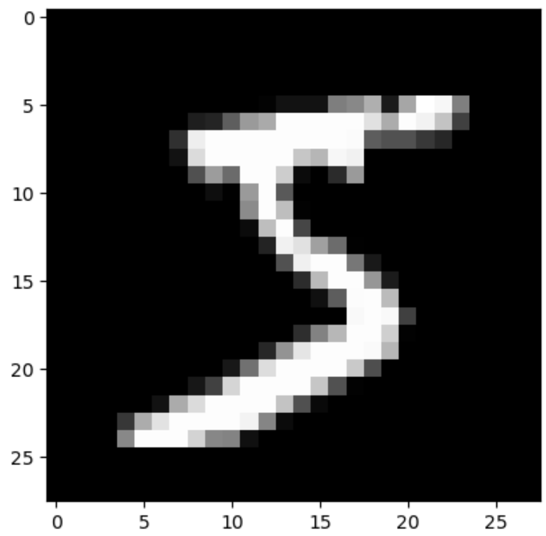
In [1]:
```
#Name :- Omkar Balwade
#Roll N.o :- 4107
#Div :- A
#Practical N.o 2(B):- Multiclass classification using Deep Neural Networks: Example: Use the OCR letter recogn
```

In [2]:
```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
from sklearn import metrics
```

```
WARNING:tensorflow:From C:\Users\Omkar\AppData\Local\anaconda3\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_e
```

In [3]:
```
# Load the OCR dataset
# The MNIST dataset is a built-in dataset provided by Keras.
# It consists of 70,000 28x28 grayscale images, each of which displays a single handwritten digit from 0 to 9.
# The training set consists of 60,000 images, while the test set has 10,000 images.
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

In [4]:
```
# X_train and X_test are our array of images while y_train and y_test are our array of labels for each image.
# The first tuple contains the training set features (X_train) and the training set labels (y_train).
# The second tuple contains the testing set features (X_test) and the testing set labels (y_test).
# For example, if the image shows a handwritten 7, then the label will be the integer 7.
plt.imshow(x_train[0], cmap='gray')  # imshow() function which simply displays an image.
plt.show()  # cmap is responsible for mapping a specific colormap to the values found in the array that you pa:
```



In [5]:
```
# This is because of the format that all the images in the dataset have:
# 1. All the images are grayscale, meaning they only contain black, white, and grey.
# 2. The images are 28 pixels by 28 pixels in size (28x28).
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   3  18  18  18 126 136
  175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
  225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251
   93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119
   25   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253
  150  27   0   0   0   0   0   0   0   0]
```

```
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252
  253 187   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249
  253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
  253 207   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253
  250 182   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201
   78   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   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 [6]:

```python
# image data is just an array of digits. You can almost make out a 5 from the pattern of the digits in the arra
# Array of 28 values
# a grayscale pixel is stored as a digit between 0 and 255 where 0 is black, 255 is white and values in betweer
# Therefore, each value in the [28][28] array tells the computer which color to put in that position when.
# reformat our X_train array and our X_test array because they do not have the correct shape.
# Reshape the data to fit the model
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 [7]:

```python
# X: Training data of shape (n_samples, n_features)
# y: Training label values of shape (n_samples, n_labels)
# 2D array of height and width, 28 pixels by 28 pixels will just become 784 pixels (28 squared).
# Remember that X_train has 60,000 elements, each with 784 total pixels so will become shape (60000,784).
# Whereas X_test has 10,000 elements, each with each with 784 total pixels so will become shape (10000, 784).
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')  # use 32-bit precision when training a neural network, so at one point th
x_test = x_test.astype('float32')
x_train /= 255  # Each image has Intensity from 0 to 255x_test
x_test /= 255
```

In [8]:

```python
# Regarding the division by 255, this is the maximum value of a byte (the input feature's type before the conve
# so this will ensure that the input features are scaled between 0.0 and 1.0.

