Task 5:- Develop A Neural Network That Can Read Handwriting:

Author:- Mitali D Shinde

Level:-Advanced

In [1]:
```
import tensorflow as tf
from tensorflow.keras import datasets,layers,models
import matplotlib.pyplot as plt
%matplotlib.inline
from keras.models import model_from_json
```

```
UsageError: Line magic function `%matplotlib.inline` not found.
```

Download MNIST dataset

THe MNSIT databse(Modified National Instiute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems.

In [2]:
```
(train_img, train_labels), (test_img, test_labels)= datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 5s 0us/step
11501568/11490434 [==============================] - 5s 0us/step
```

Normalize Images

In [3]:
```
train_img , test_img =train_img/255.0 , test_img/255.0
#normalizing the pixal value between 0 and 1
```

In [4]:
```
len(train_img)
```

Out[4]:
```
60000
```

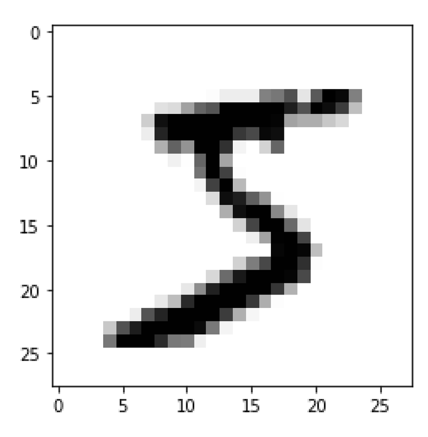
In [5]:
```
len(test_img)
```

Out[5]:
```
10000
```

Visualizing Images

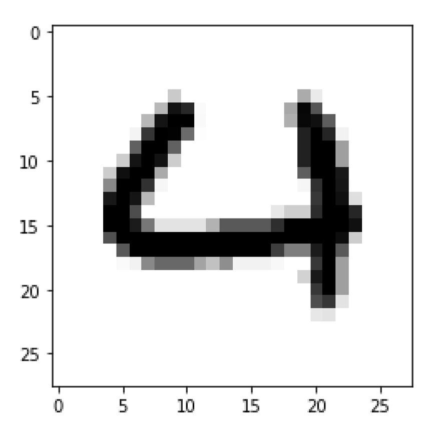
In [6]:
```
plt.imshow(train_img[0], cmap = plt.cm.gray_r, interpolation = 'nearest')
```

Out[6]:
```
<matplotlib.image.AxesImage at 0x206dad66250>
```



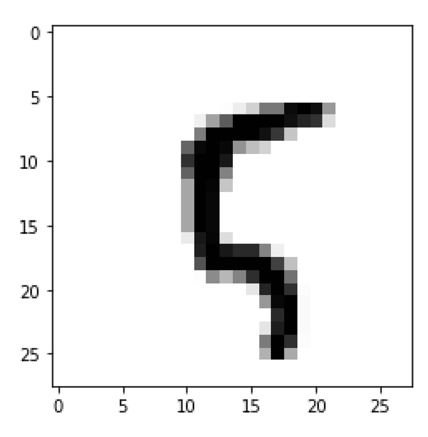
In [10]:
```
plt.imshow(train_img[60], cmap = plt.cm.gray_r, interpolation = 'nearest')
```

Out[10]:
```
<matplotlib.image.AxesImage at 0x206db6341c0>
```



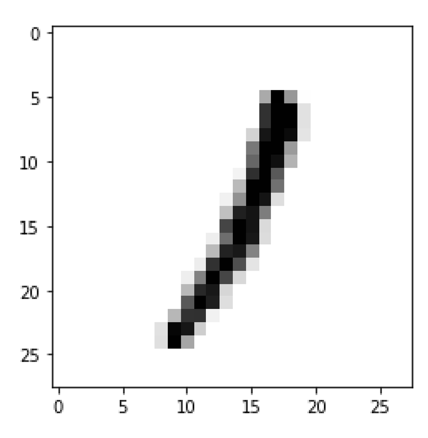
In [7]:
```
plt.imshow(train_img[100], cmap = plt.cm.gray_r, interpolation = 'nearest')
```

Out[7]:
```
<matplotlib.image.AxesImage at 0x206db4f1d00>
```



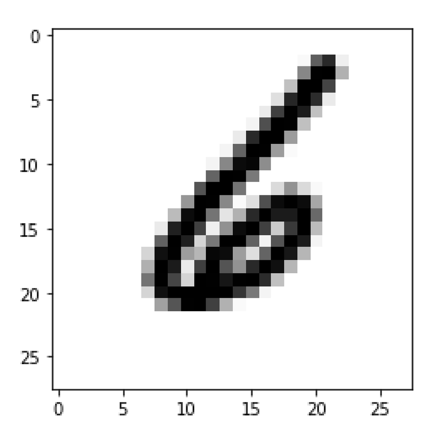
In [8]:
```
plt.imshow(train_img[99], cmap = plt.cm.gray_r, interpolation = 'nearest')
```

Out[8]:
```
<matplotlib.image.AxesImage at 0x206db5619d0>
```



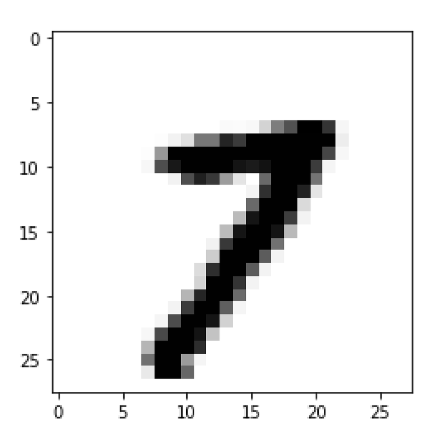
In [9]:
```
plt.imshow(train_img[999], cmap = plt.cm.gray_r, interpolation = 'nearest')
```

Out[9]:
```
<matplotlib.image.AxesImage at 0x206db5c3f10>
```



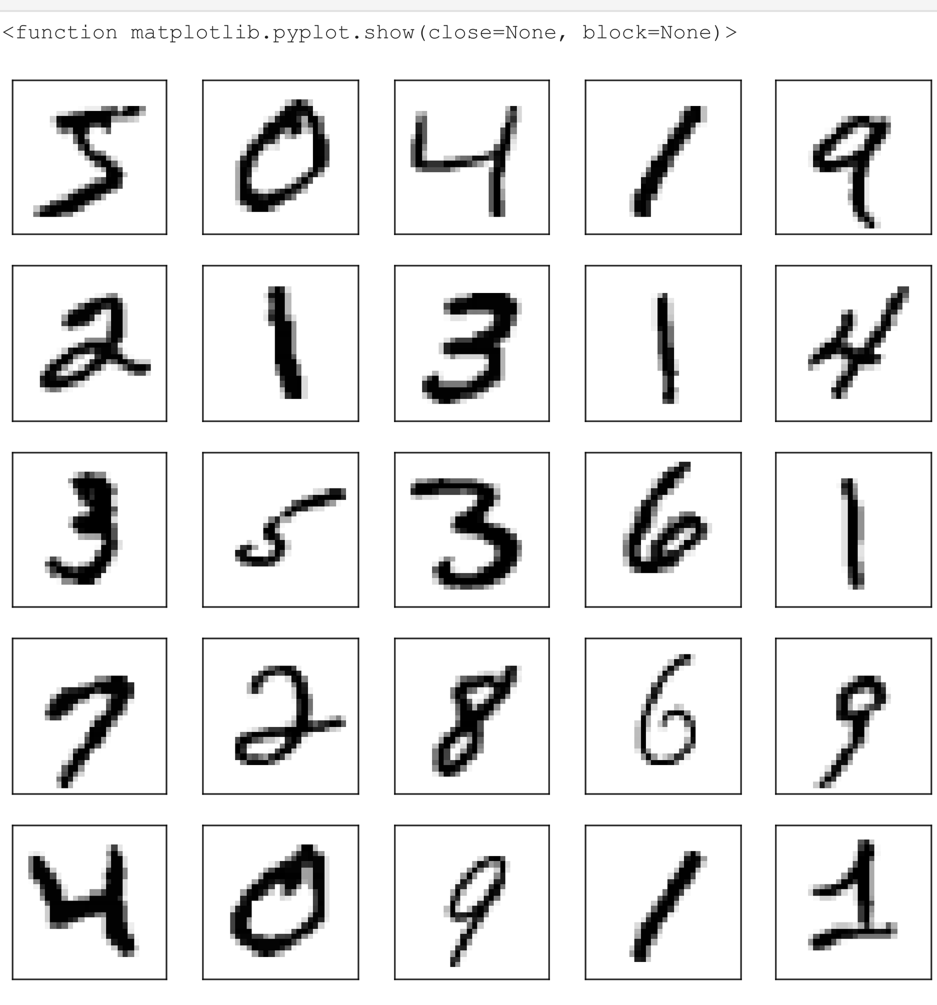
In [11]:
```
plt.imshow(train_img[40000], cmap = plt.cm.gray_r, interpolation = 'nearest')
```

