In [1]:

```
import numpy as np
import pandas as pd
import random
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Conv2D, Dense, MaxPooling2D
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist
```

In [2]:

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

In [3]:

```
print(X_train.shape)
```

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

In [4]:

```
X_train[0].min(), X_train[0].max()
```

Out[4]:

```
(0, 255)
```

In [5]:

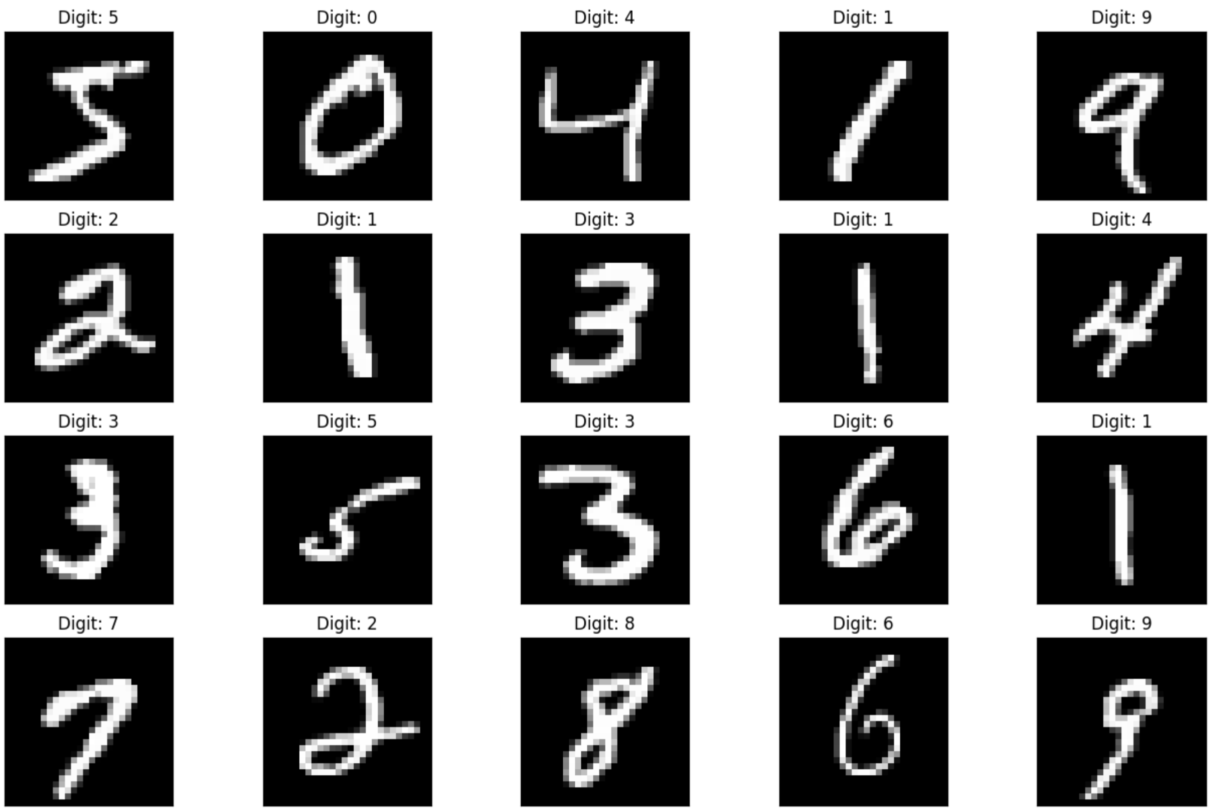
```
X_train = (X_train - 0.0) / (255.0 - 0.0)
X_test = (X_test - 0.0) / (255.0 - 0.0)
X_train[0].min(), X_train[0].max()
```

Out[5]:

```
(0.0, 1.0)
```

In [6]:

```
def plot_digit(image, digit, plt, i):
    plt.subplot(4, 5, i + 1)
    plt.imshow(image, cmap=plt.get_cmap('gray'))
    plt.title(f"Digit: {digit}")
    plt.xticks([])
    plt.yticks([])
plt.figure(figsize=(16, 10))
for i in range(20):
    plot_digit(X_train[i], y_train[i], plt, i)
plt.show()
```



In [7]:

```
X_train = X_train.reshape((X_train.shape + (1,)))
X_test = X_test.reshape((X_test.shape + (1,)))
```

In [8]:

```
y_train[0:20]
```

Out[8]:

```
array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4, 3, 5, 3, 6, 1, 7, 2, 8, 6, 9],
      dtype=uint8)
```

In [9]:

```
model = Sequential([
    Conv2D(32, (3, 3), activation="relu", input_shape=(28, 28, 1)),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(100, activation="relu"),
    Dense(10, activation="softmax")
])
```

In [10]:

```
optimizer = SGD(learning_rate=0.01, momentum=0.9)
model.compile(
    optimizer=optimizer,
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

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

 flatten (Flatten)           (None, 5408)              0

 dense (Dense)               (None, 100)               540900

 dense_1 (Dense)             (None, 10)                1010

=================================================================
Total params: 542230 (2.07 MB)
Trainable params: 542230 (2.07 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [11]:

```
model.fit(X_train, y_train, epochs=10, batch_size=32)
```

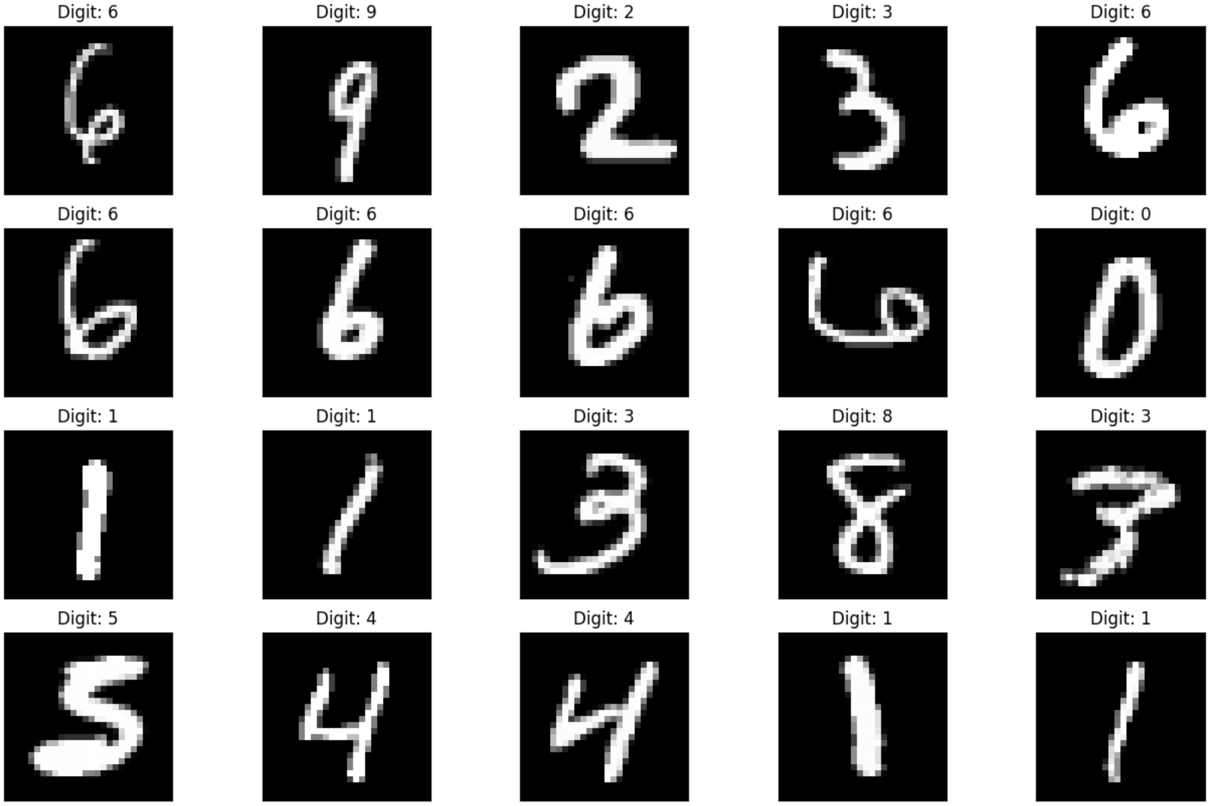
```
Epoch 1/10
1875/1875 [==============================] - 47s 24ms/step - loss: 0.2354 - accurac
y: 0.9287
Epoch 2/10
1875/1875 [==============================] - 39s 21ms/step - loss: 0.0808 - accurac
y: 0.9758
Epoch 3/10
1875/1875 [==============================] - 39s 21ms/step - loss: 0.0534 - accurac
y: 0.9836
Epoch 4/10
1875/1875 [==============================] - 40s 21ms/step - loss: 0.0378 - accurac
y: 0.9884
Epoch 5/10
1875/1875 [==============================] - 39s 21ms/step - loss: 0.0282 - accurac
y: 0.9913
Epoch 6/10
1875/1875 [==============================] - 42s 22ms/step - loss: 0.0216 - accurac
y: 0.9932
Epoch 7/10
1875/1875 [==============================] - 40s 21ms/step - loss: 0.0152 - accurac
y: 0.9954
Epoch 8/10
1875/1875 [==============================] - 38s 20ms/step - loss: 0.0113 - accurac
y: 0.9969
Epoch 9/10
1875/1875 [==============================] - 38s 20ms/step - loss: 0.0082 - accurac
y: 0.9978
Epoch 10/10
1875/1875 [==============================] - 40s 21ms/step - loss: 0.0065 - accurac
y: 0.9983
```

Out[11]:

```
<keras.src.callbacks.History at 0x1f0c6e017e0>
```

In [ ]:
```
plt.figure(figsize=(16, 10))
for i in range(20):
    image = random.choice(X_test).squeeze()
    digit = np.argmax(model.predict(image.reshape((1, 28, 28, 1)))[0], axis=-1)
    plot_digit(image, digit, plt, i)
plt.show()
```

```
1/1 [==============================] - 0s 438ms/step
1/1 [==============================] - 0s 129ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 87ms/step
1/1 [==============================] - 0s 131ms/step
1/1 [==============================] - 0s 105ms/step
1/1 [==============================] - 0s 145ms/step
1/1 [==============================] - 0s 87ms/step
1/1 [==============================] - 0s 102ms/step
1/1 [==============================] - 0s 180ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 140ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 129ms/step
1/1 [==============================] - 0s 83ms/step
1/1 [==============================] - 0s 122ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 71ms/step
1/1 [==============================] - 0s 81ms/step
1/1 [==============================] - 0s 111ms/step
```

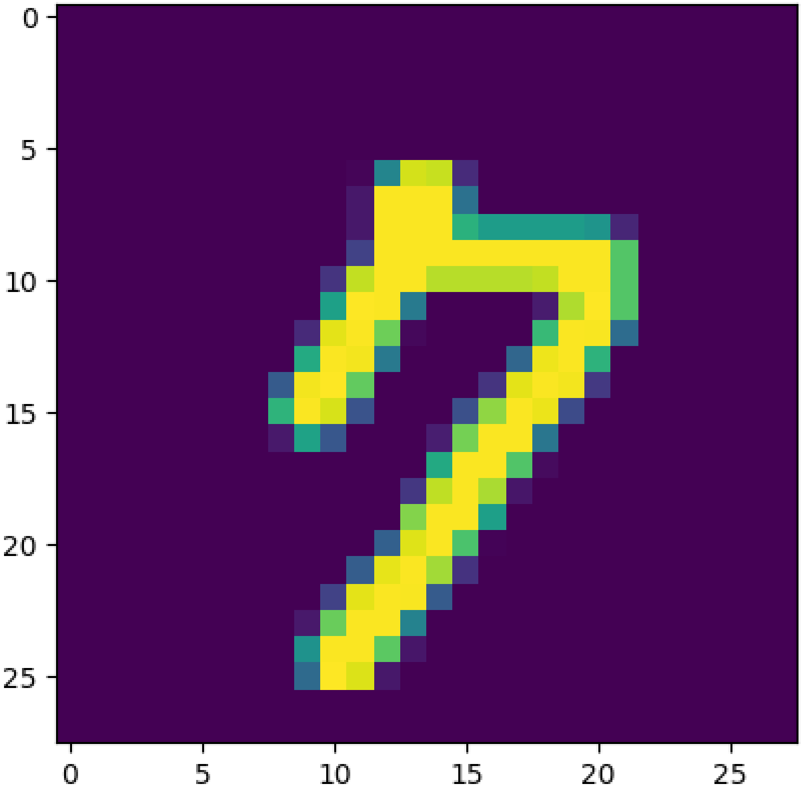


In [ ]:
```
predictions = np.argmax(model.predict(X_test), axis=-1)
accuracy_score(y_test, predictions)
```

```
313/313 [==============================] - 4s 14ms/step
```

Out[ ]:
```
0.9857
```

In [ ]:
```
n=random.randint(0,9999)
plt.imshow(X_test[n])
plt.show()
```



In [ ]:
```
predicted_value=model.predict(X_test)
print("Handwritten number in the image is= %d" %np.argmax(predicted_value[n]))
```

```
313/313 [==============================] - 4s 12ms/step
Handwritten number in the image is= 7
```

In [ ]:
```
score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0]) #Test loss: 0.0296396646054
print('Test accuracy:', score[1])
```

```
Test loss: 0.048509154468774796
Test accuracy: 0.9857000112533569
```

In [ ]:
```
#The implemented CNN model is giving Loss=0.04624301567673683  and
#accuracy: 0.9872000217437744 for test mnist dataset
```