The MNIST database (Modified National Institute of Standards and Technology database is a large database of handwritten digits that is commonly used for training various image processing systems.
The MNIST database contains 60,000 training images and 10,000 testing images.

In [14]:
```
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
import numpy
```

Loading the data

In [15]:
```
(Xtrain, ytrain), (Xtest, ytest) = mnist.load_data()
```

In [16]:
```
#total 60000 images
#each width-28, height-28
Xtrain.shape
```

Out[16]: (60000, 28, 28)

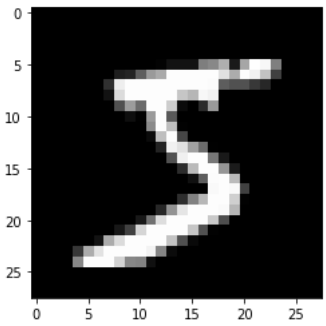
In [4]:
```
ytrain[0]
```

Out[4]: 5

In [5]:
```
import matplotlib.pyplot as plt

plt.imshow(Xtrain[0], cmap=plt.get_cmap('gray'))
```

Out[5]: <matplotlib.image.AxesImage at 0x22dcf73d340>




In [6]:
```
Xtest.shape
```

Out[6]: (10000, 28, 28)

In [7]:
```
ytest.shape
```

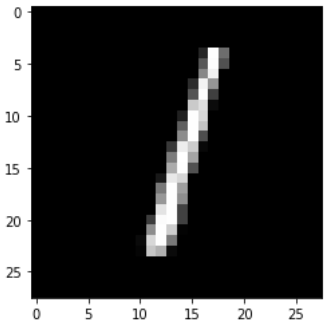
Out[7]: (10000,)

In [59]:
```
ytest[2]
```

Out[59]: 1

In [64]:
```
plt.imshow(Xtest[2], cmap=plt.get_cmap('gray'))
```

Out[64]: <matplotlib.image.AxesImage at 0x22dd403e490>



Preprocess the input data

In [10]:
```
#convert the range of data from high values to 0-1

#Ex: [0 255 250 10 100] -> original data [0-255]
#    [0 255 250 10 100]/255.0 -> [0  1  0.95  0.05  0.4]

Xtrain = Xtrain / 255.0
Xtest = Xtest / 255.0
```

In [11]:
```
#convert labels into one hot encoded labels

from tensorflow.keras.utils import to_categorical

y_train = to_categorical(ytrain, 10)
y_test = to_categorical(ytest, 10)
```

In [12]:
```
ytrain[0]
```

Out[12]: 5

In [13]:
```
y_train[0]
```

Out[13]: array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)

Building CNN Model
A Convolutional Neural Network, also known as CNN or ConvNet, is a class of neural networks that specializes in processing data that has a grid-like topology, such as an image. A digital image is a binary representation of visual data.

In [50]:
```
model= Sequential()
```

In [51]:
```
# define cnn model


model.add(Conv2D(filters=64, kernel_size = (3,3), activation="relu", input_shape=(28,28,1)))
model.add(Conv2D(filters=64, kernel_size = (3,3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))


model.add(Conv2D(filters=128, kernel_size = (3,3), activation="relu"))
model.add(Conv2D(filters=128, kernel_size = (3,3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))


model.add(Conv2D(filters=256, kernel_size = (3,3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())
model.add(Dense(512,activation="relu"))

model.add(Dense(10,activation="softmax"))

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
```

In [52]:
```
from keras.callbacks import ModelCheckpoint

filename = 'bestmodel.h5'
mc = ModelCheckpoint(filename, monitor='val_loss', verbose=1, mode='min')
```