# Convert class vectors to binary class matrices
num_classes = 10
y_train = np.eye(num_classes)[y_train]  # Return a 2-D array with ones on the diagonal and zeros elsewhere.
y_test = np.eye(num_classes)[y_test]  # if your particular categories are present then it marks as 1 else 0 in
```

In [9]:

```python
# Define the model architecture
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))  # Input consists of 784 Neuron ie 784 input,512
model.add(Dropout(0.2))  # DROP OUT RATIO 20%
model.add(Dense(512, activation='relu'))  # returns a sequence of another vectors of dimension 512
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))  # 10 neurons ie output node in the output layer.
```

```
WARNING:tensorflow:From C:\Users\Omkar\AppData\Local\anaconda3\Lib\site-packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecat
```

In [10]:

```python
# Compile the model
model.compile(loss='categorical_crossentropy',  # for a multi-class classification problemoptimizer=RMSprop(),
              metrics=['accuracy'])
```

```
WARNING:tensorflow:From C:\Users\Omkar\AppData\Local\anaconda3\Lib\site-packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer i
```

In [11]:

```python
# Train the model
batch_size = 128  # batch_size argument is passed to the layer to define a batch size for the inputs.
epochs = 20
history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1,
                    validation_data=(x_test, y_test))
# Using validation_data means you are providing the training set and validation set yourself,
# 60000image/128=469 batch each
```

```
Epoch 1/20
WARNING:tensorflow:From C:\Users\Omkar\AppData\Local\anaconda3\Lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorVal

WARNING:tensorflow:From C:\Users\Omkar\AppData\Local\anaconda3\Lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.executing_eag

469/469 [==============================] - 9s 17ms/step - loss: 0.2545 - accuracy: 0.9226 - val_loss: 0.1095 - val_accuracy: 0.9660
Epoch 2/20
469/469 [==============================] - 8s 17ms/step - loss: 0.1043 - accuracy: 0.9692 - val_loss: 0.0895 - val_accuracy: 0.9730
Epoch 3/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0748 - accuracy: 0.9771 - val_loss: 0.0814 - val_accuracy: 0.9746
Epoch 4/20
469/469 [==============================] - 8s 18ms/step - loss: 0.0589 - accuracy: 0.9816 - val_loss: 0.0607 - val_accuracy: 0.9806
Epoch 5/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0483 - accuracy: 0.9854 - val_loss: 0.0645 - val_accuracy: 0.9815
Epoch 6/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0391 - accuracy: 0.9872 - val_loss: 0.0773 - val_accuracy: 0.9802
Epoch 7/20
469/469 [==============================] - 8s 16ms/step - loss: 0.0348 - accuracy: 0.9890 - val_loss: 0.0619 - val_accuracy: 0.9837
Epoch 8/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0286 - accuracy: 0.9906 - val_loss: 0.0688 - val_accuracy: 0.9823
Epoch 9/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0264 - accuracy: 0.9911 - val_loss: 0.0677 - val_accuracy: 0.9833
Epoch 10/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0223 - accuracy: 0.9929 - val_loss: 0.0686 - val_accuracy: 0.9835
Epoch 11/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0208 - accuracy: 0.9933 - val_loss: 0.0650 - val_accuracy: 0.9843
Epoch 12/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0187 - accuracy: 0.9938 - val_loss: 0.0797 - val_accuracy: 0.9824
Epoch 13/20
469/469 [==============================] - 9s 18ms/step - loss: 0.0159 - accuracy: 0.9948 - val_loss: 0.0699 - val_accuracy: 0.9834
Epoch 14/20
469/469 [==============================] - 8s 18ms/step - loss: 0.0141 - accuracy: 0.9954 - val_loss: 0.0666 - val_accuracy: 0.9853
Epoch 15/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0138 - accuracy: 0.9952 - val_loss: 0.0710 - val_accuracy: 0.9850
Epoch 16/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0145 - accuracy: 0.9952 - val_loss: 0.0768 - val_accuracy: 0.9838
Epoch 17/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0120 - accuracy: 0.9959 - val_loss: 0.0784 - val_accuracy: 0.9855
Epoch 18/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0100 - accuracy: 0.9969 - val_loss: 0.0818 - val_accuracy: 0.9850
Epoch 19/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0098 - accuracy: 0.9966 - val_loss: 0.0868 - val_accuracy: 0.9850
Epoch 20/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0107 - accuracy: 0.9964 - val_loss: 0.0845 - val_accuracy: 0.9842
```

In [12]:

```python
# Evaluate the model
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.08453793078660965
Test accuracy: 0.9842000007629395
```

In [ ]: