In [ ]:
```
import numpy as np

import pandas as pd

from numpy import unique, argmax

from tensorflow.keras.datasets.mnist import load_data

from tensorflow.keras import Sequential

from tensorflow.keras.layers import Conv2D

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Flatten

from tensorflow.keras.layers import Dropout

from tensorflow.keras.utils import plot_model

import matplotlib.pyplot as plt

from tensorflow.keras.datasets import mnist
```

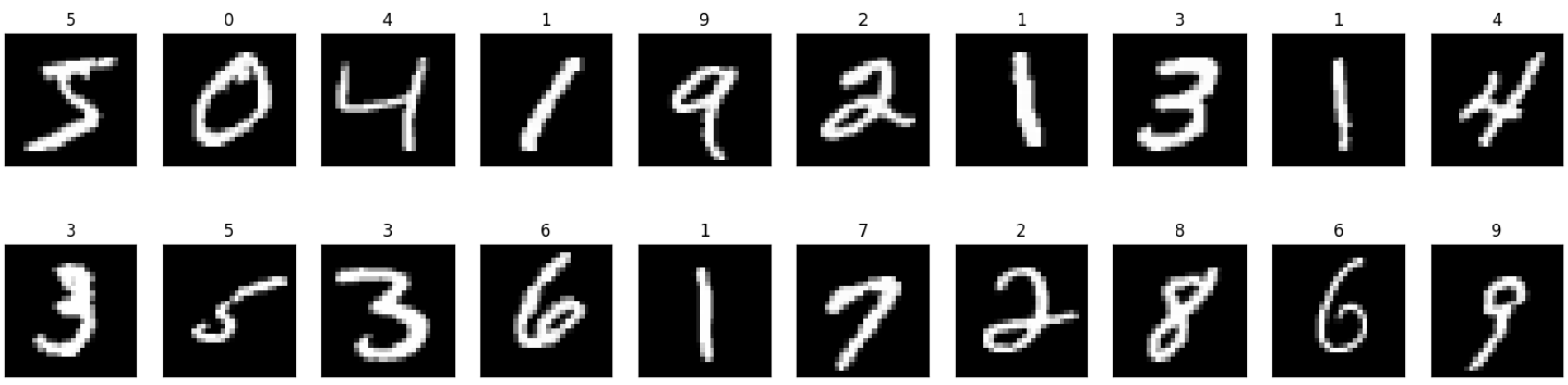
In [ ]:
```
(train_x, train_y), (test_x, test_y) = mnist.load_data()
```

In [ ]:
```
#printing the shapes

print(train_x.shape, train_y.shape)

print(test_x.shape , test_y.shape)
```

```
(60000, 28, 28) (60000,)
(10000, 28, 28) (10000,)
```

In [ ]:
```
#normalizing the pixel values of images

train_x = train_x.astype('float32')/255.0

test_x = test_x.astype('float32')/255.0
```

In [ ]:
```
#plotting images of dataset

fig = plt.figure(figsize = (20,5))

for i in range(20):

    ax= fig.add_subplot(2, 10, i+1, xticks=[], yticks=[])

    ax.imshow(np.squeeze(train_x[i]), cmap='gray')

    ax.set_title(train_y[i])
```




In [ ]:
```
shape = train_x.shape[1:]

shape
```

Out[ ]:
```
(28, 28)
```

In [ ]:
```
#CNN Model

from tensorflow.keras.layers import MaxPooling2D as MaxPool2D
model = Sequential()

#adding convolutional layer

model.add(Conv2D(32, (3,3), activation='relu', input_shape=(28, 28, 1)))

model.add(MaxPool2D((2,2)))

model.add(Conv2D(48, (3,3), activation='relu'))

model.add(MaxPool2D((2,2)))
```

```
model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(500, activation='relu'))

model.add(Dense(10, activation='softmax'))
```

In [ ]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2  (None, 13, 13, 32)        0
 D)

 conv2d_3 (Conv2D)           (None, 11, 11, 48)        13872

 max_pooling2d_1 (MaxPoolin  (None, 5, 5, 48)          0
 g2D)

 dropout (Dropout)           (None, 5, 5, 48)          0

 flatten (Flatten)           (None, 1200)              0

 dense (Dense)               (None, 500)               600500

 dense_1 (Dense)             (None, 10)                5010

=================================================================
Total params: 619702 (2.36 MB)
Trainable params: 619702 (2.36 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [ ]:
```
#compiling model

model.compile(optimizer='adam', loss = 'sparse_categorical_crossentropy',metrics= ['accuracy'] )

x=model.fit(train_x, train_y, epochs=10, batch_size = 128, verbose= 2 , validation_split = 0.1)
```

```
Epoch 1/10
422/422 - 30s - loss: 0.0231 - accuracy: 0.9924 - val_loss: 0.0286 - val_accuracy: 0.9927 - 30s/epoch - 71ms/step
Epoch 2/10
422/422 - 30s - loss: 0.0228 - accuracy: 0.9923 - val_loss: 0.0240 - val_accuracy: 0.9937 - 30s/epoch - 71ms/step
Epoch 3/10
422/422 - 31s - loss: 0.0194 - accuracy: 0.9936 - val_loss: 0.0250 - val_accuracy: 0.9945 - 31s/epoch - 73ms/step
Epoch 4/10
422/422 - 34s - loss: 0.0178 - accuracy: 0.9937 - val_loss: 0.0246 - val_accuracy: 0.9938 - 34s/epoch - 80ms/step
Epoch 5/10
422/422 - 36s - loss: 0.0177 - accuracy: 0.9937 - val_loss: 0.0281 - val_accuracy: 0.9942 - 36s/epoch - 86ms/step
Epoch 6/10
422/422 - 35s - loss: 0.0159 - accuracy: 0.9944 - val_loss: 0.0274 - val_accuracy: 0.9935 - 35s/epoch - 82ms/step
Epoch 7/10
422/422 - 36s - loss: 0.0146 - accuracy: 0.9949 - val_loss: 0.0250 - val_accuracy: 0.9940 - 36s/epoch - 86ms/step
Epoch 8/10
422/422 - 34s - loss: 0.0148 - accuracy: 0.9950 - val_loss: 0.0254 - val_accuracy: 0.9942 - 34s/epoch - 81ms/step
Epoch 9/10
422/422 - 41s - loss: 0.0140 - accuracy: 0.9951 - val_loss: 0.0266 - val_accuracy: 0.9943 - 41s/epoch - 97ms/step
Epoch 10/10
422/422 - 44s - loss: 0.0136 - accuracy: 0.9954 - val_loss: 0.0275 - val_accuracy: 0.9932 - 44s/epoch - 104ms/step
```

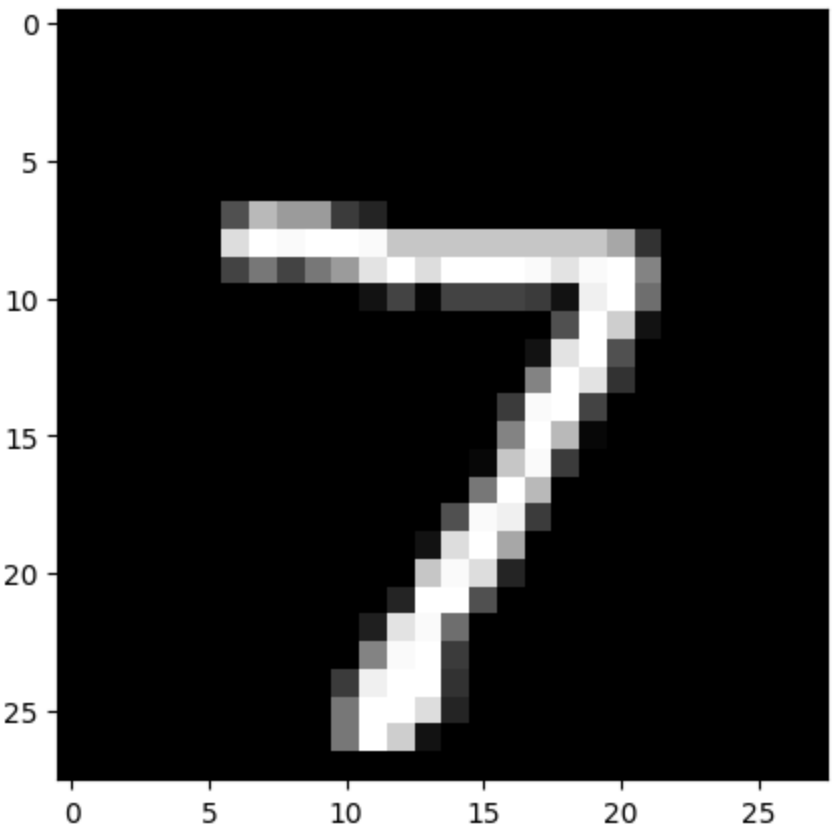
In [ ]:
```
loss, accuracy= model.evaluate(test_x, test_y, verbose = 0)

print(f'Accuracy: {accuracy*100}')
```

```
Accuracy: 99.26000237464905
```

In [ ]:
```
model.save(r'C:\Users\92341\Desktop\DipLab\final_model.h5')
```

In [ ]:
```
# make a prediction for a new image.
from numpy import argmax
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import load_model
import matplotlib.pyplot as plt

# load and prepare the image
def load_image(filename):
 # load the image
 img = load_img(filename, grayscale=True, target_size=(28, 28))
 plt.figure(figsize=(5,5))
 plt.imshow(img,cmap='gray')
 plt.show()
 # convert to array
 img = img_to_array(img)
```

```
 # reshape into a single sample with 1 channel
 img = img.reshape(1, 28, 28, 1)
 # prepare pixel data
 img = img.astype('float32')
 img = img / 255.0
 return img

# load an image and predict the class
def run_example():
 # load the image
 img = load_image(r'C:\Users\92341\Desktop\7.png')

 # load model
 model = load_model(r'C:\Users\92341\Desktop\DipLab\final_model.h5')
 # predict the class
 predict_value = model.predict(img)
 digit = argmax(predict_value)

 print('Predicted',digit)

# entry point, run the example
run_example()
```




```
1/1 [==============================] - 0s 64ms/step
Predicted 7
1/1 [==============================] - 0s 64ms/step
Predicted 7
```