In [3]:

```
# Basic Libraries
import numpy as np
import pandas as pd

# Visualization libraries
import matplotlib.pyplot as plt
import pydot
import seaborn as sns

#Evaluation library
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

# 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 [1]:

```
#pip install keras
```

```
Requirement already satisfied: keras in c:\anaconda3\envs\ml\lib\site-pac
kages (2.7.0)
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:

```
#pip install tensorflow
```

In [5]:

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

In [ ]:

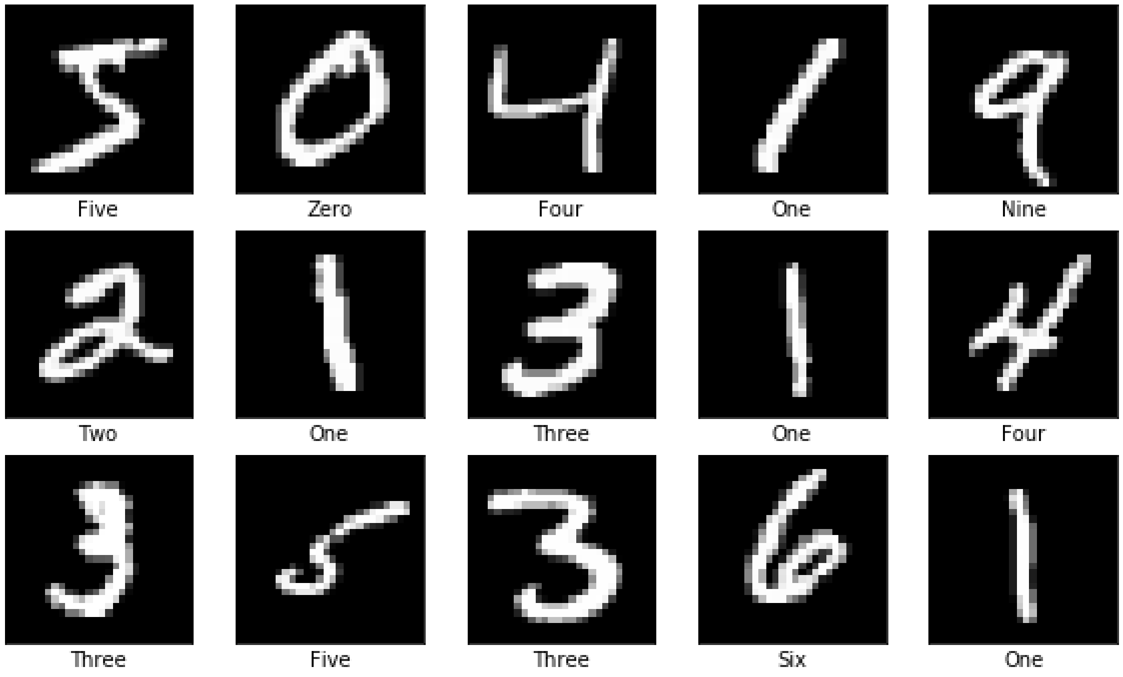
```

```

In [6]:

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

#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 [7]:

```
X_train_digit = X_train_digit.reshape(60000, 784)
X_test_digit = X_test_digit.reshape(10000, 784)
```

In [8]:

```
#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 [9]:

```
#Creating base neural network
model = keras.Sequential([
    layers.Dense(128, activation='relu', input_shape=(784,)),
    layers.Dropout(0.3),
    layers.BatchNormalization(),
    layers.Dense(24, activation='relu'),
    layers.Dropout(0.3),
    layers.BatchNormalization(),
    layers.Dense(24, activation='relu'),
    layers.Dropout(0.3),
    layers.BatchNormalization(),
    layers.Dense(10,activation='sigmoid'),

])
```

In [10]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               100480

 dropout (Dropout)           (None, 128)               0

 batch_normalization (BatchN  (None, 128)              512
 ormalization)
 dense_1 (Dense)             (None, 24)                3096      
                                                                 
 dropout_1 (Dropout)         (None, 24)                0         
                                                                 
 batch_normalization_1 (Batc  (None, 24)               96        
 hNormalization)                                                 
                                                                 
 dense_2 (Dense)             (None, 24)                600       
                                                                 
 dropout_2 (Dropout)         (None, 24)                0         
                                                                 
 batch_normalization_2 (Batc  (None, 24)               96        
 hNormalization)                                                 
                                                                 
 dense_3 (Dense)             (None, 10)                250       
                                                                 
=================================================================
Total params: 105,130
Trainable params: 104,778
Non-trainable params: 352
_________________________________________________________________
```

In [ ]:

```
#https://keras.io/api/losses/probabilistic_losses/#sparsecategoricalcros
```

In [11]:

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

In [12]:

```
history=model.fit(X_train_digit, y_train_digit, batch_size=100, epochs=1
```

```
Epoch 1/30
WARNING:tensorflow:AutoGraph could not transform <function Model.make_tra
in_function.<locals>.train_function at 0x000001965FA5E558> and will run i
t as-is.
Please report this to the TensorFlow team. When filing the bug, set the v
erbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the
full output.
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with @tf.autograph.experim
ental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_train_functio
n.<locals>.train_function at 0x000001965FA5E558> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the v
erbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the
full output.
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with @tf.autograph.experim
ental.do_not_convert
597/600 [============================>.] - ETA: 0s - loss: 0.9807 - accur
acy: 0.6969WARNING:tensorflow:AutoGraph could not transform <function Mod
el.make_test_function.<locals>.test_function at 0x0000019661FD4C18> and w
ill run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the v
erbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the
full output.
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with @tf.autograph.experim
```

```
ental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_test_functio
n.<locals>.test_function at 0x0000019661FD4C18> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the v
erbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the
full output.
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with @tf.autograph.experim
ental.do_not_convert
600/600 [==============================] - 6s 6ms/step - loss: 0.9788 - a
ccuracy: 0.6975 - val_loss: 0.2401 - val_accuracy: 0.9343
Epoch 2/30
600/600 [==============================] - 4s 6ms/step - loss: 0.4859 - a
ccuracy: 0.8613 - val_loss: 0.1675 - val_accuracy: 0.9548
Epoch 3/30
600/600 [==============================] - 3s 6ms/step - loss: 0.3922 - a
ccuracy: 0.8914 - val_loss: 0.1469 - val_accuracy: 0.9575
Epoch 4/30
600/600 [==============================] - 3s 5ms/step - loss: 0.3501 - a
ccuracy: 0.9054 - val_loss: 0.1403 - val_accuracy: 0.9604
Epoch 5/30
600/600 [==============================] - 3s 5ms/step - loss: 0.3202 - a
ccuracy: 0.9129 - val_loss: 0.1243 - val_accuracy: 0.9639
Epoch 6/30
600/600 [==============================] - 4s 6ms/step - loss: 0.2973 - a
ccuracy: 0.9191 - val_loss: 0.1156 - val_accuracy: 0.9677
Epoch 7/30
600/600 [==============================] - 3s 5ms/step - loss: 0.2821 - a
ccuracy: 0.9230 - val_loss: 0.1087 - val_accuracy: 0.9693
Epoch 8/30
600/600 [==============================] - 3s 4ms/step - loss: 0.2672 - a
ccuracy: 0.9285 - val_loss: 0.1053 - val_accuracy: 0.9705
Epoch 9/30
600/600 [==============================] - 3s 5ms/step - loss: 0.2569 - a
ccuracy: 0.9297 - val_loss: 0.1038 - val_accuracy: 0.9731
Epoch 10/30
600/600 [==============================] - 3s 5ms/step - loss: 0.2512 - a
ccuracy: 0.9318 - val_loss: 0.1072 - val_accuracy: 0.9709
Epoch 11/30
600/600 [==============================] - 3s 5ms/step - loss: 0.2475 - a
ccuracy: 0.9332 - val_loss: 0.1019 - val_accuracy: 0.9726
Epoch 12/30
600/600 [==============================] - 3s 5ms/step - loss: 0.2409 - a
ccuracy: 0.9354 - val_loss: 0.1023 - val_accuracy: 0.9733
Epoch 13/30
600/600 [==============================] - 3s 5ms/step - loss: 0.2323 - a
ccuracy: 0.9377 - val_loss: 0.1014 - val_accuracy: 0.9738
Epoch 14/30
600/600 [==============================] - 3s 5ms/step - loss: 0.2290 - a
ccuracy: 0.9375 - val_loss: 0.0966 - val_accuracy: 0.9733
Epoch 15/30
600/600 [==============================] - 3s 5ms/step - loss: 0.2225 - a
ccuracy: 0.9391 - val_loss: 0.1014 - val_accuracy: 0.9735
Epoch 16/30
600/600 [==============================] - 3s 5ms/step - loss: 0.2211 - a
ccuracy: 0.9388 - val_loss: 0.1021 - val_accuracy: 0.9737
Epoch 17/30
600/600 [==============================] - 3s 5ms/step - loss: 0.2146 - a
ccuracy: 0.9417 - val_loss: 0.1071 - val_accuracy: 0.9745
Epoch 18/30
600/600 [==============================] - 3s 5ms/step - loss: 0.2142 - a
ccuracy: 0.9414 - val_loss: 0.0939 - val_accuracy: 0.9735
Epoch 19/30
```

```
600/600 [==============================] - 2s 4ms/step - loss: 0.2119 - a
ccuracy: 0.9425 - val_loss: 0.0871 - val_accuracy: 0.9762
Epoch 20/30
600/600 [==============================] - 2s 3ms/step - loss: 0.2115 - a
ccuracy: 0.9421 - val_loss: 0.0940 - val_accuracy: 0.9760
Epoch 21/30
600/600 [==============================] - 2s 4ms/step - loss: 0.2087 - a
ccuracy: 0.9422 - val_loss: 0.0925 - val_accuracy: 0.9773
Epoch 22/30
600/600 [==============================] - 2s 3ms/step - loss: 0.2084 - a
ccuracy: 0.9426 - val_loss: 0.0912 - val_accuracy: 0.9763
Epoch 23/30
600/600 [==============================] - 2s 4ms/step - loss: 0.2015 - a
ccuracy: 0.9452 - val_loss: 0.1089 - val_accuracy: 0.9768
Epoch 24/30
600/600 [==============================] - 2s 3ms/step - loss: 0.1959 - a
ccuracy: 0.9453 - val_loss: 0.0875 - val_accuracy: 0.9768
Epoch 25/30
600/600 [==============================] - 2s 4ms/step - loss: 0.1978 - a
ccuracy: 0.9456 - val_loss: 0.0903 - val_accuracy: 0.9767
Epoch 26/30
600/600 [==============================] - 2s 3ms/step - loss: 0.1981 - a
ccuracy: 0.9452 - val_loss: 0.0881 - val_accuracy: 0.9759
Epoch 27/30
600/600 [==============================] - 2s 3ms/step - loss: 0.1928 - a
ccuracy: 0.9466 - val_loss: 0.0977 - val_accuracy: 0.9753
Epoch 28/30
600/600 [==============================] - 2s 3ms/step - loss: 0.1940 - a
ccuracy: 0.9474 - val_loss: 0.1366 - val_accuracy: 0.9755
Epoch 29/30
600/600 [==============================] - 2s 4ms/step - loss: 0.1926 - a
ccuracy: 0.9466 - val_loss: 0.1038 - val_accuracy: 0.9777
Epoch 30/30
600/600 [==============================] - 2s 4ms/step - loss: 0.1837 - a
ccuracy: 0.9497 - val_loss: 0.0859 - val_accuracy: 0.9776
```

In [14]:

```
test_loss_digit, test_acc_digit = model.evaluate(X_test_digit, y_test_di
```

```
313/313 [==============================] - 2s 5ms/step - loss: 0.0859 - a
ccuracy: 0.9776
```

In [15]:

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

```
Digit MNIST Test accuracy: 0.9776
```

In [16]:

```
#Predicting the labels-DIGIT
y_predict = model.predict(X_test_digit)
y_predict=np.argmax(y_predict, axis=1) # Here we get the index of maximu
y_test_digit_eval=np.argmax(y_test_digit, axis=1)
```

```
WARNING:tensorflow:AutoGraph could not transform <function Model.make_pre
dict_function.<locals>.predict_function at 0x00000196621E34C8> and will r
un it as-is.
Please report this to the TensorFlow team. When filing the bug, set the v
erbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the
full output.
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with @tf.autograph.experim
ental.do_not_convert
WARNING: AutoGraph could not transform <function Model.make_predict_funct
ion.<locals>.predict_function at 0x00000196621E34C8> and will run it as-i
```

```
s.
Please report this to the TensorFlow team. When filing the bug, set the v
erbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the
full output.
Cause: 'arguments' object has no attribute 'posonlyargs'
To silence this warning, decorate the function with @tf.autograph.experim
ental.do_not_convert
```

In [32]:

```
X_test_digit
```

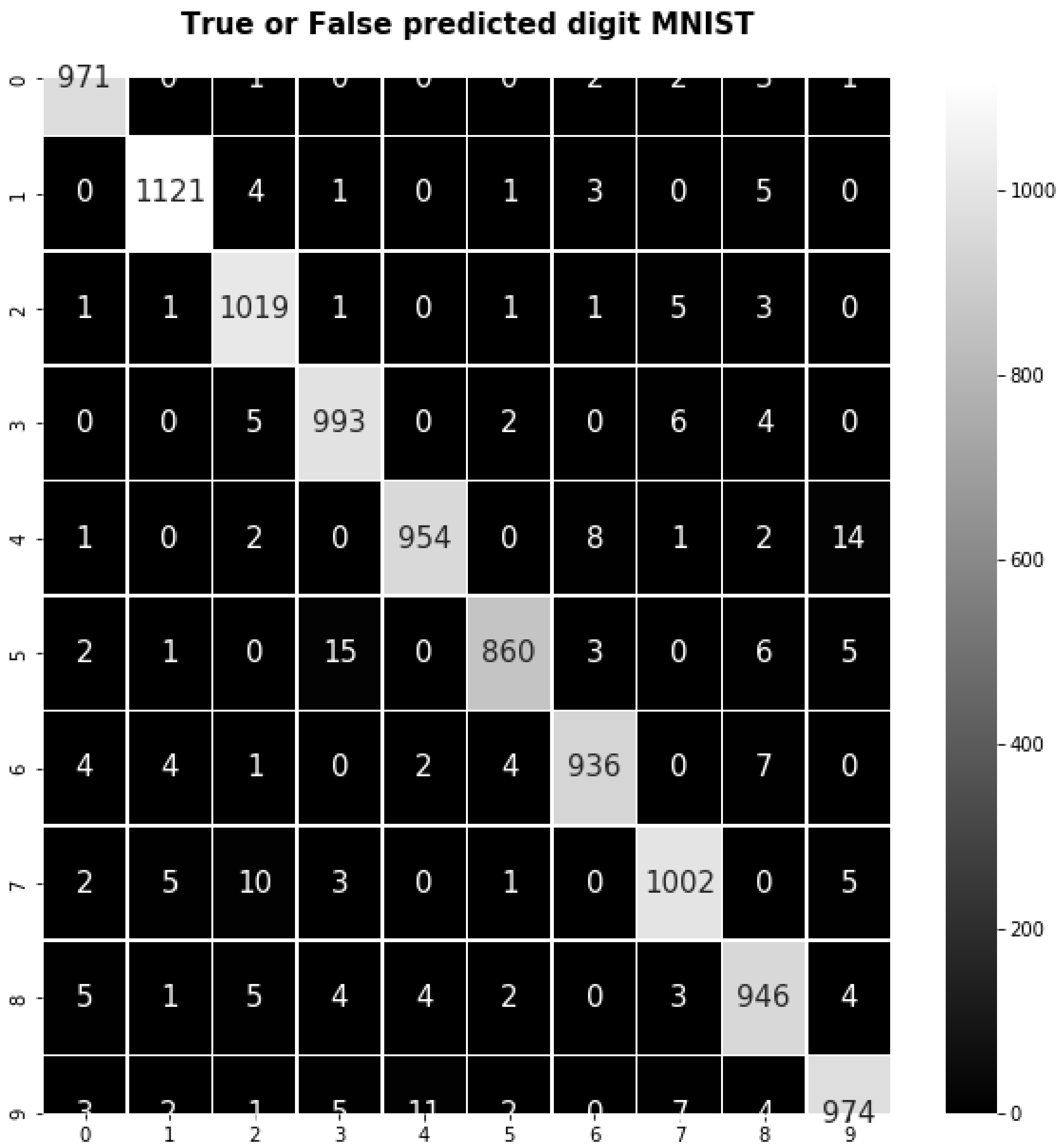
```
C:\Anaconda3\envs\ML\lib\site-packages\ipykernel_launcher.py:1: FutureWar
ning: Using a non-tuple sequence for multidimensional indexing is depreca
ted; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future this will
be interpreted as an array index, `arr[np.array(seq)]`, which will result
either in an error or a different result.
  """Entry point for launching an IPython kernel.
```

Out[32]:

```
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,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  84, 185,
        159, 151,  60,  36,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0, 222, 254, 254, 254, 254, 241, 198, 198, 198, 198,
        198, 198, 198, 198, 170,  52,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,  67, 114,  72, 114, 163, 227,
        254, 225, 254, 254, 254, 250, 229, 254, 254, 140,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,  17,  66,  14,  67,  67,  67,  59,  21, 236,
        254, 106,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,  83, 253, 209,  18,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,  22, 233, 255,  83,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0, 129, 254, 238,
         44,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         59, 249, 254,  62,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0, 133, 254, 187,   5,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   9, 205, 248,  58,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 126,
        254, 182,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,  75, 251, 240,  57,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,  19, 221, 254, 166,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   3, 203, 254, 219,
         35,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         38, 254, 254,  77,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,  31, 224, 254, 115,   1,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0, 133, 254, 254,  52,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  61, 242,
        254, 254,  52,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0, 121, 254, 254, 219,  40,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0, 121, 254, 207,  18,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   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 [17]:

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



In [18]:

```
from sklearn.metrics import classification_report

print(classification_report(y_test_digit_eval,y_predict))
```

```
              precision    recall  f1-score   support

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

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

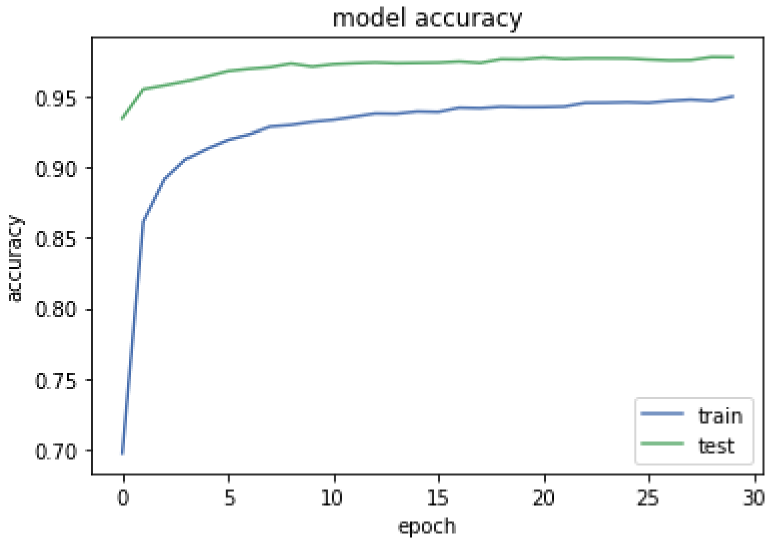
In [19]:

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

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

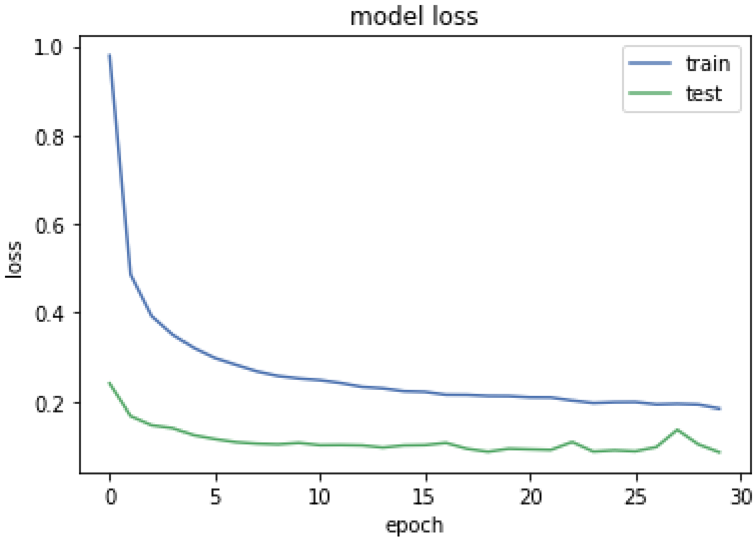
In [20]:

```
# 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 [21]:

```
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 [56]:

```
#tf.expand_dims(X_test_digit[0])
y_predict = model.predict(X_test_digit[[0]])
y_predict=np.argmax(y_predict, axis=1) # Here we get the index of maximu
y_test_digit_eval=np.argmax(y_test_digit, axis=1)
```

In [57]:

```
y_predict[0]
```

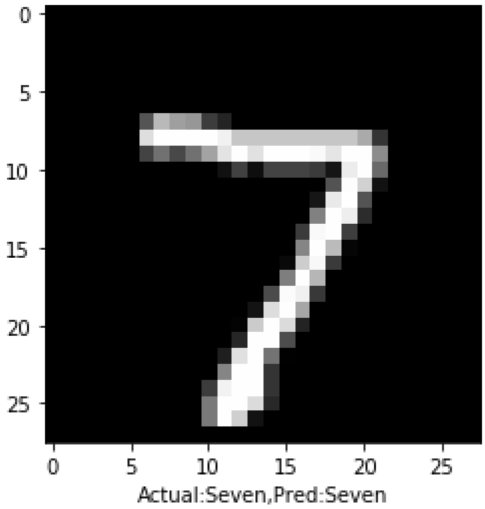
Out[57]:

```
7
```

In [58]:

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

#Visualizing the digits
#plt.figure(figsize=(10,10))
plt.imshow(X_test_digit[0].reshape(28,28), cmap='gray')
plt.xlabel("Actual:{},Pred:{}".format(col_names[np.argmax(y_test_digit[0
plt.show()
```



In [52]:

```
y_test_digit[8]
```

Out[52]:

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

In [ ]: