Quiz 2

1. The cross entropy loss function is $L(w) = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(g_i) + (1-y_i)\log(1-g_i)$
where $g_i = \sigma(w^T x_i) = 1/(1+e^{-w^T x_i})$ for $(x_1, y_1) \dots (x_n, y_n)$ in $\mathbb{R}^n \times \{0,1\}$.
To show that this is convex, set $t = w^T x_i$.
$\partial/\partial t\, g_i = \partial/\partial t\, (1+e^{-t})^{-1} = e^{-t}(1+e^{-t})^{-2} = g_i(1-g_i)$
$\partial \log(g_i)/\partial w^T = 1/g_i\ \partial g_i/\partial w^T = 1/g_i\ \partial g_i/\partial t\ \partial t/\partial w^T = (1-g_i)x_i$
$\partial \log(1-g_i)/\partial w^T = 1/g_i\ \partial(1-g_i)/\partial w^T = -g_i x_i$
Summation component $l_i(w) = -y_i \log(g_i) - (1-y_i)\log(1-g_i)$
$\nabla l_i(w) = -y_i x_i(1-g_i) + (1-y_i)x_i g_i = x_i(g_i - y_i)$.
$\nabla^2 l_i(w) = x_i x_i^T g_i(1-g_i) = \frac{1}{n}\sum_{i=1}^{m} \nabla^2 l_i(w) = \frac{1}{n}\sum_{i=1}^{m} x_i x_i^T g_i(1-g_i) \equiv XDX^T$
where D is a diagonal matrix with all entries $D_{ii} = g_i(1-g_i) > 0$.
So $\nabla^2 L(w)$ is positive semidefinite and $L(w)$ is convex.
$L(w)$ is not strongly convex. So min of a convex function must achieve global min.
For gradient descent method $x_{k+1} = x_k - s_k \nabla L_{B_k}(x_k)$, $w_{k+1} = w_k - s_k \nabla L(w_k)$.
$L(w) \in C^2$, $|\lambda_i| \le M$ for all λ_i of $\nabla^2 L(w)$. Use backtracking line search to find s_k. # iterations $= O(1/\varepsilon)$ for gradient decent to reach $|L(w_k) - L(w^*)| < \varepsilon$.
Adding an l_2 regularization term we get cross-entropy loss $L(w)$ due to convexity. So converge rate becomes $O(\log(1/\varepsilon))$ for reaching $|L(w_k) - L(w^*)| < \varepsilon$.

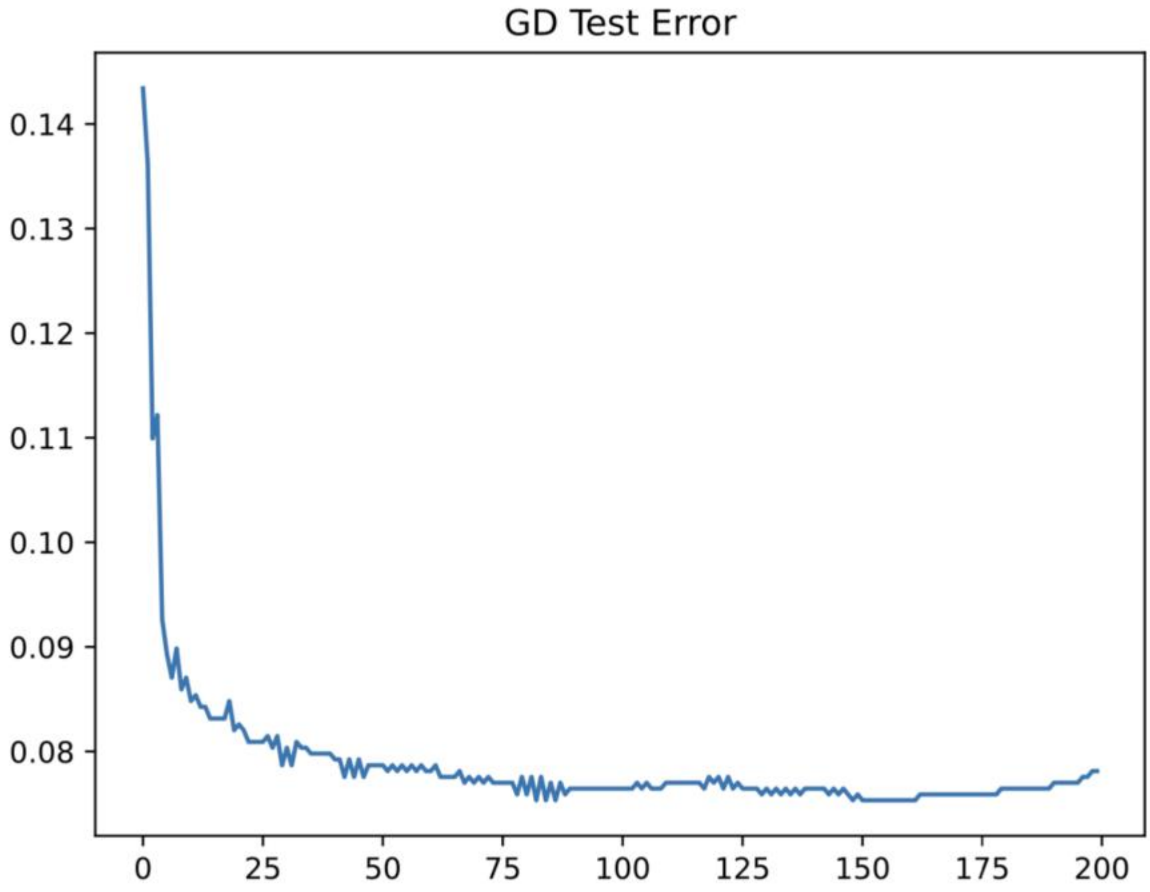
2. (b) With $L(W) = \frac{1}{n}\sum_{i=1}^{m} l_i(W)$,

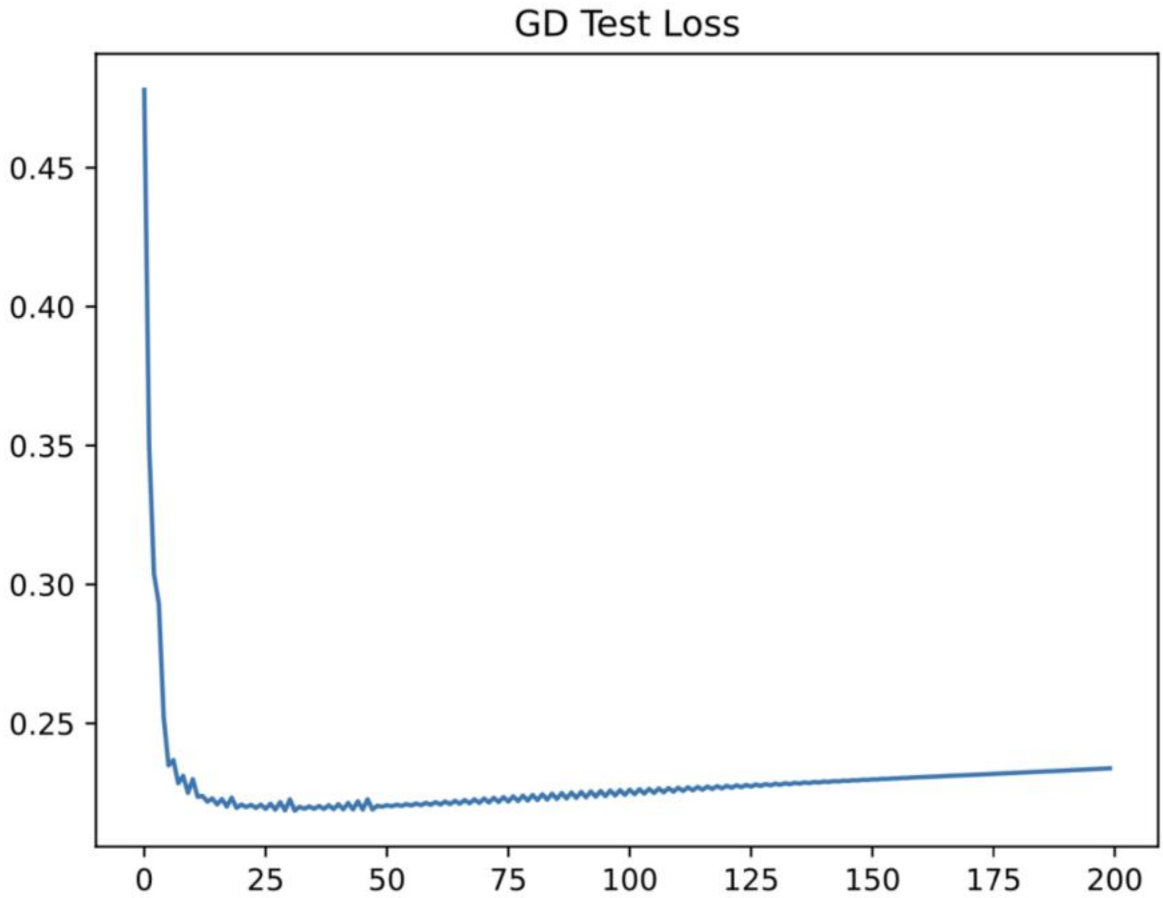
$\nabla l_i(W) = -y_i x_i (1-g_i) + (1-y_i) x_i g_i = x_i(g_i - y_i)$

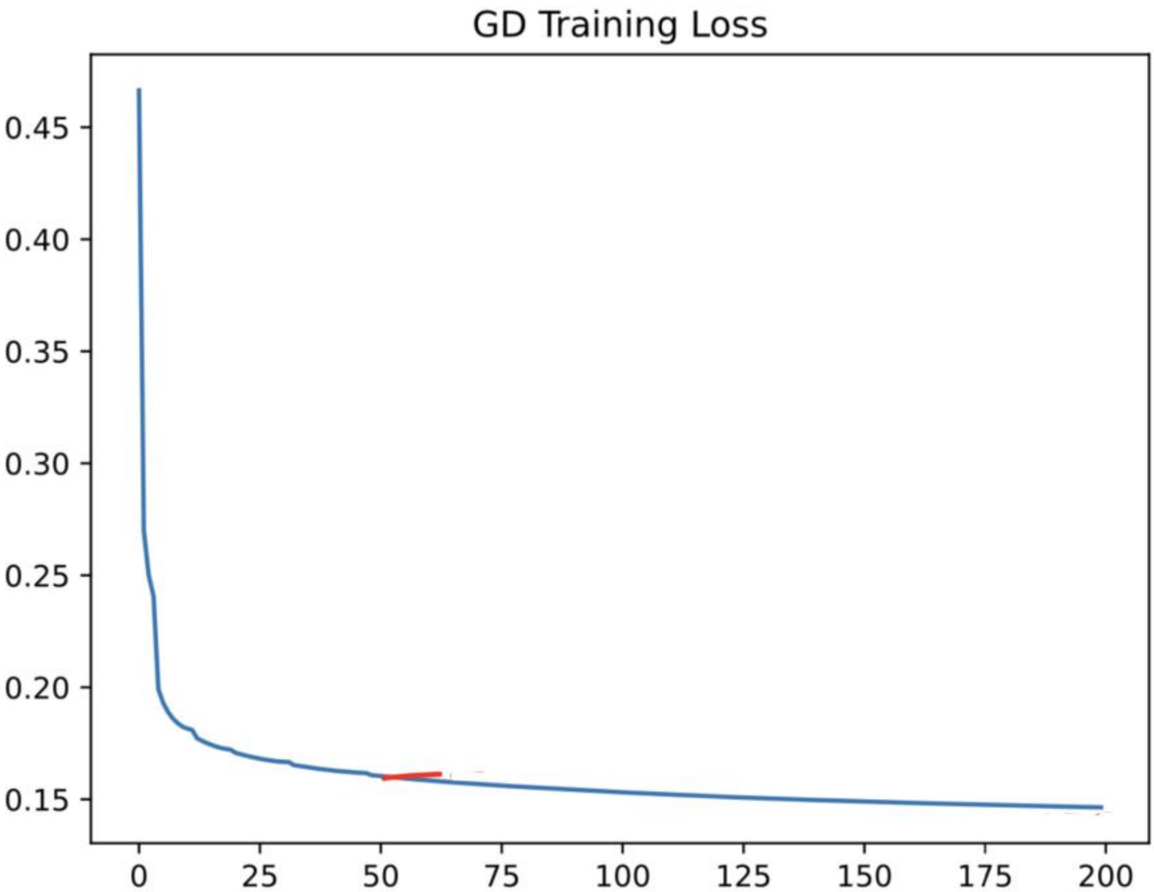
$= \frac{1}{n}\sum_{i=1}^{m} x_i(g_i - y_i)$ where $g_i = (1+e^{-W^T x_i})^{-1}$

(e) From the output data, the resulting loss becomes stable or converges at roughly 50 iterations. This is in accordance with $O(1/\varepsilon)$ for $|L(W_K) - L(W^*)| < \varepsilon$.

Problem 2 Outputs:






GD Training Loss
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0
25
50
75
100
125
150
175
200

