EXPERIMENT-2

Build an Artificial Neural Network to implement Multi-Class Classification task using the Back-propagation algorithm and test the same using appropriate data sets

Database

- The data that will be incorporated is the **MNIST database** (Modified National Institute of Standards and Technology database) which contains 60,000 images for training and 10,000 test images.
- The dataset consists of small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9
- The MNIST dataset is conveniently bundled within Keras, and we can easily analyze some of its features in Python.

In [2]:
```
pip install matplotlib
```

```
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dis
t-packages (3.8.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.
10/dist-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/d
ist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python
3.10/dist-packages (from matplotlib) (4.47.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python
3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: numpy<2,>=1.21 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib) (1.22.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib) (23.1)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.10/dist
-packages (from matplotlib) (9.5.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.
10/dist-packages (from matplotlib) (3.1.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/pyth
on3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
WARNING: Running pip as the 'root' user can result in broken permissions a
nd conflicting behaviour with the system package manager. It is recommende
d to use a virtual environment instead: https://pip.pypa.io/warnings/venv
Note: you may need to restart the kernel to use updated packages.
```

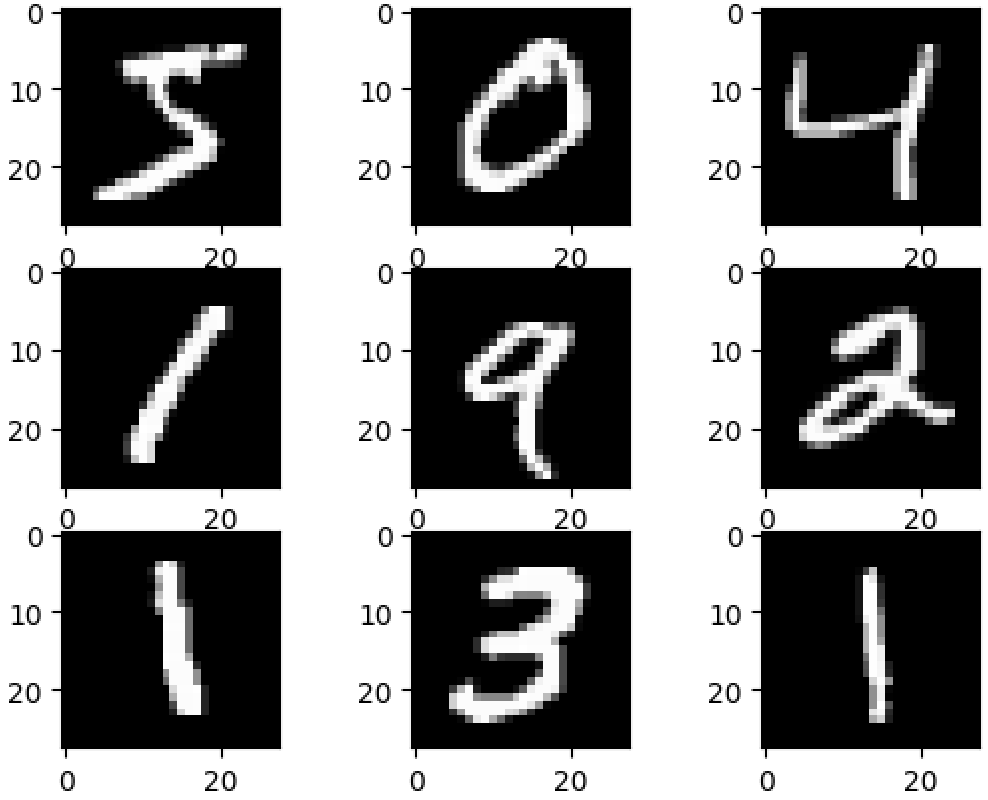
In [21]:
```
from tensorflow import keras
from keras.datasets import mnist     # MNIST dataset is included in Keras
(X_train, y_train), (X_test, y_test) = mnist.load_data()

print("X_train shape", X_train.shape)
print("y_train shape", y_train.shape)
print("X_test shape", X_test.shape)
print("y_test shape", y_test.shape)
```

```
X_train shape (60000, 28, 28)
y_train shape (60000,)
X_test shape (10000, 28, 28)
y_test shape (10000,)
```

In [22]:

```
# Plot first few images
import matplotlib.pyplot as plt
for i in range(9):
        # define subplot
        plt.subplot(3,3,i+1) # 3 rows, 3 col, pos
        # plot raw pixel data
        plt.imshow(X_train[i], cmap='gray')
# show the figure
plt.show()
```



In [23]:

```
X_train[i].shape
```

Out[23]:

```
(28, 28)
```

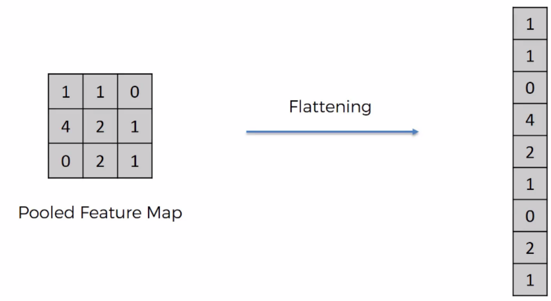
In [24]:

```
# Each pixel is an 8-bit integer from 0-255 (0 is full black, 255 is full
# single-channel pixel or monochrome image
X_train[i][10:20,10:20]
```

```
Out[24]: array([[  0,   0,  20, 254, 254, 108,   0,   0,   0,   0],
               [  0,   0,  16, 239, 254, 143,   0,   0,   0,   0],
               [  0,   0,   0, 178, 254, 143,   0,   0,   0,   0],
               [  0,   0,   0, 178, 254, 143,   0,   0,   0,   0],
               [  0,   0,   0, 178, 254, 162,   0,   0,   0,   0],
               [  0,   0,   0, 178, 254, 240,   0,   0,   0,   0],
               [  0,   0,   0, 113, 254, 240,   0,   0,   0,   0],
               [  0,   0,   0,  83, 254, 245,  31,   0,   0,   0],
               [  0,   0,   0,  79, 254, 246,  38,   0,   0,   0],
               [  0,   0,   0,   0, 214, 254, 150,   0,   0,   0]], dtype=uint8)
```

Formatting the input data

- Reshape (or flatten) the 28x28 image into a 784-length vector.



- Input values [0-255] are Normalized in the range [0-1]

A `Min-Max Scaling` is typically done via the following equation:

$$X_{norm} = \frac{X_{i} - X_{min}}{X_{max} - X_{min}}$$

$X_i$ is the $i^{th}$ sample of dataset.

```
In [25]: # reshape 28 x 28 matrices into 784-length vectors
         X_train = X_train.reshape(60000, 784)
         X_test = X_test.reshape(10000, 784)

         # normalize each value for each pixel for the entire vector for each inpu
         # change integers to 32-bit floating point numbers
         X_train = X_train.astype('float32')
         X_test = X_test.astype('float32')
         # normalize by dividing by largest pixel value
         X_train /= 255
         X_test /= 255

         print("Training matrix shape", X_train.shape)
         print("Testing matrix shape", X_test.shape)
```

```
Training matrix shape (60000, 784)
Testing matrix shape (10000, 784)
```

DNN for Multi-class classification using Keras library

Build the model

```
In [26]: # Sequential keras model with Dense layes (DIY)
```

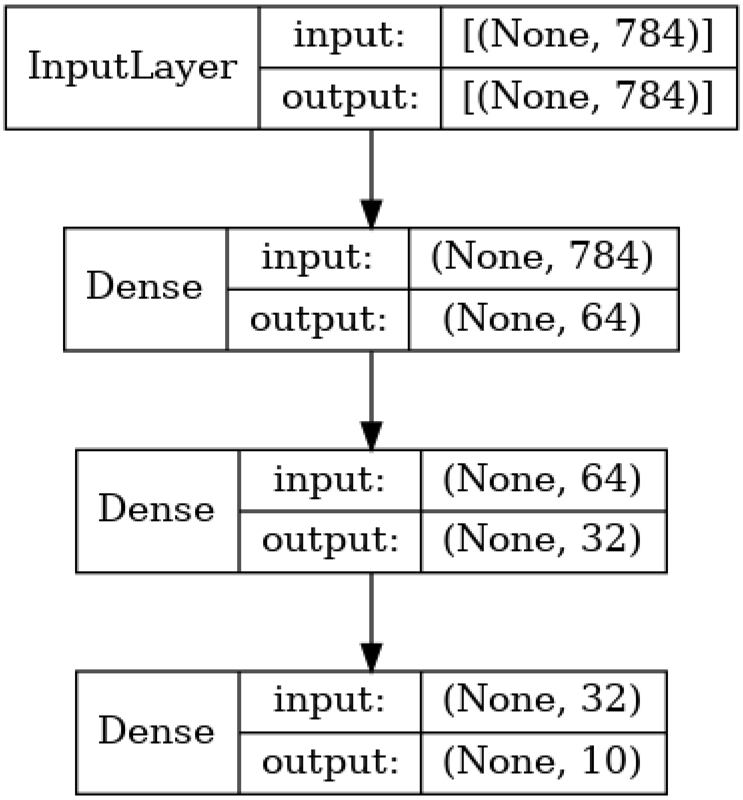
```
from keras.models import Sequential  # Model type to be used
from keras.layers.core import Dense # Types of layers to be used in our m

mdl = Sequential()
mdl.add(Dense(64, input_dim = 28 * 28, activation= 'relu'))
mdl.add(Dense(32, activation = 'relu'))
mdl.add(Dense(10, activation = 'softmax'))
mdl.compile(loss= 'categorical_crossentropy', optimizer = 'adam', metrics
```

In [27]:

```
# Visualize the model
from keras.utils.vis_utils import plot_model
plot_model(mdl, show_shapes=True, show_layer_names=False)
```

Out[27]:



In [28]:

```
# Display model summary
mdl.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_15 (Dense)            (None, 64)                50240     
                                                                 
 dense_16 (Dense)            (None, 32)                2080      
                                                                 
 dense_17 (Dense)            (None, 10)                330       
                                                                 
=================================================================
Total params: 52,650
Trainable params: 52,650
Non-trainable params: 0
_________________________________________________________________
```

In [29]:

```
#understand model summary
784*64 + 64
```

```
Out[29]: 50240
```

```
In [30]: 64*32 + 32
```

```
Out[30]: 2080
```

```
In [31]: 32*10 + 10
```

```
Out[31]: 330
```

Convert labels to "one-hot" vectors using the to_categorical function

```
0 -> [1, 0, 0, 0, 0, 0, 0, 0, 0]
1 -> [0, 1, 0, 0, 0, 0, 0, 0, 0]
2 -> [0, 0, 1, 0, 0, 0, 0, 0, 0]
etc.
```

```
In [32]: from tensorflow.keras.utils import to_categorical
         y_train1 = to_categorical(y_train)
         y_test1 = to_categorical(y_test)
         print(y_test[6])
         print(y_test1[6,:])
```

```
4
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

Train the model

- If unspecified, by default batch_size=32
- 60,000/64 = 938 minibatches
- Reference: https://keras.io/api/models/model_training_apis/

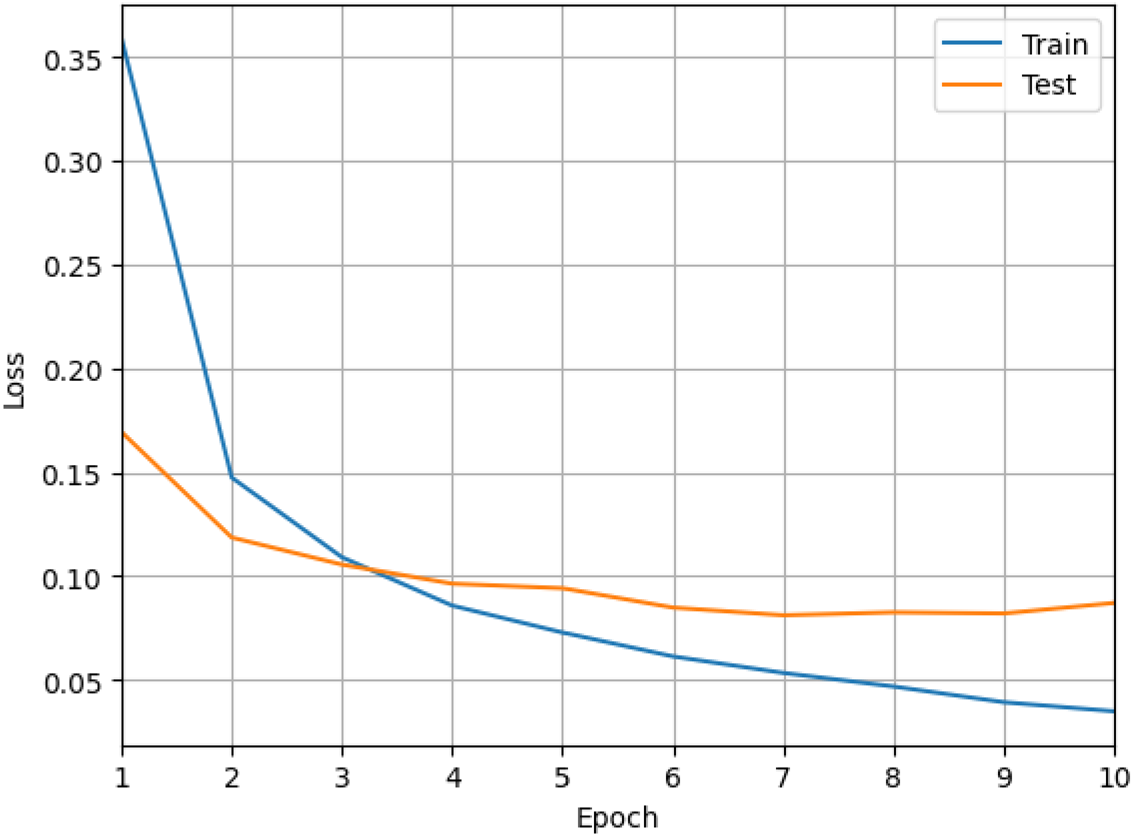
```
In [33]: # Train the model
         epochs=10
         batch = 64
         history = mdl.fit(X_train, y_train1,epochs=epochs, batch_size=batch,verbo
Epoch 1/10
938/938 [==============================] - 2s 1ms/step - loss: 0.3589 - ac
curacy: 0.8974 - val_loss: 0.1695 - val_accuracy: 0.9487
Epoch 2/10
938/938 [==============================] - 1s 1ms/step - loss: 0.1476 - ac
curacy: 0.9570 - val_loss: 0.1187 - val_accuracy: 0.9639
Epoch 3/10
938/938 [==============================] - 1s 1ms/step - loss: 0.1092 - ac
curacy: 0.9676 - val_loss: 0.1057 - val_accuracy: 0.9664
Epoch 4/10
938/938 [==============================] - 1s 1ms/step - loss: 0.0859 - ac
curacy: 0.9744 - val_loss: 0.0965 - val_accuracy: 0.9701
Epoch 5/10
938/938 [==============================] - 1s 1ms/step - loss: 0.0729 - ac
curacy: 0.9783 - val_loss: 0.0943 - val_accuracy: 0.9705
Epoch 6/10
938/938 [==============================] - 1s 1ms/step - loss: 0.0614 - ac
curacy: 0.9808 - val_loss: 0.0849 - val_accuracy: 0.9735
Epoch 7/10
938/938 [==============================] - 1s 1ms/step - loss: 0.0535 - ac
curacy: 0.9831 - val_loss: 0.0812 - val_accuracy: 0.9756
Epoch 8/10
938/938 [==============================] - 1s 1ms/step - loss: 0.0469 - ac
curacy: 0.9853 - val_loss: 0.0827 - val_accuracy: 0.9749
Epoch 9/10
938/938 [==============================] - 1s 1ms/step - loss: 0.0394 - ac
curacy: 0.9878 - val_loss: 0.0821 - val_accuracy: 0.9759
Epoch 10/10
938/938 [==============================] - 1s 1ms/step - loss: 0.0350 - ac
curacy: 0.9892 - val_loss: 0.0872 - val_accuracy: 0.9755
```

Evaluate Model

Plot Learning graphs

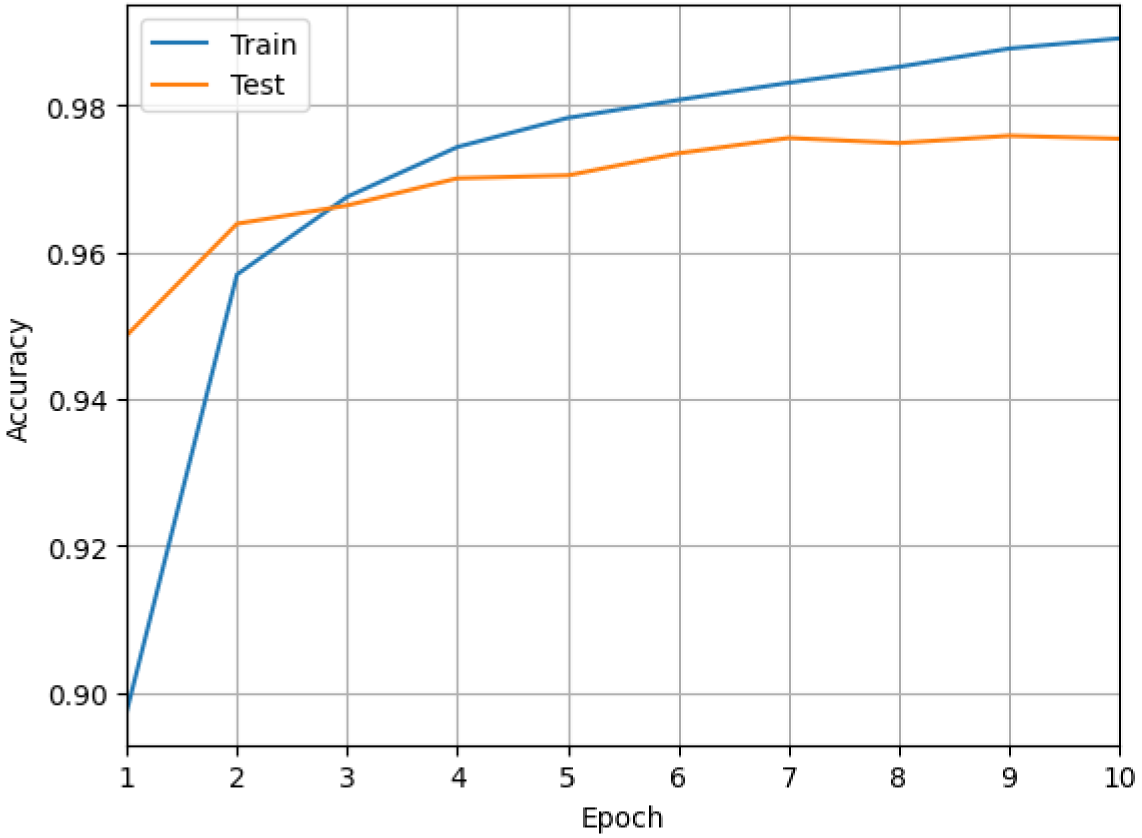
In [34]:

```python
epochRange = range(1,epochs+1);
plt.plot(epochRange,history.history['loss'])
plt.plot(epochRange,history.history['val_loss'])
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid()
plt.xlim((1,epochs))
plt.legend(['Train','Test'])
plt.show()
```



In [35]:

```
plt.plot(epochRange,history.history['accuracy'])
plt.plot(epochRange,history.history['val_accuracy'])
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.grid()
plt.xlim((1,epochs))
plt.legend(['Train','Test'])
plt.show()
```



Performance metrics

In [36]:

```
import numpy as np
yhat_test_mdl_prob = mdl.predict(X_test);
yhat_test_mdl = np.argmax(yhat_test_mdl_prob,axis=-1)
print(yhat_test_mdl_prob[0])
print(yhat_test_mdl[0:10])
print(y_test[0:10])
```

```
313/313 [==============================] - 0s 529us/step
[4.6729565e-07 7.4642479e-08 1.1356531e-05 6.4800479e-03 2.9261302e-09
 1.0041108e-06 2.7862160e-10 9.9341923e-01 1.1755909e-05 7.5990327e-05]
[7 2 1 0 4 1 4 9 5 9]
[7 2 1 0 4 1 4 9 5 9]
```

In [37]:

```
from sklearn.metrics import accuracy_score
print('Accuracy:')
print(float(accuracy_score(y_test, yhat_test_mdl))*100,'%')
```

```
Accuracy:
97.55 %
```

In [38]:

```
from sklearn.metrics import confusion_matrix
print('Confusion Matrix:')
print(confusion_matrix(y_test, yhat_test_mdl))
```

```
Confusion Matrix:
[[ 969    1    3    1    0    1    2    0    1    2]
 [   0 1126    2    1    0    1    2    1    2    0]
 [   4    5  988   12    1    0    3    6   13    0]
 [   1    1    2  995    0    2    0    4    3    2]
 [   1    1    2    1  953    1    0    1    1   21]
 [   2    1    0   18    1  850    7    2    7    4]
 [   4    3    0    1    1    2  941    0    5    1]
 [   1   12    8    1    0    0    0  994    2   10]
 [   3    1    2    5    2    4    1    3  949    4]
 [   2    4    0    3    5    3    0    2    0  990]]
```

Conclusion : Built an Artificial Neural Network to implement Multi-Class Classification task using the Back-propagation algorithm and tested the datasets with an accuracy of 97.55%