Name : SHEIK PAREETH

Deep learning ¶

ANN

In [8]:

```
#normal library
import numpy as np
import pandas as pd
```

In [7]:

```
# vizualisation library
import matplotlib.pyplot as plt
import pydot
import seaborn as sns
```

In [6]:

```
#evaluation library
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
```

In [5]:

```
#deep learning libraries
import tensorflow as tf
from tensorflow.keras import layers
import keras
from keras.models import Sequential
from keras.layers.core import Dense,Activation,Dropout
from keras.datasets import mnist
from keras.utils.np_utils import to_categorical
from keras.wrappers.scikit_learn import KerasClassifier
```

In [4]:

```
import tensorflow as tf
```

In [11]:

```
#Digit MNIST dataset
(X_train_digit, y_train_digit), (X_test_digit, y_test_digit) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dat
asets/mnist.npz (https://storage.googleapis.com/tensorflow/tf-keras-dataset
s/mnist.npz)
11493376/11490434 [==============================] - 14s 1us/step
11501568/11490434 [==============================] - 14s 1us/step
```

In [13]:

```
X_train_digit[3]
```

Out[13]:

```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0, 124, 253, 255,  63,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,  96, 244, 251, 253,  62,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0, 127, 251, 251, 253,  62,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,  68, 236, 251, 211,  31,   8,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,  60, 228, 251, 251,  94,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0, 155, 253, 253, 189,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,  20, 253, 251, 235,  66,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         32, 205, 253, 251, 126,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        104, 251, 253, 184,  15,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  80,
        240, 251, 193,  23,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  32, 253,
        253, 253, 159,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 151, 251,
        251, 251,  39,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  48, 221, 251,
        251, 172,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 234, 251, 251,
```

```
        196,  12,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 253, 251, 251,
         89,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0, 159, 255, 253, 253,
         31,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  48, 228, 253, 247, 140,
          8,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  64, 251, 253, 220,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  64, 251, 253, 220,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  24, 193, 253, 220,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0]], dtype=uint8)
```

In [14]:

```
#Names of numbers in the dataset in order
col_names = ['Zero','One','Two','Three','Four','Five','Six','Seven','Eight','Nine']
```

In [ ]:

```
#Visualizing the digits
plt.figure(figsize=(10,10))
for i in range(15):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(X_train_digit[i], cmap='gray')
    plt.xlabel(col_names[y_train_digit[i]])
plt.show()
```

In [15]:

```
X_train_digit.shape
```

Out[15]:

```
(60000, 28, 28)
```

In [16]:

```
X_test_digit.shape
```

Out[16]:

```
(10000, 28, 28)
```

In [17]:

```
#Preprocessing the input-Converting 3d to 2d
X_train_digit = X_train_digit.reshape(60000, 784)
X_test_digit = X_test_digit.reshape(10000, 784)
```

In [18]:

```
X_train_digit[3]
```

Out[18]:

```
array([  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0, 124, 253, 255,  63,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,  96, 244, 251, 253,  62,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0, 127, 251, 251, 253,  62,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,  68, 236, 251, 211,  31,   8,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,  60, 228, 251, 251,  94,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0, 155, 253, 253, 189,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  20, 253, 251,
       235,  66,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  32, 205,
       253, 251, 126,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
       104, 251, 253, 184,  15,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,  80, 240, 251, 193,  23,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,  32, 253, 253, 253, 159,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0, 151, 251, 251, 251,  39,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,  48, 221, 251, 251, 172,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0, 234, 251, 251, 196,  12,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0, 253, 251, 251,  89,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 159, 255, 253,
       253,  31,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  48, 228,
       253, 247, 140,   8,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
        64, 251, 253, 220,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,  64, 251, 253, 220,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,  24, 193, 253, 220,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
        0,    0,    0,    0], dtype=uint8)
```

In [19]:

```
#Encoding Digit MNIST Labels
y_train_digit = to_categorical(y_train_digit, num_classes=10)
y_test_digit = to_categorical(y_test_digit, num_classes=10)
```

In [24]:

```
y_train_digit[2]
```

Out[24]:

```
array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0.], dtype=float32)
```

In [25]:

```
#Creating base neural network
model = keras.Sequential([
    layers.Dense(256, activation='relu', input_shape=(784,)),
    #layers.Dropout(0.3),
    #layers.BatchNormalization(),
    layers.Dense(64, activation='relu'),
    #layers.Dropout(0.3),
    #layers.BatchNormalization(),
    layers.Dense(64, activation='relu'),
    #layers.Dropout(0.3),
    #layers.BatchNormalization(),
    layers.Dense(10,activation='sigmoid'),
])
```

In [26]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               200960

 dense_1 (Dense)             (None, 64)                16448

 dense_2 (Dense)             (None, 64)                4160

 dense_3 (Dense)             (None, 10)                650

=================================================================
Total params: 222,218
Trainable params: 222,218
Non-trainable params: 0
_________________________________________________________________
```

In [27]:

```
#Compiling the model
model.compile(loss="categorical_crossentropy",optimizer="adam",metrics = ['accuracy'])
```

In [28]:

```
history=model.fit(X_train_digit, y_train_digit, batch_size=100, epochs=10,validation_data=(
```

```
Epoch 1/10
600/600 [==============================] - 14s 16ms/step - loss: 1.3529 - ac
curacy: 0.8536 - val_loss: 0.3168 - val_accuracy: 0.9181
Epoch 2/10
600/600 [==============================] - 8s 14ms/step - loss: 0.2337 - acc
uracy: 0.9393 - val_loss: 0.2119 - val_accuracy: 0.9425
Epoch 3/10
600/600 [==============================] - 7s 12ms/step - loss: 0.1565 - acc
uracy: 0.9564 - val_loss: 0.2014 - val_accuracy: 0.9478
Epoch 4/10
600/600 [==============================] - 7s 12ms/step - loss: 0.1164 - acc
uracy: 0.9659 - val_loss: 0.2004 - val_accuracy: 0.9480
Epoch 5/10
600/600 [==============================] - 7s 11ms/step - loss: 0.1066 - acc
uracy: 0.9695 - val_loss: 0.1503 - val_accuracy: 0.9595
Epoch 6/10
600/600 [==============================] - 7s 12ms/step - loss: 0.0833 - acc
uracy: 0.9754 - val_loss: 0.1707 - val_accuracy: 0.9600
Epoch 7/10
600/600 [==============================] - 8s 13ms/step - loss: 0.0868 - acc
uracy: 0.9754 - val_loss: 0.1424 - val_accuracy: 0.9666
Epoch 8/10
600/600 [==============================] - 8s 13ms/step - loss: 0.0760 - acc
uracy: 0.9776 - val_loss: 0.1425 - val_accuracy: 0.9676
Epoch 9/10
600/600 [==============================] - 7s 11ms/step - loss: 0.0779 - acc
uracy: 0.9779 - val_loss: 0.1260 - val_accuracy: 0.9698
Epoch 10/10
600/600 [==============================] - 11s 18ms/step - loss: 0.0638 - ac
curacy: 0.9812 - val_loss: 0.1411 - val_accuracy: 0.9664
```

In [29]:

```
test_loss_digit, test_acc_digit = model.evaluate(X_test_digit, y_test_digit)
```

```
313/313 [==============================] - 7s 8ms/step - loss: 0.1411 - accu
racy: 0.9664
```

In [30]:

```
print("Digit MNIST Test accuracy:", round(test_acc_digit,4))
```

```
Digit MNIST Test accuracy: 0.9664
```

In [31]:

```
#Predicting the labels-DIGIT
y_predict = model.predict(X_test_digit)
```

In [32]:

```
y_predict[0]
```

Out[32]:

```
array([4.8274117e-11, 8.5088217e-01, 6.5431267e-02, 3.1192201e-01,
       5.3272545e-03, 4.4774115e-03, 1.1904573e-11, 9.9999940e-01,
       1.4269352e-03, 8.6200720e-01], dtype=float32)
```

In [33]:

```
# Here we get the index of maximum value in the encoded vector
y_predicts=np.argmax(y_predict, axis=1)
y_test_digit_eval=np.argmax(y_test_digit, axis=1)
```

In [35]:

```
y_predicts[0]
```

Out[35]:

```
7
```

In [36]:

```
y_test_digit_eval
```

Out[36]:

```
array([7, 2, 1, ..., 4, 5, 6], dtype=int64)
```

In [37]:

```
y_pre=pd.DataFrame(y_predicts)
```

In [38]:

```
y_pre
```

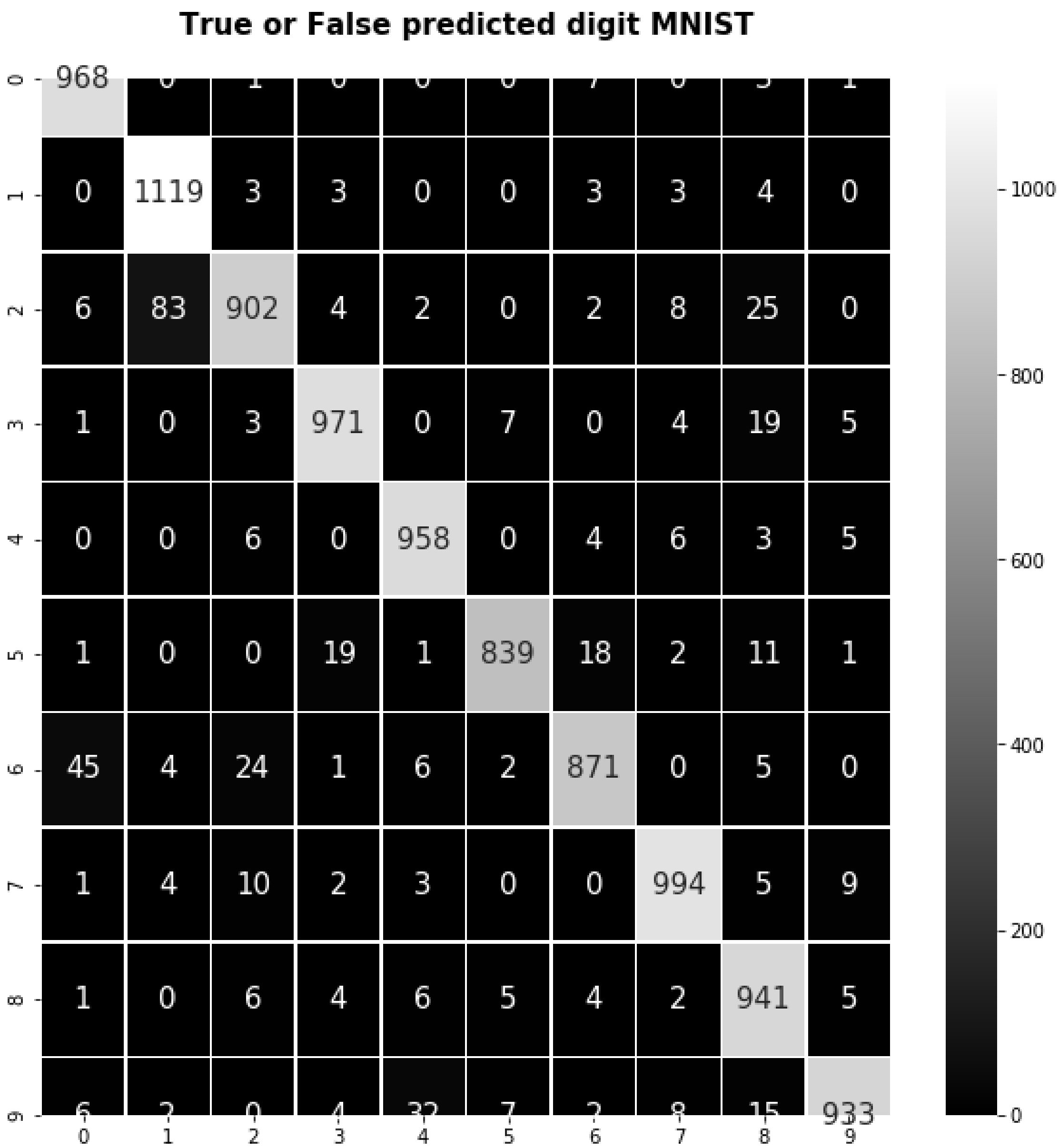
Out[38]:

	0
0	7
1	2
2	1
3	0
4	4
...	...
9995	2
9996	3
9997	4
9998	5
9999	6

10000 rows × 1 columns

In [39]:

```
#Confusion matrix for Digit MNIST
con_mat=confusion_matrix(y_test_digit_eval,y_predicts)
plt.style.use('seaborn-deep')
plt.figure(figsize=(10,10))
sns.heatmap(con_mat,annot=True,annot_kws={'size': 15},linewidths=0.5,fmt="d",cmap="gray")
plt.title('True or False predicted digit MNIST\n',fontweight='bold',fontsize=15)
plt.show()
```



In [40]:

```
#classification report
from sklearn.metrics import classification_report
print(classification_report(y_test_digit_eval,y_predicts))
```

```
              precision    recall  f1-score   support

           0       0.94      0.99      0.96       980
           1       0.92      0.99      0.95      1135
           2       0.94      0.87      0.91      1032
           3       0.96      0.96      0.96      1010
           4       0.95      0.98      0.96       982
           5       0.98      0.94      0.96       892
           6       0.96      0.91      0.93       958
           7       0.97      0.97      0.97      1028
           8       0.91      0.97      0.94       974
           9       0.97      0.92      0.95      1009

    accuracy                           0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000
```

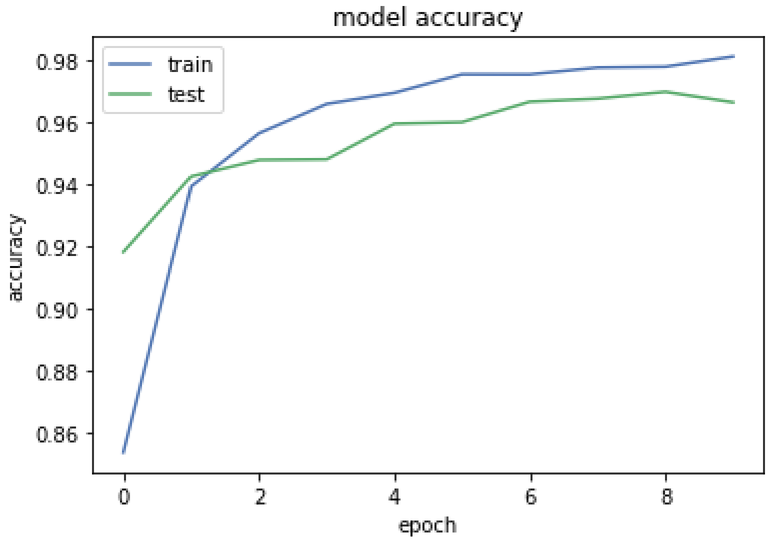
In [41]:

```
print(history.history.keys())
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

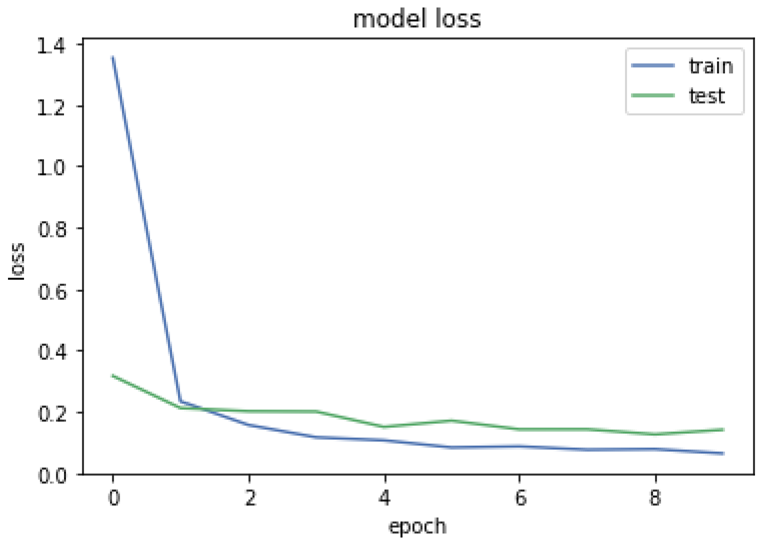
In [42]:

```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='best')
plt.show()
```



In [43]:

```
#Loss graph
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='best')
plt.show()
```



In [44]:

```
#tf.expand_dims(X_test_digit[0])
y_predict_single = model.predict(X_test_digit[[2]])
y_predicts_single=np.argmax(y_predict_single, axis=1) # Here we get the index of maximum va
y_test_digit_eval=np.argmax(y_test_digit, axis=1)
```

In [45]:

```
y_predicts_single[0]
```

Out[45]:

1

In [46]:

```
#Names of numbers in the dataset in order
col_names = ['Zero','One','Two','Three','Four','Five','Six','Seven','Eight','Nine']
```

In [47]:

```
#Visualizing the digits
#plt.figure(figsize=(10,10))
plt.imshow(X_test_digit[2].reshape(28,28), cmap='gray')
plt.xlabel("Actual:{},Pred:{}".format(col_names[np.argmax(y_test_digit[2])],col_names[y_pre
plt.show()
```