Problem 2

(a) (c) (d)

```
from mnist_tools import *
import numpy as np
import matplotlib.pyplot as plt
import time
from scipy.special import logsumexp

"""
Sigmoid function that takes a numpy array of any shape.
"""
def f(t) :
    return 1/(1+np.exp(-t))

"""
Forecast function which given the learned parameter vectors w and
the data x produces the forecasts.
"""
def h(w,x) :
    return f(np.dot(x,w))>0.5

"""
Computes the loss function L.
Parameters:
w: numpy array of length m containing the parameter vector
X: numpy array of shape (n,m) containing n data samples as rows (each row is a data
y: numpy array of length n containing the labels (0 or 1)
Returns:
A single float, the loss evaluated on the given arguments.
"""
def L(w,X,y):
    n = len(y)
    sum = 0
    for i in range(n):
        g_i = np.dot(w, X[i])
        term = y[i] * np.logaddexp(0, - g_i) + (1 - y[i]) * np.logaddexp(0, g_
        sum = sum + term / n
    return sum

"""
Tests the L function
"""
def test_L() :
    np.random.seed(1000)
    v = np.array([1000])
    w = np.random.randn(10)
    X = np.random.randn(20,10)
    y = np.random.randint(0,2,20)
```

```
    L1 = L(v,v,np.array([0]))
    L2 = L(v,v,np.array([1]))
    L3 = L(w,X,y)
    assert np.abs(L1-1000000) < 1e-9
    assert np.abs(L2) < 1e-9
    assert np.abs(L3-1.08007365415) < 1e-9

"""
Computes the gradient of the loss function.
Parameters:
w: numpy array of length m containing the parameter vector
X: numpy array of shape (n,m) containing n data samples as rows (each row is a data
y: numpy array of length n containing the labels (0 or 1)
Returns:
A numpy vector of length m containing the gradient of the
loss evaluated on the given arguments.
"""
def dL(w,X,y) :
    n = len(y)
    sum = 0
    for i in range(n):
        g_i = np.dot(w, X[i])
        term = X[i] * (f(g_i) - y[i])
        sum = sum + term / n
    return sum

"""
Tests the dL function
"""
def test_dL() :
    np.random.seed(1000)
    v = np.array([1000])
    w = np.random.randn(3)
    X = np.random.randn(200,3)
    y = np.random.randint(0,2,200)
    dL1 = dL(v,v,np.array([0]))
    dL2 = dL(v,v,np.array([1]))
    dL3 = dL(w,X,y)
    assert np.abs(dL1-1000) < 1e-9
    assert np.abs(dL2) < 1e-9
    assert np.linalg.norm(dL3-np.array([-0.12669153,-0.00341384,0.02274541])) < 1e-6

"""
Runs (batch) gradient descent with a backtracking line search to minimize L.
While typically this would include conditions/tolerances for how to stop the
algorithm, here we only required a simplified implementation that has a given fixed
number of steps.
Parameters:
w0: numpy array of length m containing the initial value of w
X: numpy array of shape (n,m) containing the n data samples as rows
y: numpy array of length n containing the labels (0 or 1)
num_steps: number of gradient descent steps to run
alpha: Armijo constant used to make sure the L function sufficiently decreases on each
iteration
beta: backtracking line search constant that determines how much to shrink the step
```

```
size parameter by each time
Returns: the tuple w,ws where
w: numpy array of length m containing the final value of w
ws: a python list of num_steps numpy arrays of length m containing the w-values compu
at each iteration
"""



def gradient_descent(w0,X,y,num_steps=200,alpha=0.01,beta=0.5) :
        w_s = []
        w = w0
        for i in range(num_steps):
                t = 1
                while (L(w, X, y) - L(w - t * dL(w, X, y), X, y) - alpha * t * n
                        t = t * beta
                print("t = ", t)
                w = w - t * dL(w, X, y)
                w_s.append(w)
        return w, w_s



"""
Standarizes the training and test data using the training data to compute
the mean and standard deviation.
"""
def standardize(train,test) :
        m = np.mean(train,axis=0)
        std = np.std(train,axis=0)
        std[np.abs(std)<1e-9] = 1
        return (train-m)/std,(test-m)/std,m,std

"""
Runs the optimization and creates the plots
"""
def run(name,fun,train_x,train_y,test_x,test_y,mean,std) :
        t = time.time()
        g_w,g_ws = fun(np.zeros(train_x.shape[1]),train_x,train_y)
        print('%s Time = %fs'%(name,time.time()-t))
        print('%s Training Loss = %f'%(name,L(g_w,train_x,train_y)))
        test_err = np.sum(np.abs(h(g_w,test_x)-test_y))*1.0/test_x.shape[0]
        print('%s Test Error = %f'%(name,test_err))
        ls = [L(w,train_x,train_y) for w in g_ws]
        tls = [L(w,test_x,test_y) for w in g_ws]
        terr = [np.sum(np.abs(h(w,test_x)-test_y))/test_x.shape[0] for w in g_ws]
        plt.plot(ls)
        plt.title('%s Training Loss'%name)
        plt.savefig('%s_Train_Loss.pdf'%name,bbox_inches='tight')
        plt.close()
        plt.title('%s Test Loss'%name)
        plt.plot(tls)
        plt.savefig('%s_Test_Loss.pdf'%name,bbox_inches='tight')
        plt.close()
        plt.plot(terr)
```

```
        plt.title('%s Test Error'%name)
        plt.savefig('%s_Test_Error.pdf'%name,bbox_inches='tight')
        plt.close()

def main() :
        test_L()
        test_dL()

        train = load_train_data("mnist_all.mat")
        test = load_test_data("mnist_all.mat")
        print('Using %d training examples and %d test examples'%(train.shape[0],test.shape[0
        #We will determine if the image is a '5' or not
        train[:,-1] = train[:,-1]==5
        test[:,-1] = test[:,-1]==5
        train_x,train_y = train[:,:-1],train[:,-1]
        test_x,test_y = test[:,:-1],test[:,-1]
        train_x,test_x,mean,std = standardize(train_x,test_x)

        run('GD',gradient_descent,train_x,train_y,test_x,test_y,mean,std)


if __name__ == "__main__" :
        main()
```

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)
```

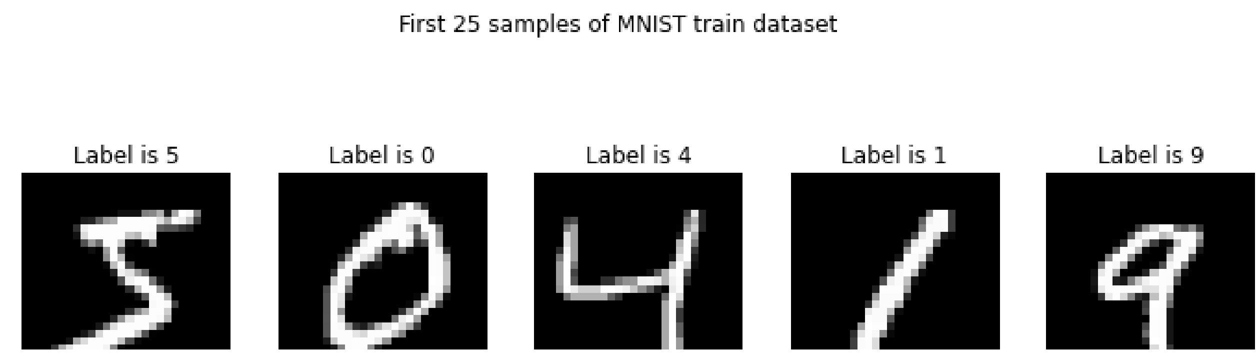
```
    plt.show()

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(
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Shape of train_x is : (60000, 28, 28) (min= 0.00, max= 255.00)
Shape of train_y is : (60000,) (min= 0, max= 9)
```

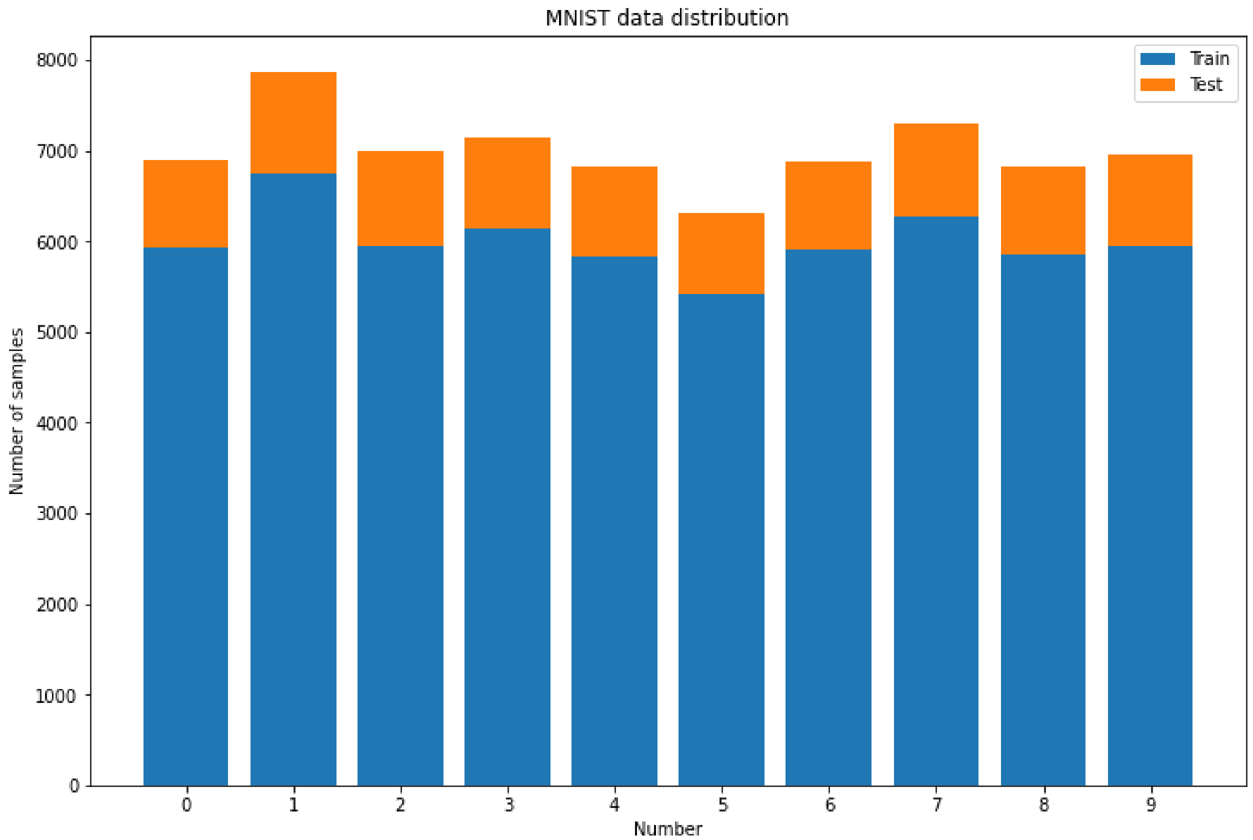
```
#preprocessing input
train_x = train_x.astype('float') / 255.0
test_x = test_x.astype('float') / 255.0

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)
```

```
#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')
```




```
#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()
```




```
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 #
=================================================================
```

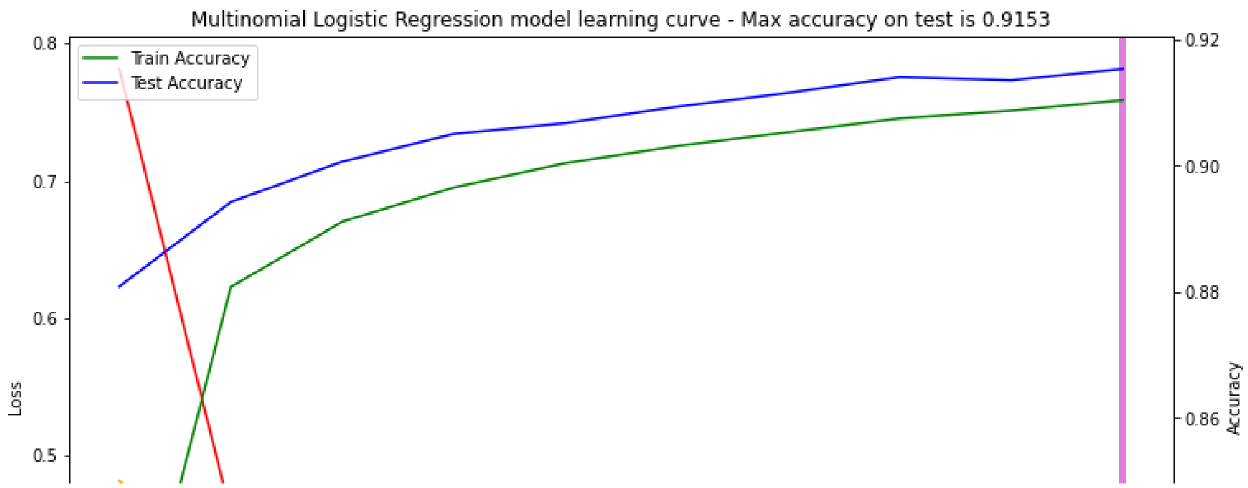
```
!mkdir LinearModelCheckPoints
```

```
 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
```

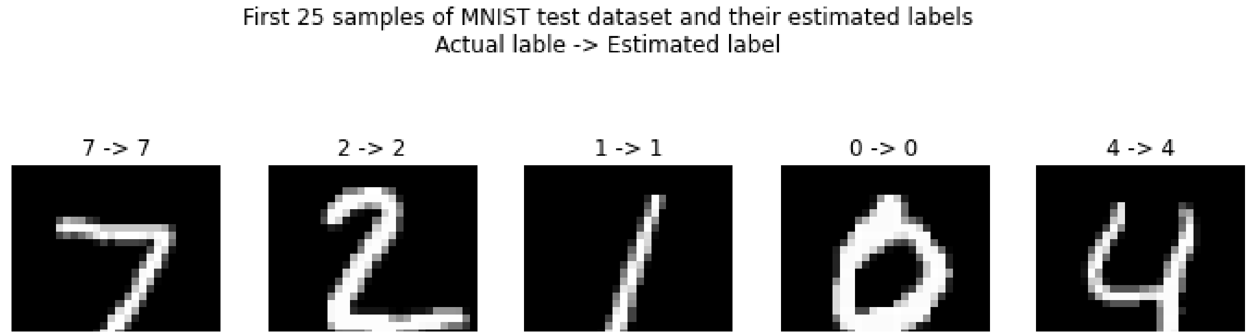
```
Epoch 1/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.7814 - accuracy: 0.8120 -
Epoch 2/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.4574 - accuracy: 0.8807 -
Epoch 3/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.4041 - accuracy: 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)
```

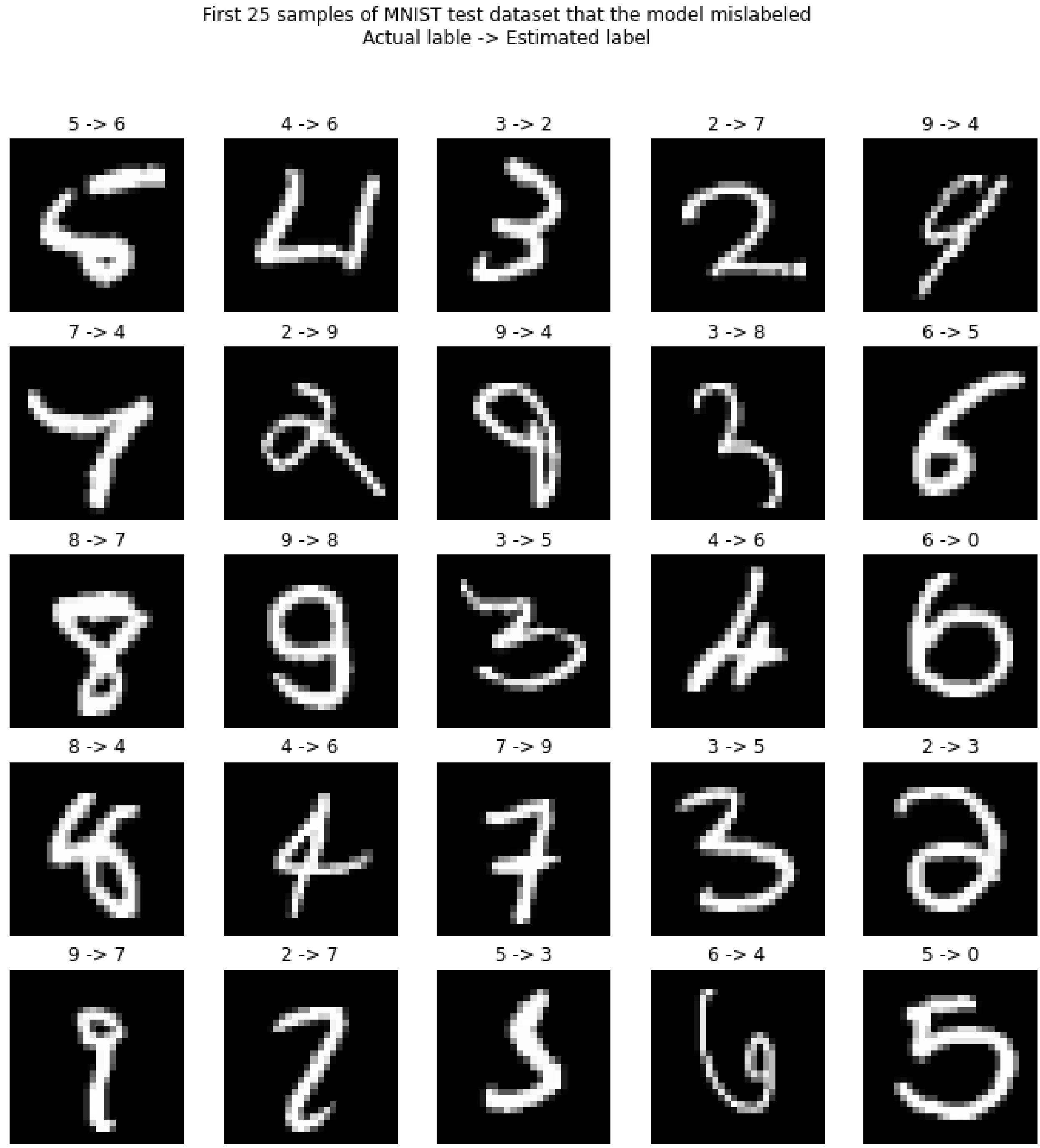



```
#loading best model (least validation loss)
model = load_model('LinearModelCheckPoints/best_dense_model.h5')
#evaluating model on test set
test_y_hat = model.predict(test_x)
print('test_y_hat shape is : ' + str(test_y_hat.shape))
#finding max and min of predictions
test_y_hat = np.argmax(test_y_hat, 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)
```

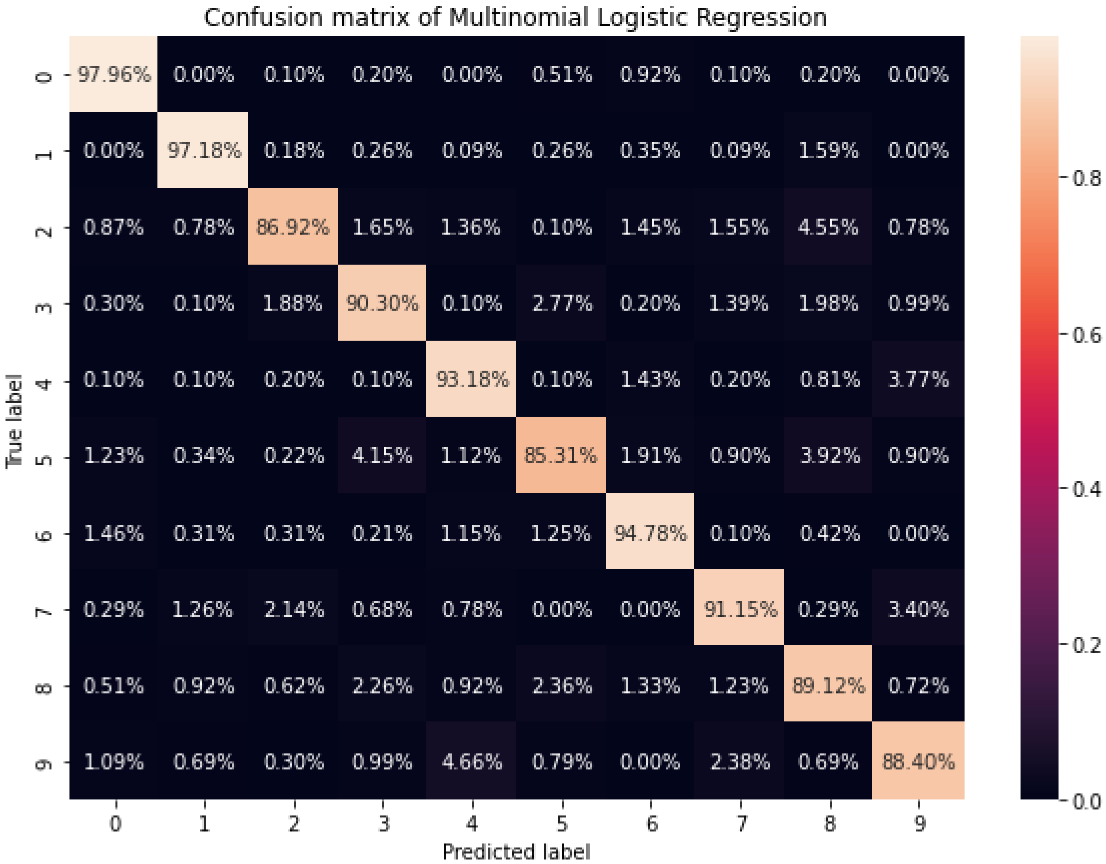
```
313/313 [==============================] - 0s 1ms/step
test_y_hat shape is : (10000, 10)
Now test_y_hat shape is : (10000,) (min = 0, max = 9)
```




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



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




```
#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')
```

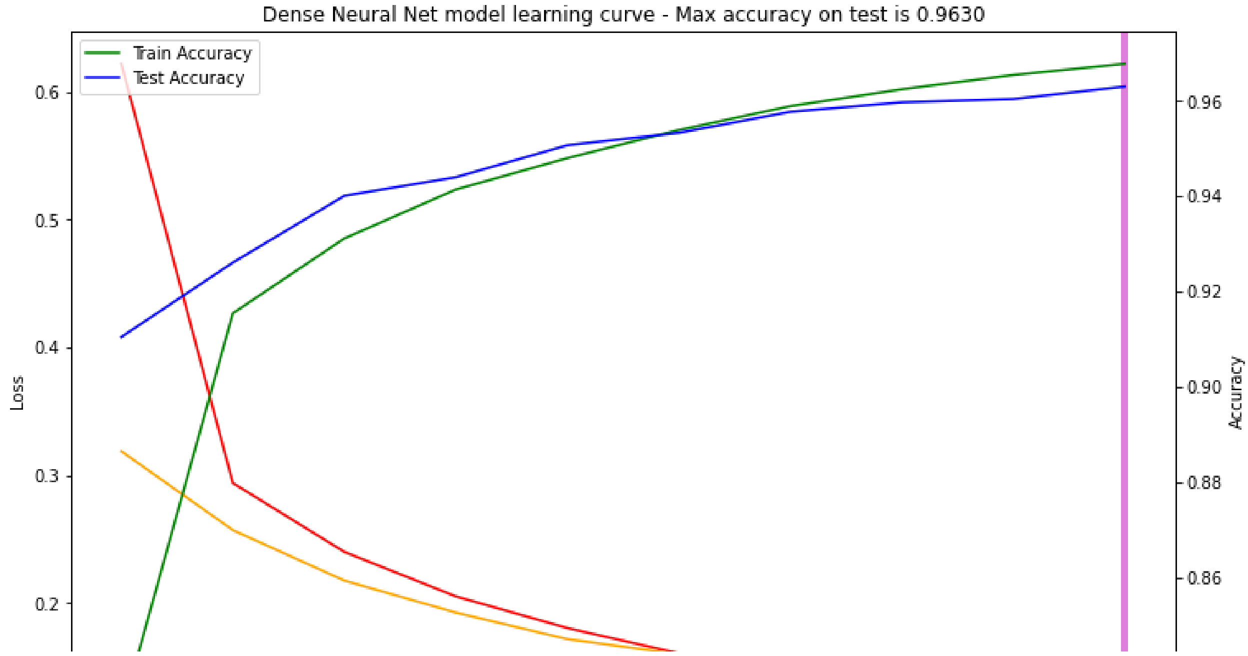
```
Model: "DenseModel"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 InputLayer (InputLayer)     [(None, 28, 28, 1)]       0

 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)
```




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



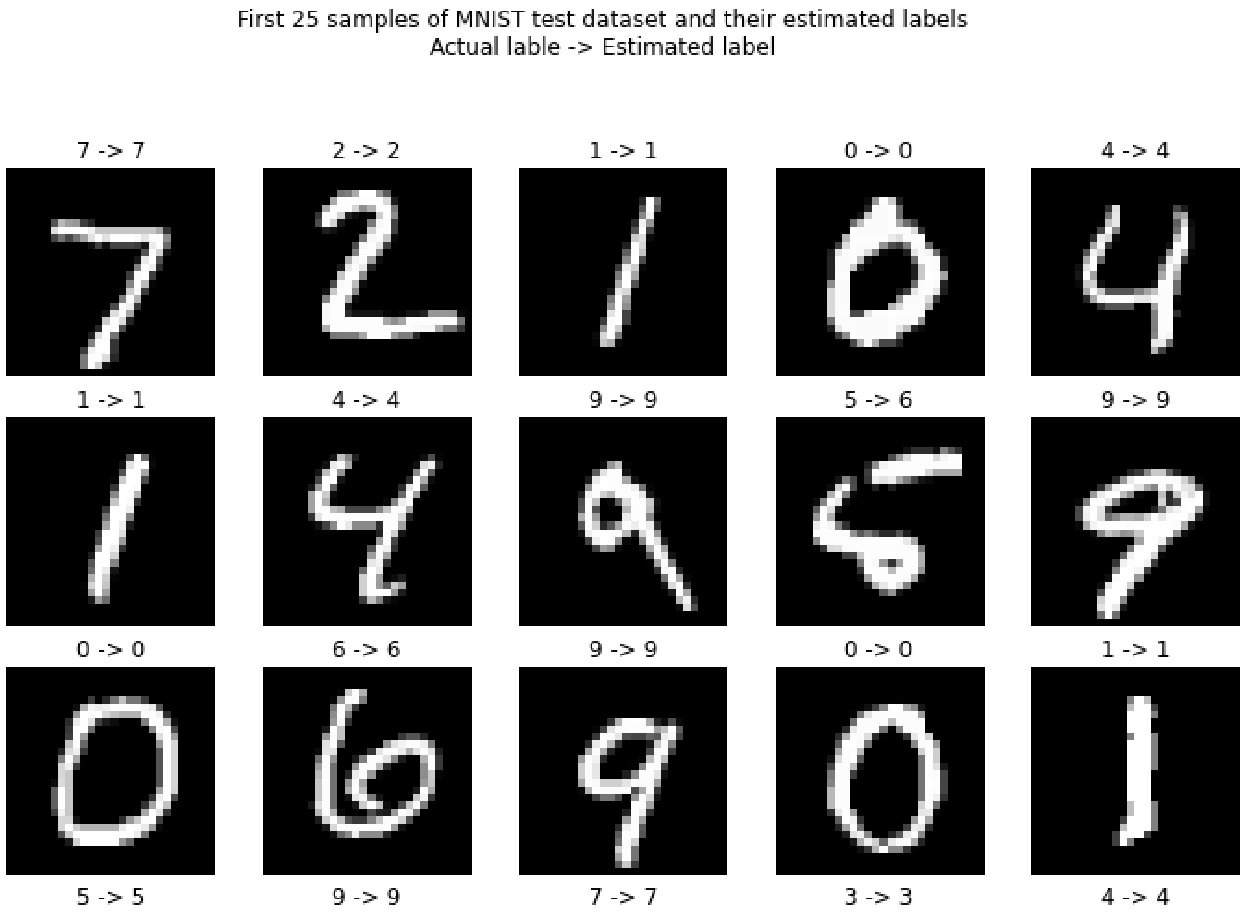
```
#evaluating model on test set
test_y_hat = model.predict(test_x)
print('test_y_hat shape is : ' + str(test_y_hat.shape))
```

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

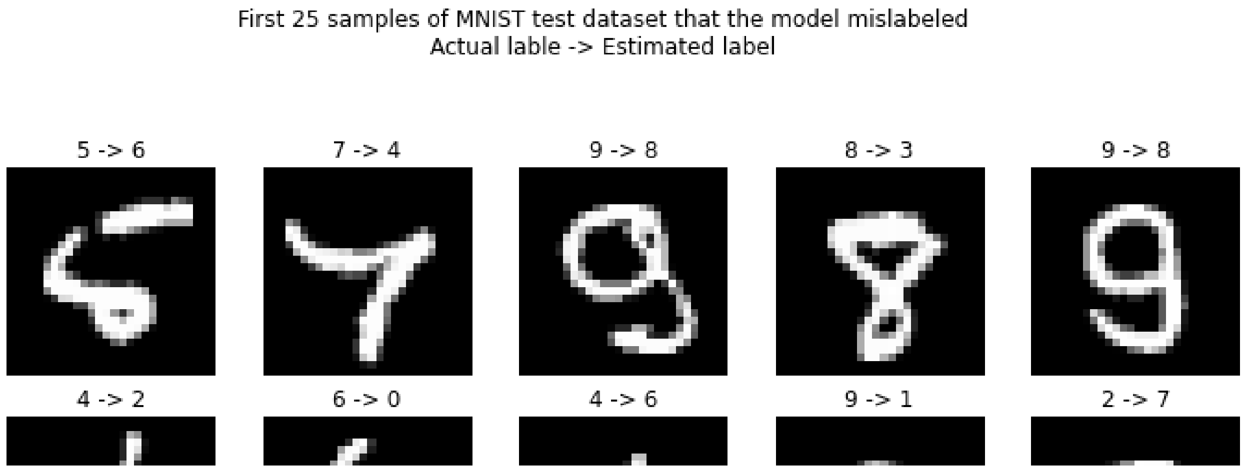
```
#finding max and min of predictions
test_y_hat = np.argmax(test_y_hat, axis=1)
print('Now test_y_hat shape is : %s (min = %d, max = %d)' % (str(test_y_hat.shape), t
```

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

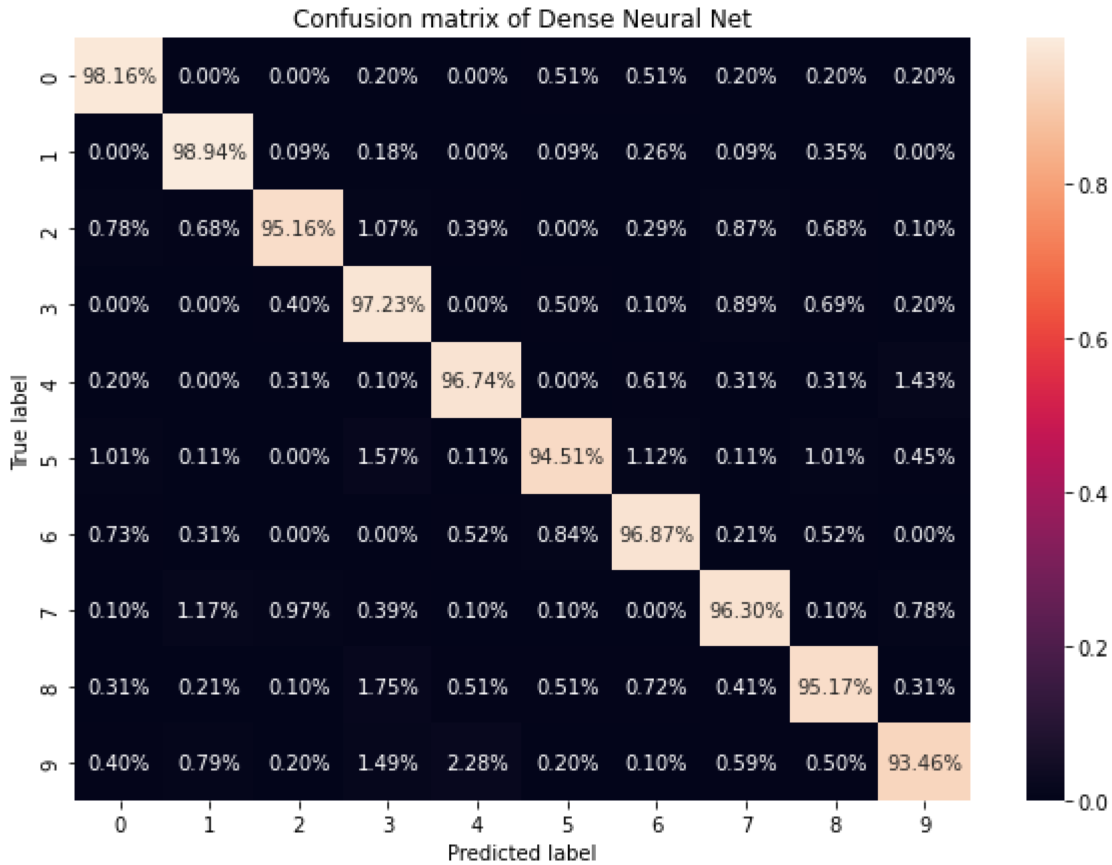
```
#plotting first 25 samples with labels
plot_first25labels(test_x, test_y, test_y_hat)
```




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




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




```
#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
```

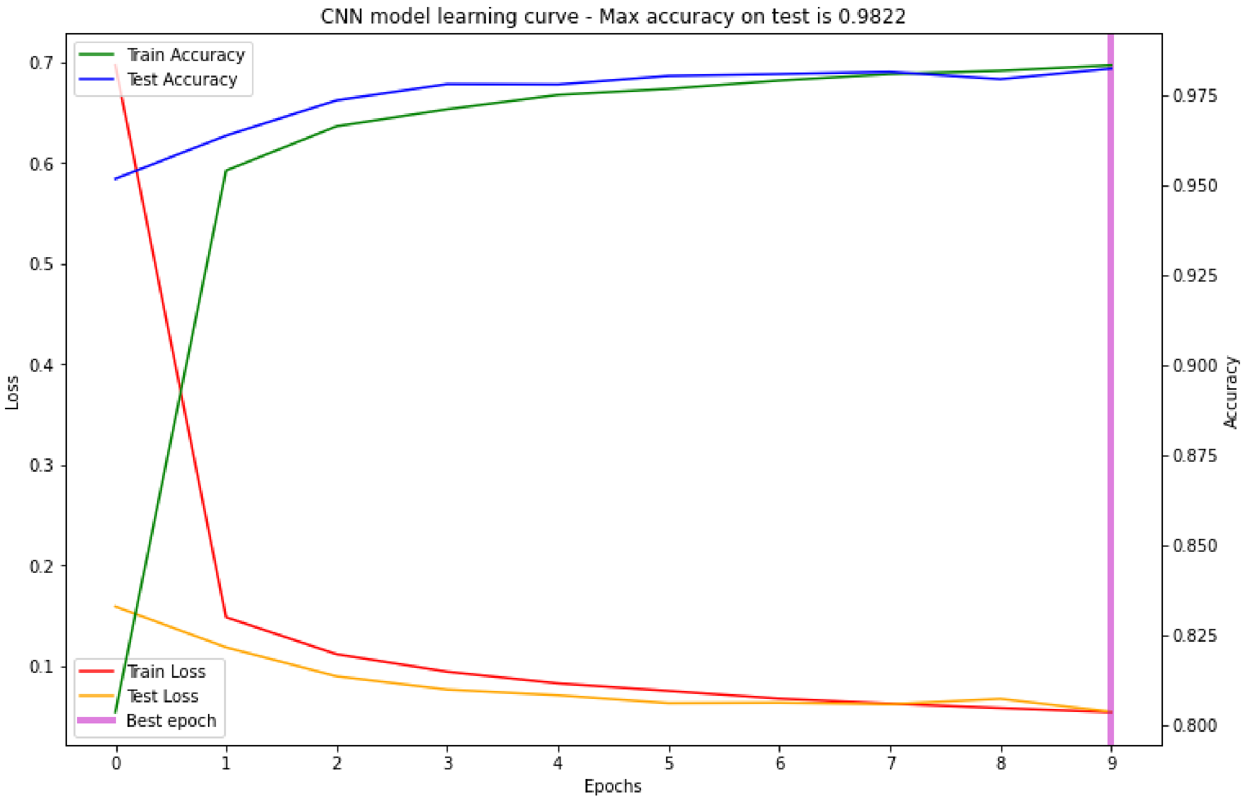
```
!mkdir ConvModelCheckPoints
```

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

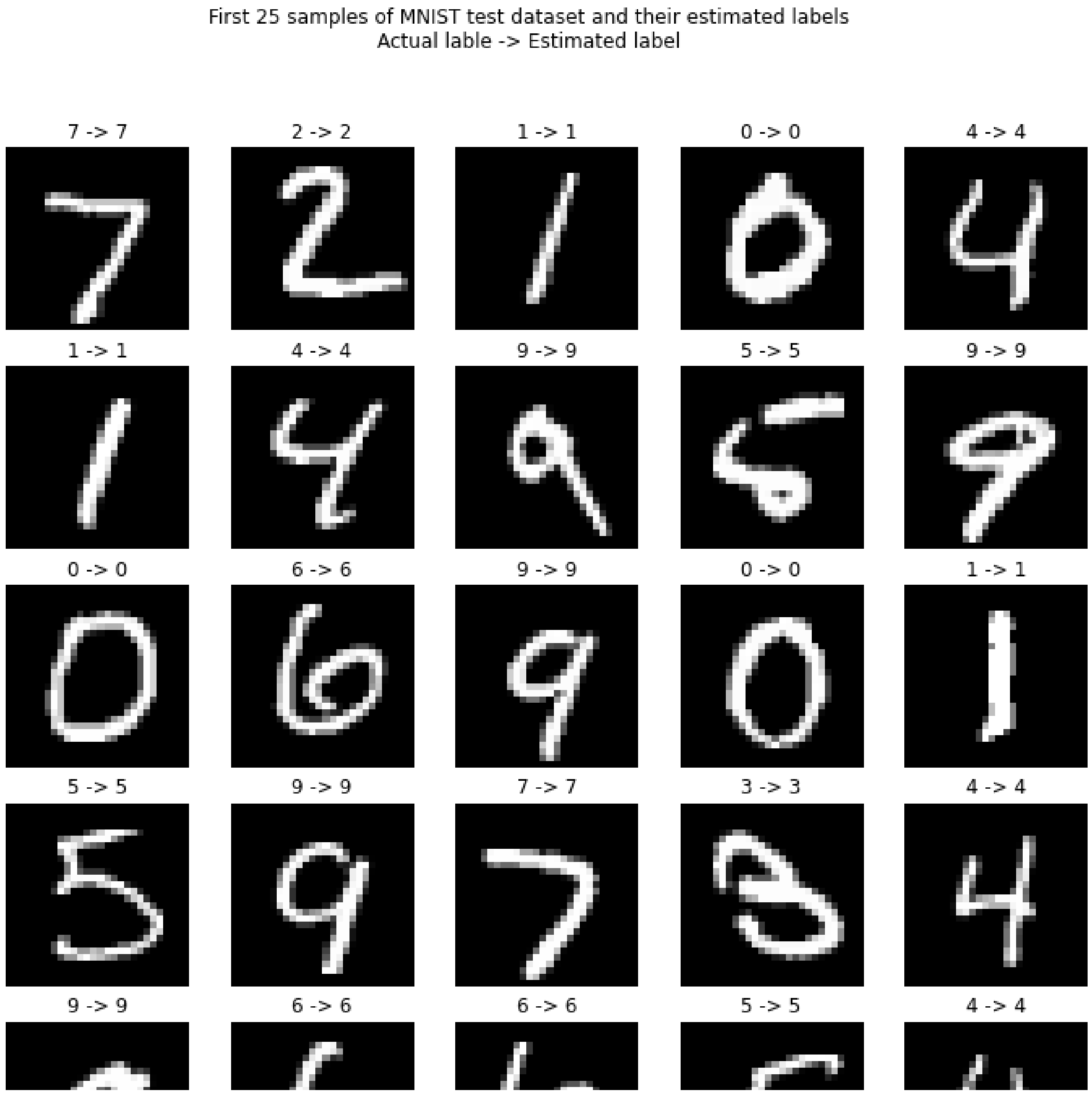
```
#training the model
history = model.fit(train_x, one_hot_train_y,
                    validation_data=(test_x, one_hot_test_y),
                    epochs=Epochs,
                    batch_size=BatchSize,
                    callbacks=[
                        ModelCheckpoint(filepath='ConvModelCheckPoints/best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

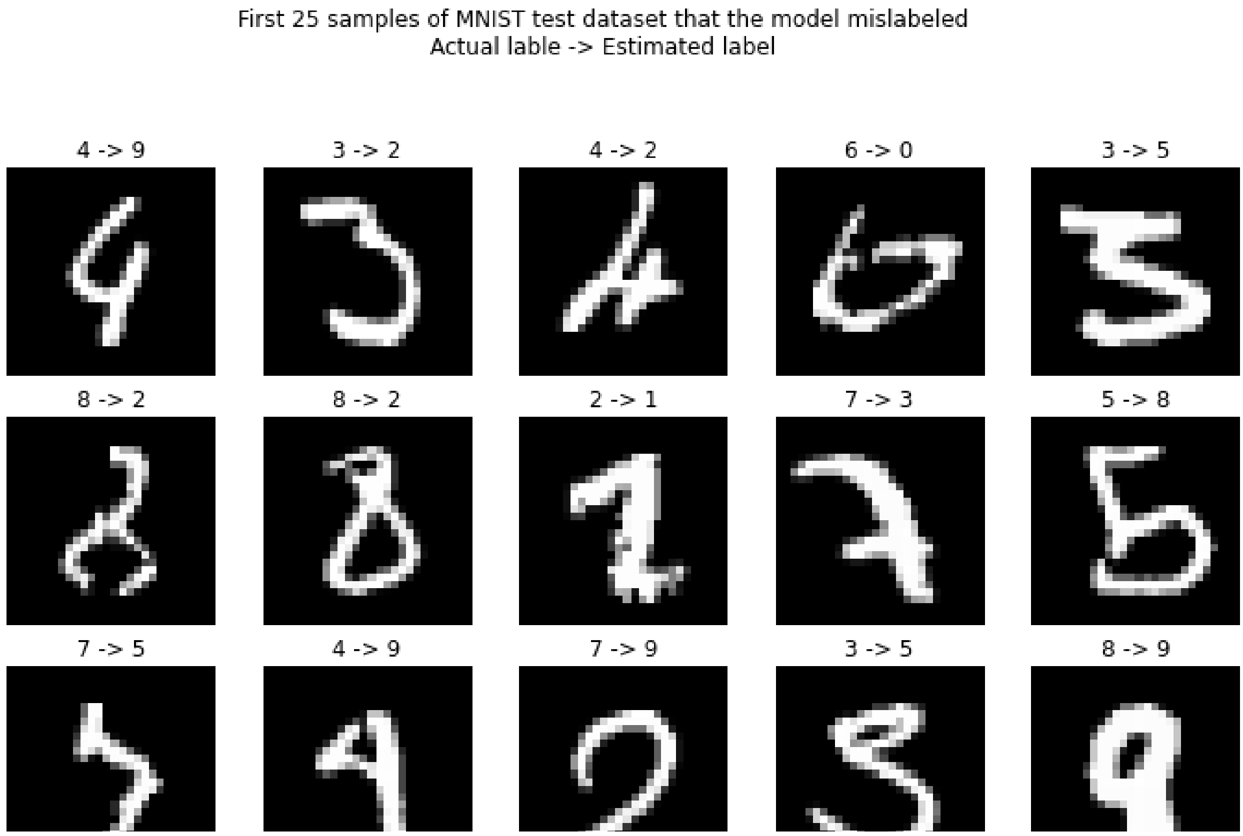
```
#plotting learning curves and labelling them
filename='ConvModelLearningCurve.png'
model_name = 'CNN'
plot_history(history, filename, model_name)
```



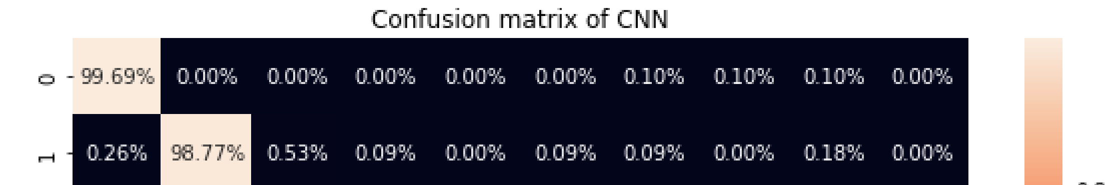

```
#loading best model
model = load_model('ConvModelCheckPoints/best_conv_model.h5')


#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)
```




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




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




```
#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