In [54]:
```
history =model.fit(Xtrain, y_train, epochs=10, validation_split=0.25, callbacks=[mc])
```

```
Epoch 1/10
1407/1407 [==============================] - ETA: 0s - loss: 0.1747 - accuracy: 0.9510
Epoch 1: saving model to bestmodel.h5
1407/1407 [==============================] - 515s 361ms/step - loss: 0.1747 - accuracy: 0.9510 - val_loss: 0.0990 - val_accurac
y: 0.9731
Epoch 2/10
1407/1407 [==============================] - ETA: 0s - loss: 0.0565 - accuracy: 0.9830
Epoch 2: saving model to bestmodel.h5
1407/1407 [==============================] - 462s 328ms/step - loss: 0.0565 - accuracy: 0.9830 - val_loss: 0.0704 - val_accurac
y: 0.9791
Epoch 3/10
1407/1407 [==============================] - ETA: 0s - loss: 0.0457 - accuracy: 0.9866
Epoch 3: saving model to bestmodel.h5
1407/1407 [==============================] - 485s 345ms/step - loss: 0.0457 - accuracy: 0.9866 - val_loss: 0.0670 - val_accurac
y: 0.9796
Epoch 4/10
1407/1407 [==============================] - ETA: 0s - loss: 0.0380 - accuracy: 0.9886
Epoch 4: saving model to bestmodel.h5
1407/1407 [==============================] - 508s 361ms/step - loss: 0.0380 - accuracy: 0.9886 - val_loss: 0.0555 - val_accurac
y: 0.9859
Epoch 5/10
1407/1407 [==============================] - ETA: 0s - loss: 0.0340 - accuracy: 0.9902
Epoch 5: saving model to bestmodel.h5
1407/1407 [==============================] - 478s 340ms/step - loss: 0.0340 - accuracy: 0.9902 - val_loss: 0.0866 - val_accurac
y: 0.9781
Epoch 6/10
1407/1407 [==============================] - ETA: 0s - loss: 0.0336 - accuracy: 0.9900
Epoch 6: saving model to bestmodel.h5
1407/1407 [==============================] - 467s 332ms/step - loss: 0.0336 - accuracy: 0.9900 - val_loss: 0.0471 - val_accurac
y: 0.9878
Epoch 7/10
1407/1407 [==============================] - ETA: 0s - loss: 0.0276 - accuracy: 0.9919
Epoch 7: saving model to bestmodel.h5
1407/1407 [==============================] - 489s 347ms/step - loss: 0.0276 - accuracy: 0.9919 - val_loss: 0.0593 - val_accurac
y: 0.9822
Epoch 8/10
1407/1407 [==============================] - ETA: 0s - loss: 0.0231 - accuracy: 0.9932
Epoch 8: saving model to bestmodel.h5
1407/1407 [==============================] - 548s 390ms/step - loss: 0.0231 - accuracy: 0.9932 - val_loss: 0.0654 - val_accurac
y: 0.9869
Epoch 9/10
1407/1407 [==============================] - ETA: 0s - loss: 0.0263 - accuracy: 0.9928
Epoch 9: saving model to bestmodel.h5
1407/1407 [==============================] - 835s 594ms/step - loss: 0.0263 - accuracy: 0.9928 - val_loss: 0.0461 - val_accurac
y: 0.9893
Epoch 10/10
1407/1407 [==============================] - ETA: 0s - loss: 0.0188 - accuracy: 0.9946
Epoch 10: saving model to bestmodel.h5
1407/1407 [==============================] - 637s 453ms/step - loss: 0.0188 - accuracy: 0.9946 - val_loss: 0.0477 - val_accurac
y: 0.9890
```

In [55]:
```
from keras.models import load_model
```

In [56]:
```
bestmodel = load_model('bestmodel.h5')
```

In [57]:
```
bestmodel.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 26, 26, 64)        640

 conv2d_5 (Conv2D)           (None, 24, 24, 64)        36928

 max_pooling2d_4 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 conv2d_6 (Conv2D)           (None, 10, 10, 128)       73856

 conv2d_7 (Conv2D)           (None, 8, 8, 128)         147584

 max_pooling2d_5 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 conv2d_8 (Conv2D)           (None, 2, 2, 256)         295168

 max_pooling2d_6 (MaxPooling  (None, 1, 1, 256)        0
 2D)

 flatten_4 (Flatten)         (None, 256)               0

 dense_8 (Dense)             (None, 512)               131584

 dense_9 (Dense)             (None, 10)                5130

=================================================================
Total params: 690,890
Trainable params: 690,890
Non-trainable params: 0
_________________________________________________________________
```

In [58]:
```
bestmodel.evaluate(Xtest, y_test)
```

```
313/313 [==============================] - 27s 85ms/step - loss: 0.0409 - accuracy: 0.9909
```

Out[58]:
```
[0.04092770069837570, 0.9908999800682068]
```

In [ ]:
```

```