Name: Madhuri Wavhal

Roll No:88

Batch:BE IT B4

In [31]:
```
#importing required libraries
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 [32]:
```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Type *Markdown* and LaTeX: α^2

In [33]:
```
print(X_train.shape)
```

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

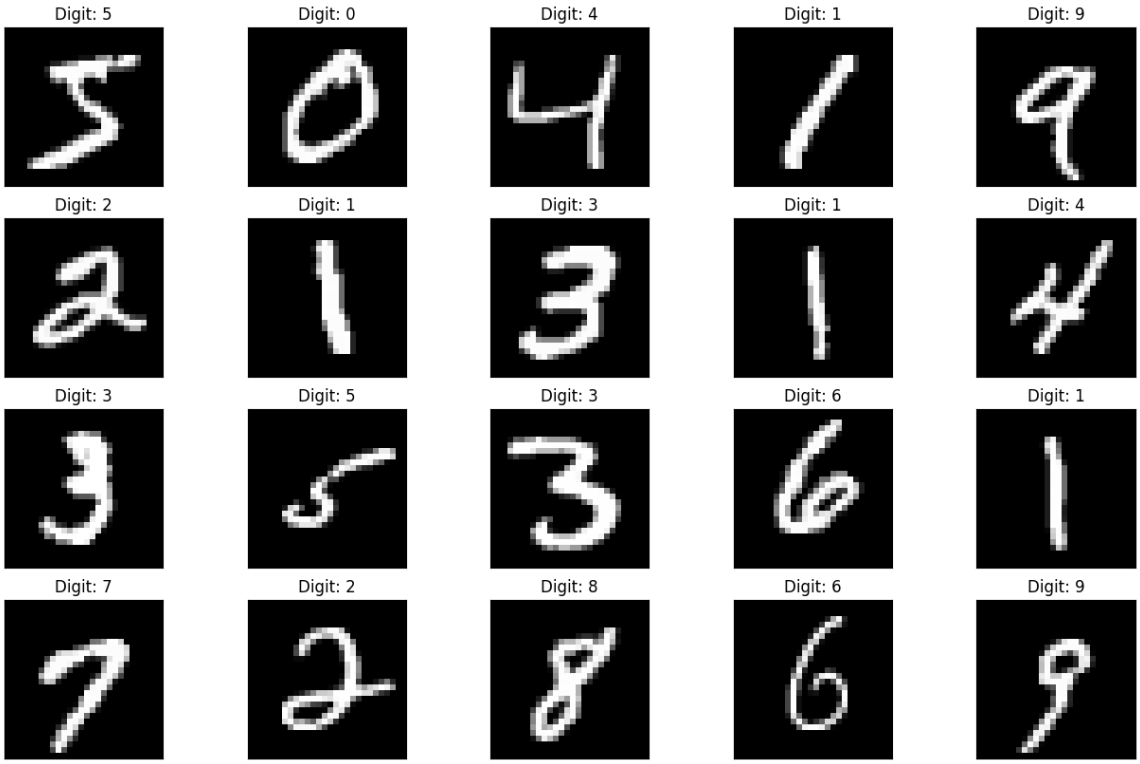
In [34]:
```
X_train[0].min(), X_train[0].max()
```

Out[34]:
```
(0, 255)
```

In [35]:
```
X_train = (X_train - 0.0) / (255.0 - 0.0)#RESHAPING THE DATA INTO 0 TO 1
X_test = (X_test - 0.0) / (255.0 - 0.0)
X_train[0].min(), X_train[0].max()
```

Out[35]:
```
(0.0, 1.0)
```

In [36]:
```
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 [37]:
```
X_train = X_train.reshape((X_train.shape + (1,)))
X_test = X_test.reshape((X_test.shape + (1,)))
```

In [38]:
```
y_train[0:20]
```

Out[38]:
```
array([5, 0, 4, 1, 9, 2, 1, 3, 1, 4, 3, 5, 3, 6, 1, 7, 2, 8, 6, 9],
      dtype=uint8)
```

In [39]:
```
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 [40]:
```
optimizer = SGD(learning_rate=0.01, momentum=0.9)
model.compile(
    optimizer=optimizer,
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_1 (MaxPoolin  (None, 13, 13, 32)        0
 g2D)

 flatten_1 (Flatten)         (None, 5408)              0

 dense_2 (Dense)             (None, 100)               540900

 dense_3 (Dense)             (None, 10)                1010

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

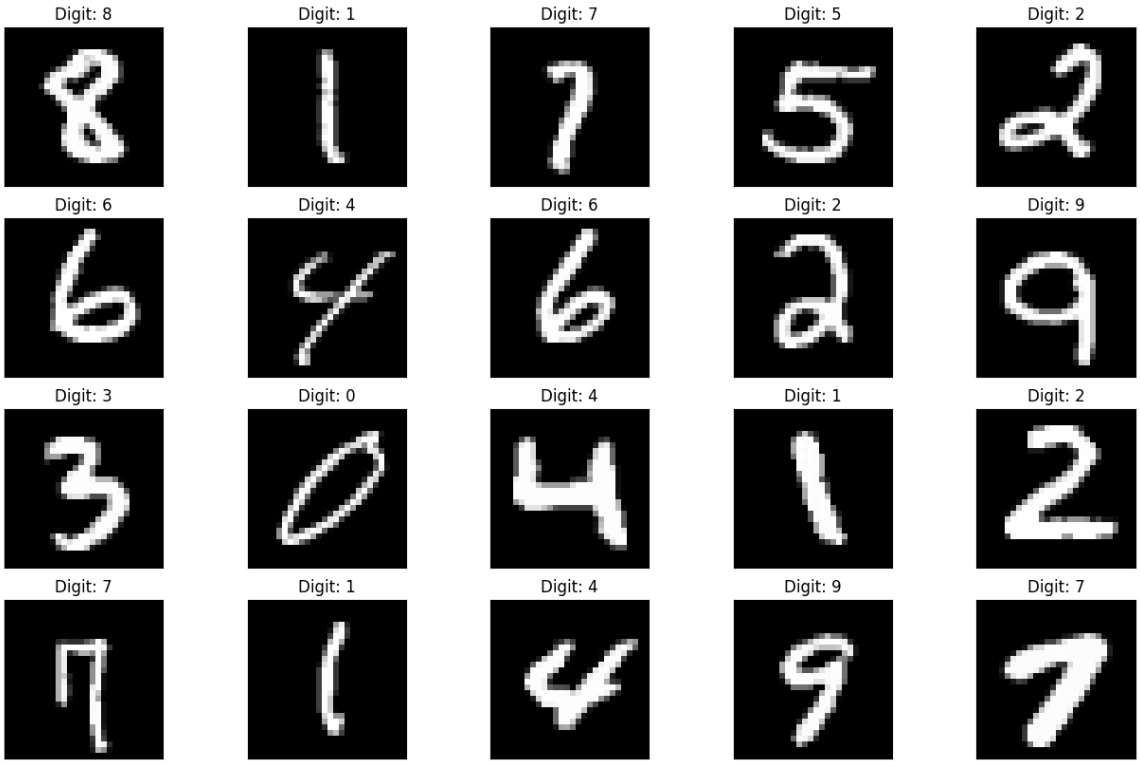
In [41]:
```
model.fit(X_train, y_train, epochs=10, batch_size=32)
```

```
Epoch 1/10
1875/1875 [==============================] - 19s 10ms/step - loss: 0.2302
- accuracy: 0.9301
Epoch 2/10
1875/1875 [==============================] - 17s 9ms/step - loss: 0.0698 -
accuracy: 0.9791
Epoch 3/10
1875/1875 [==============================] - 17s 9ms/step - loss: 0.0455 -
accuracy: 0.9865
Epoch 4/10
1875/1875 [==============================] - 17s 9ms/step - loss: 0.0326 -
accuracy: 0.9900
Epoch 5/10
1875/1875 [==============================] - 17s 9ms/step - loss: 0.0238 -
accuracy: 0.9927
Epoch 6/10
1875/1875 [==============================] - 18s 9ms/step - loss: 0.0170 -
accuracy: 0.9948
Epoch 7/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.0130
- accuracy: 0.9963
Epoch 8/10
1875/1875 [==============================] - 18s 9ms/step - loss: 0.0096 -
accuracy: 0.9974
Epoch 9/10
1875/1875 [==============================] - 18s 9ms/step - loss: 0.0066 -
accuracy: 0.9984
Epoch 10/10
1875/1875 [==============================] - 18s 10ms/step - loss: 0.0052
- accuracy: 0.9987
```

Out[41]:
```
<keras.src.callbacks.History at 0x1ef91456ad0>
```

In [46]:
```
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=
    plot_digit(image, digit, plt, i)
plt.show()
```

```
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 17ms/step
```

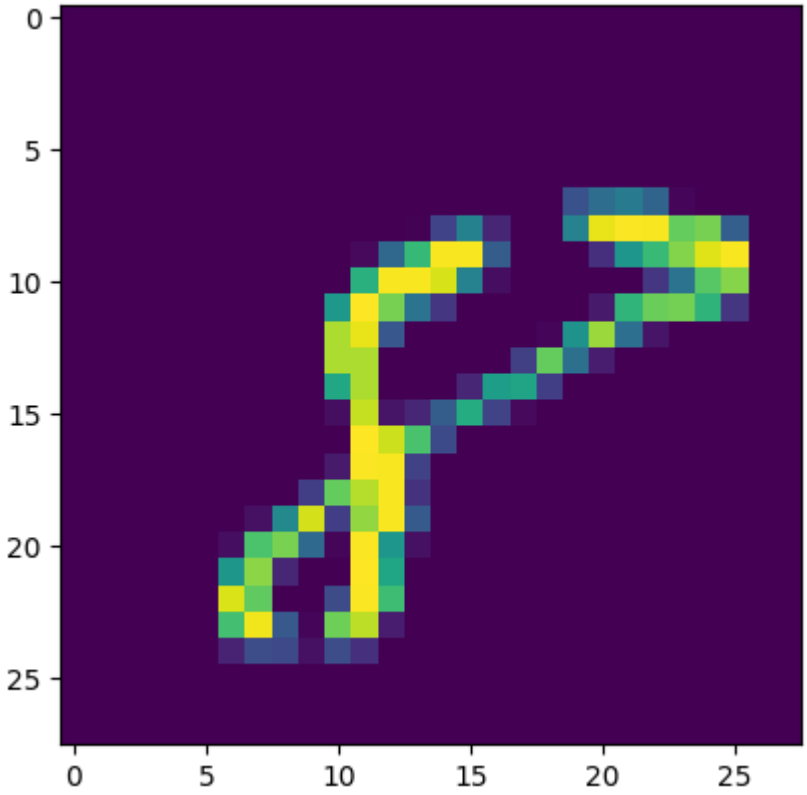


In [49]:
```
#pred = model.predict(X_test)
#classes_x=np.argmax(X_test,axis=1)

#predictions = np.argmax(model.predict(X_test), axis=-1)

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

```
313/313 [==============================] - 1s 4ms/step
```

Out[49]:
```
0.9875
```

In [50]:
```
n=random.randint(0,9999)
plt.imshow(X_test[n])
plt.show()
```



In [51]:
```
predicted_value=model.predict(X_test)
print("Handwritten number in the image is= %d" %np.argmax(predicted_value[n]
```

```
313/313 [==============================] - 2s 5ms/step
Handwritten number in the image is= 8
```

In [52]:
```
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: 0.04007310792803764
Test accuracy: 0.987500011920929
```

In [26]:
```
#The implemented CNN model is giving Loss=0.04624301567673683  and
#accuracy: 0.987500011920929 for test mnist dataset
```

In [ ]:

In [ ]: