In [1]:
```
'''TASK 03 - NUMBER RECOGNITION'''

# Ignore warnings for cleaner output
import warnings
warnings.filterwarnings('ignore')

# Importing libraries for machine learning and deep learning
from sklearn.preprocessing import MinMaxScaler # For data normaliza
from keras.models import Sequential # For creating a sequential neu
from keras.layers import Dense, Dropout, LSTM, Bidirectional # For
```

In [2]:
```
import tensorflow as tf
```

In [3]:
```
from keras.datasets import mnist
from keras.utils import to_categorical
```

In [4]:
```
# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-ker
as-datasets/mnist.npz (https://storage.googleapis.com/tensorflow/tf-ke
ras-datasets/mnist.npz)
11490434/11490434 [==============================] - 6s 1us/step
```

In [9]:
```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)

```

```
(60000, 28, 28)
(60000, 10)
(10000, 28, 28)
(10000, 10)
```

In [10]:

```
print(x_train)
```

```
[[[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 ...

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]]
```

In [11]:

```
print(x_test)
```

```
[[[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 ...

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]

 [[0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  ...
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]
  [0. 0. 0. ... 0. 0. 0.]]]
```

In [12]:

```
# Reshaping the input Data which is used as a input in CNN in Tense
# CNN takes the input Data in 4D Format with the shape (num_samples
# Here (num_channels) is set to 1 which means input image is Graysc

x_train = x_train.reshape((x_train.shape[0] , x_train.shape[1] , x_
x_test = x_test.reshape((x_test.shape[0] , x_test.shape[1] , x_test
print(x_train.shape)
print(x_test.shape)
print(x_train.dtype)
print(x_test.dtype)
```

```
(60000, 28, 28, 1)
(10000, 28, 28, 1)
float64
float64
```

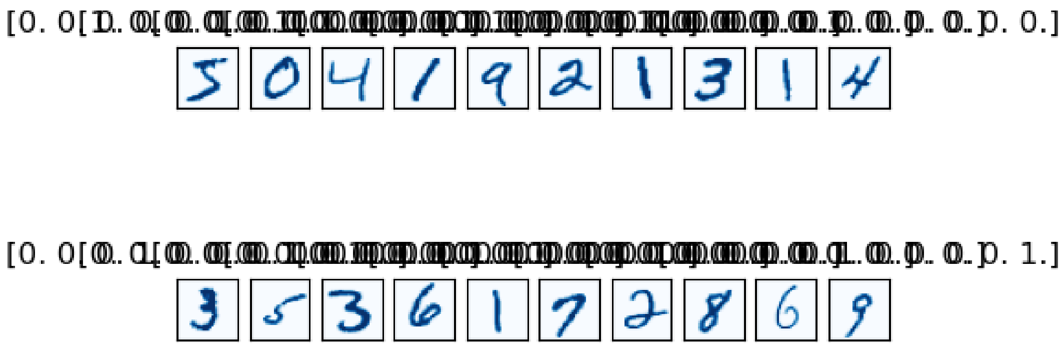
In [13]:

```
# Normalizing Pixel Values

x_train = x_train.astype('float32')/255.0
x_test = x_test.astype('float32')/255.0
print(x_train.dtype)
print(x_test.dtype)

```

```
float32
float32
```

In [14]:

```
fig=plt.figure(figsize=(5,3))
for i in range(20):
    ax =fig.add_subplot(2,10,i+1, xticks=[], yticks=[])
    ax.imshow(np.squeeze(x_train[i]), cmap='Blues')
    ax.set_title(y_train[i])
```



In [15]:

```
# showing shape of single image
img_shape= x_train.shape[1:]
img_shape
```

Out[15]:

```
(28, 28, 1)
```

In [48]:
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28,1)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10)
])
```

In [49]:
```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_5 (Flatten)         (None, 784)               0

 dense_10 (Dense)            (None, 128)               100480

 dropout_5 (Dropout)         (None, 128)               0

 dense_11 (Dense)            (None, 10)                1290

=================================================================
Total params: 101770 (397.54 KB)
Trainable params: 101770 (397.54 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [50]:
```
# Displaying Neural Network Model
from tensorflow.keras.utils import plot_model
plot_model(model, 'model.jpg', show_shapes = True)
```

```
You must install pydot (`pip install pydot`) and install graphviz (see instructions at https://graphviz.gitlab.io/download/) (https://graphviz.gitlab.io/download/)) for plot_model to work.
```

In [ ]:
```

```

In [ ]:
```

```

In [51]:
```
# Applying Softmax() Function to prediction array
# This convert an output vector of real numbers into a probability
tf.nn.softmax(prediction).numpy()
```

Out[51]:
```
array([[0.09981512, 0.09993055, 0.09990993, 0.1002295 , 0.10028599,
        0.09991155, 0.10023038, 0.09992979, 0.09978843, 0.09996876]],
      dtype=float32)
```

In [57]:
```


# Compile the model with this loss
model.compile(optimizer='adam', loss='categorical_crossentropy', me


```

In [58]:
```
model.fit(x_train, y_train, epochs=5,batch_size=32)

```

```
Epoch 1/5
1875/1875 [==============================] - 3s 2ms/step - loss: 9.459
3 - accuracy: 0.1020
Epoch 2/5
1875/1875 [==============================] - 3s 1ms/step - loss: 9.525
3 - accuracy: 0.1026
Epoch 3/5
1875/1875 [==============================] - 3s 1ms/step - loss: 9.627
3 - accuracy: 0.1154
Epoch 4/5
1875/1875 [==============================] - 3s 1ms/step - loss: 9.542
2 - accuracy: 0.1138
Epoch 5/5
1875/1875 [==============================] - 3s 2ms/step - loss: 9.594
6 - accuracy: 0.1132
```

Out[58]:
```
<keras.src.callbacks.History at 0x20897f708b0>
```

In [59]:
```
# Evaluating the Model
model.evaluate(x_test, y_test, verbose=2)
```

```
313/313 - 0s - loss: 11.1021 - accuracy: 0.1437 - 367ms/epoch - 1ms/st
ep
```

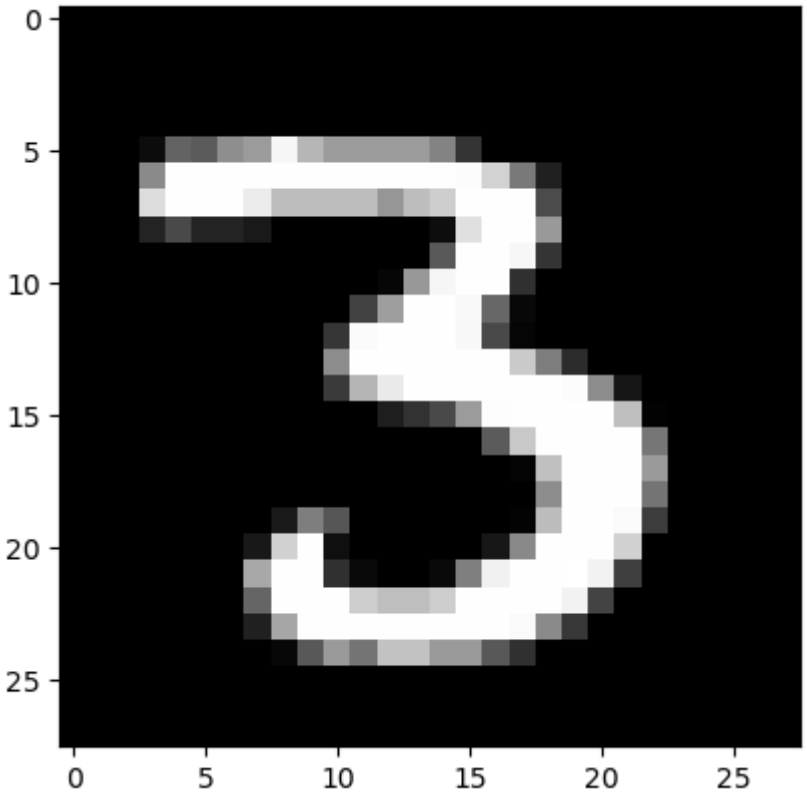
Out[59]:
```
[11.10214900970459, 0.1437000036239624]
```

In [60]:
```
# Creating a new sequential model which includes both previously tr
probability_model = tf.keras.Sequential([ model,tf.keras.layers.Sof
probability_model(x_test[:5])
```

Out[60]:
```
<tf.Tensor: shape=(5, 10), dtype=float32, numpy=
array([[0.09602615, 0.09623323, 0.10268615, 0.09590027, 0.10280494,
        0.10257605, 0.10288402, 0.10250293, 0.10222689, 0.09615933],
       [0.09596132, 0.09618185, 0.10310774, 0.09572108, 0.10281739,
        0.10295112, 0.10280609, 0.10200837, 0.10237346, 0.09607158],
       [0.09649432, 0.09598102, 0.10274434, 0.09603737, 0.10268992,
        0.10249823, 0.10279985, 0.10238555, 0.10209569, 0.09627371],
       [0.09496093, 0.09643601, 0.10280189, 0.0956667 , 0.10293792,
        0.10328193, 0.10299419, 0.10222725, 0.10274   , 0.09595316],
       [0.0960004 , 0.09665856, 0.10272933, 0.096179  , 0.10258953,
        0.10252755, 0.10248248, 0.10221114, 0.10235294, 0.09626911]],
      dtype=float32)>
```

In [61]:

```
# Displaying a Grayscale Image
img = x_train[12]
plt.imshow(np.squeeze(img) ,cmap='gray')
plt.show()
```



In [ ]:

```
1
```

In [ ]:

```
1
```