernel_size=(3,3), activation='relu', name='Conv1')(inp)

layers.MaxPool2D(pool_size=(2,2), strides=(2,2), name='MaxPooling1')(x)

```
#Model: "ConvModel"
#_____
# Layer  (type)                       Output  Shape                     Param  #
#================================================================
#  InputLayer  (InputLayer)          [(None,  28,  28,  1)]              0
#
#  Conv1  (Conv2D)                    (None,  26,  26,  32)              320
#                                        i.e.,  filters=32,  kernel_size=(3,3)
#
#  MaxPooling1  (MaxPooling2D)     (None,  13,  13,  32)                0
#                                        i.e.,  pool_size=(2,2),  strides=(2,2)
#
#  Conv2  (Conv2D)                    (None,  11,  11,  64)              18496
#
#  MaxPooling2  (MaxPooling2D)     (None,  3,  3,  64)                 0
#                                        i.e.,  filters=64,  kernel_size=(3,3)
#
#  FlattenLayer  (Flatten)          (None,  576)                        0
#
#  DenseLayer1  (Dense)             (None,  128)                        73856
#
#  OutputLayer  (Dense)             (None,  10)                         1290
inp =  layers.Input(shape=(28,28,1),  name='InputLayer')
x =  layers.Conv2D(filters  =  32,  kernel_size  =  (3,3),  activation  =  'relu',  name='Conv1')(

x =  layers.MaxPool2D(pool_size  =  (2,2),  strides  =  (2,2),  name  =  'MaxPooling1')(x)
x =  layers.Conv2D(filters  =  64,  kernel_size  =  (3,3),  activation  =  'relu',  name='Conv2')(
x =  layers.MaxPool2D(pool_size  =  (3,3),  name  =  'MaxPooling2')(x)
x =  layers.Flatten(name='FlattenLayer')(x)
x =  layers.Dense(128,  activation  =  'relu',  name  =  'DenseLayer1')(x)
outp =  layers.Dense(10,  activation='softmax',  name='OutputLayer')(x)
model =  Model(inp,  outp,  name='ConvModel')
model.compile(loss  =  losses.CategoricalCrossentropy(),  optimizer  =  optimizers.SGD(),  metrics=
model.summary()
```

```
plot_model(model,  show_shapes=True,  to_file  =  'ConvModel.png')
```

```
Model: "ConvModel"
_____
 Layer (type)                Output Shape              Param #
=================================================================
 InputLayer (InputLayer)     [(None, 28, 28, 1)]       0

 Conv1 (Conv2D)              (None, 26, 26, 32)        320

 MaxPooling1 (MaxPooling2D)  (None, 13, 13, 32)        0

 Conv2 (Conv2D)              (None, 11, 11, 64)        18496

 MaxPooling2 (MaxPooling2D)  (None, 3, 3, 64)          0

 FlattenLayer (Flatten)      (None, 576)               0
```

```
!mkdir ConvModelCheckPoints
```

```
 OutputLayer (Dense)         (None, 10)                1290
```

```python
#training the model
history = model.fit(train_x, one_hot_train_y,
                            validation_data=(test_x, one_hot_test_y),
                            epochs=Epochs,
                            batch_size=BatchSize,
                            callbacks=[
                                    ModelCheckpoint(filepath='ConvModelCheckPoints/best
                                    ModelCheckpoint(filepath='ConvModelCheckPoints/conv
```

```
    Epoch 1/10
    1875/1875 [==============================] - 45s 24ms/step - loss: 0.6970 - accuracy: 0.8035
    Epoch 2/10
    1875/1875 [==============================] - 49s 26ms/step - loss: 0.1488 - accuracy: 0.9539
    Epoch 3/10
    1875/1875 [==============================] - 45s 24ms/step - loss: 0.1120 - accuracy: 0.9662
    Epoch 4/10
    1875/1875 [==============================] - 43s 23ms/step - loss: 0.0946 - accuracy: 0.9709
    Epoch 5/10
    1875/1875 [==============================] - 42s 23ms/step - loss: 0.0830 - accuracy: 0.9749
    Epoch 6/10
    1875/1875 [==============================] - 44s 24ms/step - loss: 0.0756 - accuracy: 0.9766
    Epoch 7/10
    1875/1875 [==============================] - 44s 24ms/step - loss: 0.0679 - accuracy: 0.9789
    Epoch 8/10
    1875/1875 [==============================] - 43s 23ms/step - loss: 0.0629 - accuracy: 0.9807
    Epoch 9/10
    1875/1875 [==============================] - 43s 23ms/step - loss: 0.0585 - accuracy: 0.9816
    Epoch 10/10
    1875/1875 [==============================] - 43s 23ms/step - loss: 0.0544 - accuracy: 0.9831
```
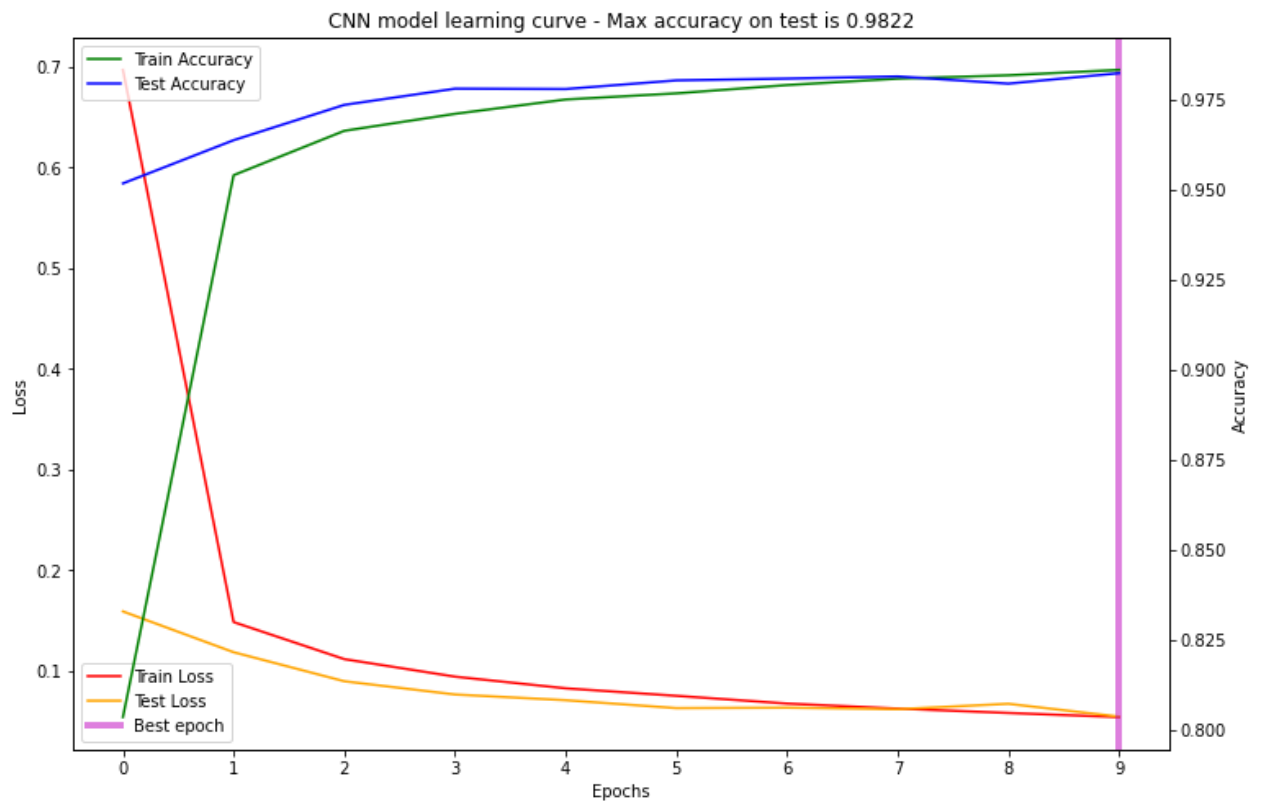
```python
#plotting learning curves and labelling them
filename='ConvModelLearningCurve.png'
model_name = 'CNN'
plot_history(history, filename, model_name)
```

CNN model learning curve - Max accuracy on test is 0.9822

```
#loading  best  model
model  =  load_model('ConvModelCheckPoints/best_conv_model.h5')


#finding  shape  of  prediction
test_y_hat  =  model.predict(test_x)
print('test_y_hat  shape  is  :  '  +  str(test_y_hat.shape))

    313/313 [==============================] - 3s 8ms/step
    test_y_hat shape is : (10000, 10)


#finding  minimum  and  maximum  prediction  values
test_y_hat  =  np.argmax(test_y_hat,  axis=1)
print('Now  test_y_hat  shape  is  :  %s  (min  =  %d,  max  =  %d)'  %  (str(test_y_hat.shape),  t

    Now test_y_hat shape is : (10000,) (min = 0, max = 9)


#plotting  first  25  samples
plot_first25labels(test_x,  test_y,  test_y_hat)
```
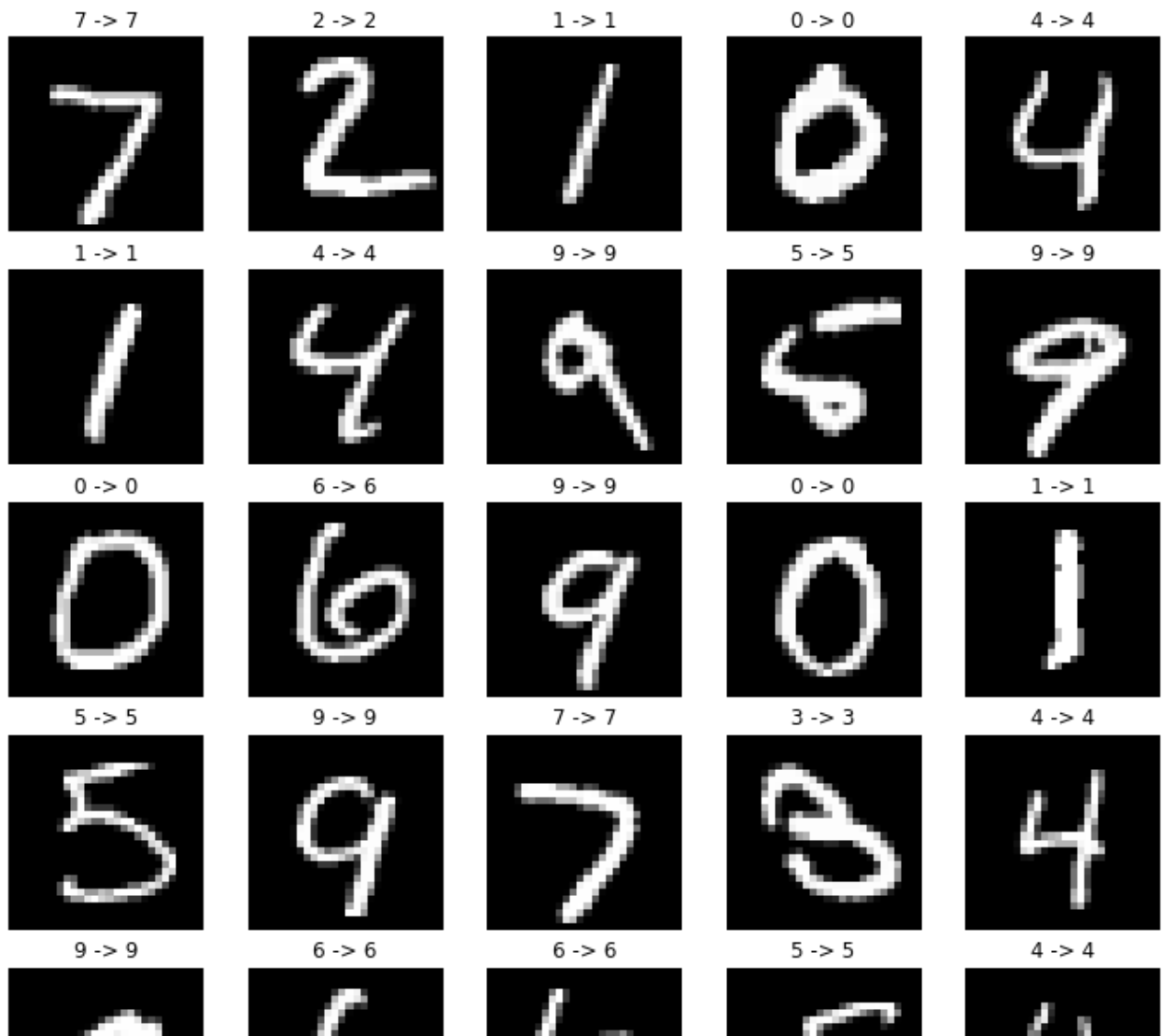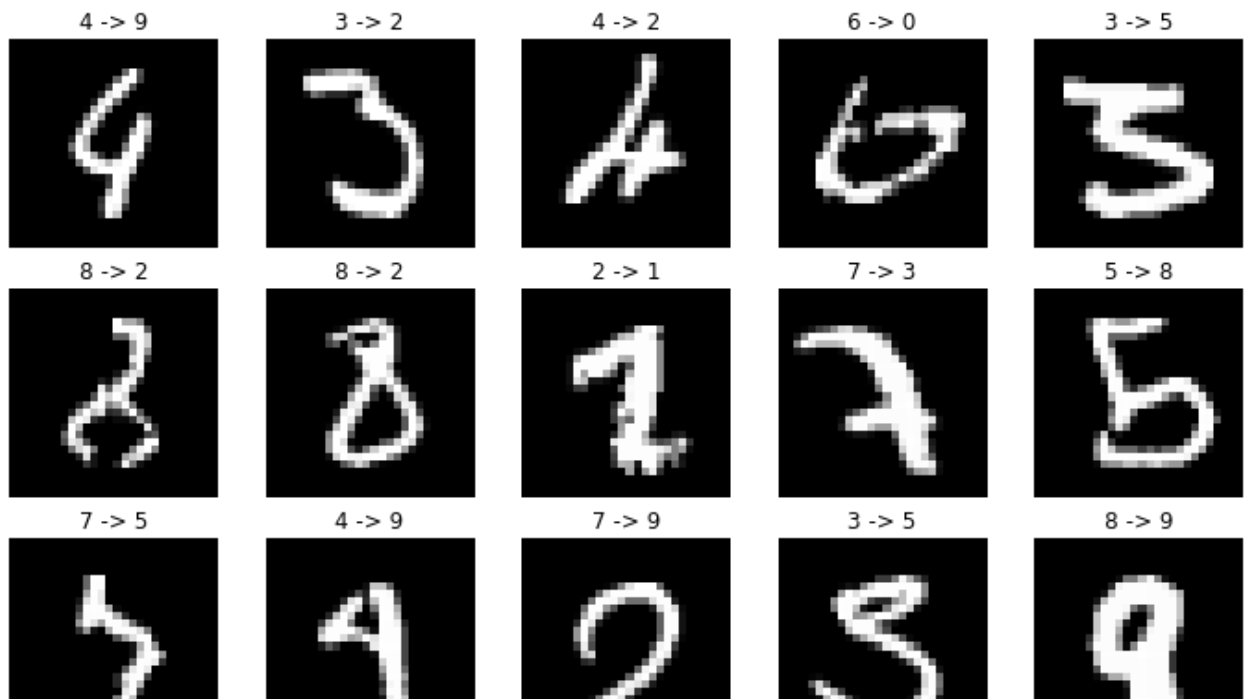
First 25 samples of MNIST test dataset and their estimated labels
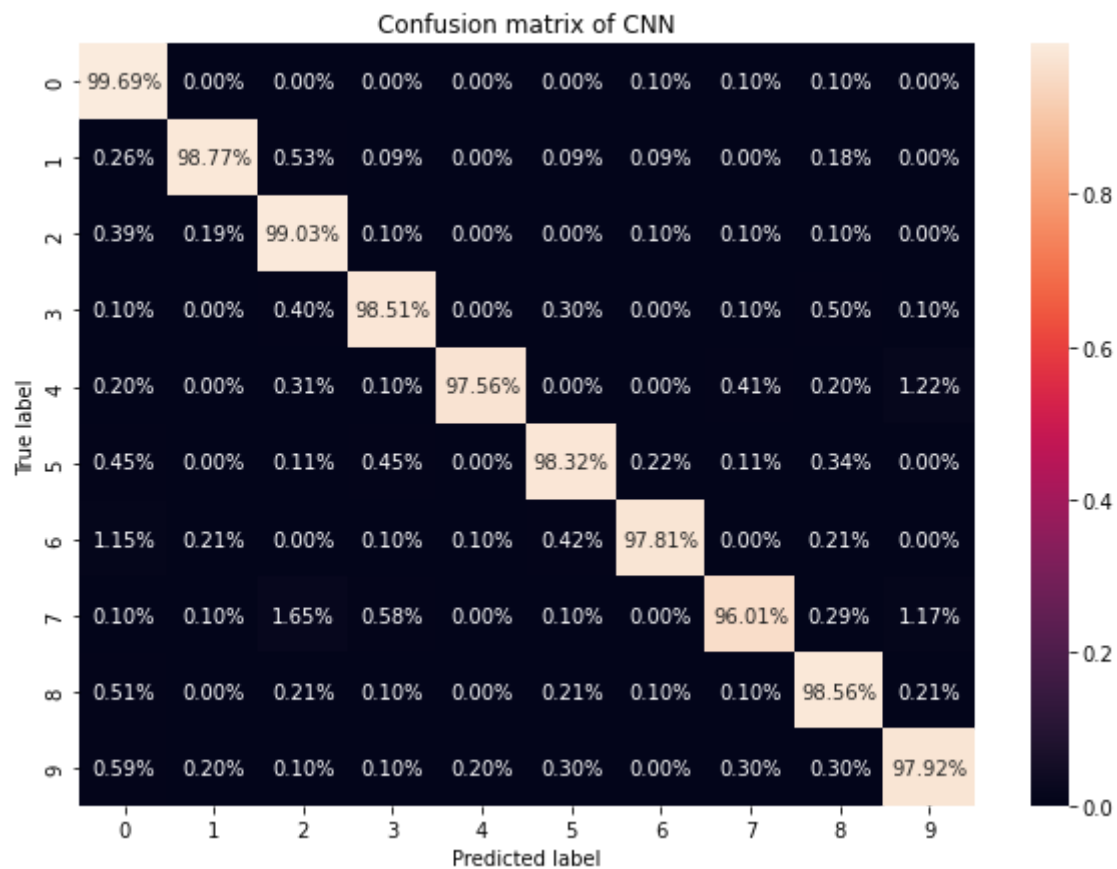Actual lable -> Estimated label


```
#plotting first 25 mislabeled images
plot_mislabeled(test_x, test_y, test_y_hat)
```

First 25 samples of MNIST test dataset that the model mislabeled
Actual lable -> Estimated label

```
#plotting   confusion   matrix
plot_confusion(test_y,   test_y_hat)
```


Confusion matrix of CNN

```
#plotting   performance
print(classification_report(test_y,   test_y_hat,   labels=[i   for   i   in   range(10)]))
```

                  precision   recall f1-score   support

```
              0       0.96      1.00      0.98       980
              1       0.99      0.99      0.99      1135
              2       0.97      0.99      0.98      1032
              3       0.98      0.99      0.98      1010
              4       1.00      0.98      0.99       982
              5       0.98      0.98      0.98       892
              6       0.99      0.98      0.99       958
              7       0.99      0.96      0.97      1028
              8       0.98      0.99      0.98       974
              9       0.97      0.98      0.98      1009

       accuracy                           0.98     10000
      macro avg       0.98      0.98      0.98     10000
   weighted avg       0.98      0.98      0.98     10000
```

Double-click (or enter) to edit

✓   0s   completed at 4:11 PM   ● ✕