Out[11]:
```
<matplotlib.image.AxesImage at 0x206db6954c0>
```



In [12]:
```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_img[i], cmap=plt.cm.binary)
plt.show
```

Out[12]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Creating and training the model

Sequential model

In [13]:
```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,28)),

    tf.keras.layers.Dense(128,activation='relu'),

    tf.keras.layers.Dense(10)
])
```

In [25]:
```
model.compile(
   optimizer=tf.keras.optimizers.Adam(0.001),
   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
   metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],

)
```

In [27]:
```
model.fit(
     train_img , train_labels,
     epochs = 10,
    validation_data=(test_img, test_labels)
)
```

```
Epoch 1/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2595 - sparse_categorical_accuracy: 0.9258 -
val_loss: 0.1423 - val_sparse_categorical_accuracy: 0.9563
Epoch 2/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1154 - sparse_categorical_accuracy: 0.9657 -
val_loss: 0.1038 - val_sparse_categorical_accuracy: 0.9682
Epoch 3/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0794 - sparse_categorical_accuracy: 0.9767 -
val_loss: 0.0898 - val_sparse_categorical_accuracy: 0.9736
Epoch 4/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0601 - sparse_categorical_accuracy: 0.9812 -
val_loss: 0.0881 - val_sparse_categorical_accuracy: 0.9732
Epoch 5/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0470 - sparse_categorical_accuracy: 0.9854 -
val_loss: 0.0759 - val_sparse_categorical_accuracy: 0.9776
Epoch 6/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0365 - sparse_categorical_accuracy: 0.9887 -
val_loss: 0.0820 - val_sparse_categorical_accuracy: 0.9741
Epoch 7/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0296 - sparse_categorical_accuracy: 0.9906 -
val_loss: 0.0721 - val_sparse_categorical_accuracy: 0.9780
Epoch 8/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0230 - sparse_categorical_accuracy: 0.9926 -
val_loss: 0.0662 - val_sparse_categorical_accuracy: 0.9804
Epoch 9/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0195 - sparse_categorical_accuracy: 0.9940 -
val_loss: 0.0864 - val_sparse_categorical_accuracy: 0.9758
Epoch 10/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0160 - sparse_categorical_accuracy: 0.9948 -
val_loss: 0.0888 - val_sparse_categorical_accuracy: 0.9750
```

Out[27]:
```
<keras.callbacks.History at 0x206dc9556d0>
```

Model summary

summary() method will display the architecture of the model

In [32]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
_________________________________________________________________
flatten_1 (Flatten)          (None, 10)                0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                704
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 103,124
Trainable params: 103,124
Non-trainable params: 0
_________________________________________________________________
```

Flatten

In [29]:
```
model.add(layers.Flatten())
```

Adding dense layer

In [30]:
```
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
```

Compile

In [31]:
```
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),metrics=['a
```

Training the model

In [33]:
```
model.fit(train_img, train_labels, epochs=12, validation_data=(test_img, test_labels))
```

```
Epoch 1/12
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0803 - accuracy: 0.9785 - val_loss: 0.0835 -
val_accuracy: 0.9775
Epoch 2/12
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0205 - accuracy: 0.9934 - val_loss: 0.1099 -
val_accuracy: 0.9738
Epoch 3/12
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0169 - accuracy: 0.9942 - val_loss: 0.0930 -
val_accuracy: 0.9778
Epoch 4/12
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0163 - accuracy: 0.9945 - val_loss: 0.0969 -
val_accuracy: 0.9803
Epoch 5/12
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0166 - accuracy: 0.9946 - val_loss: 0.1161 -
val_accuracy: 0.9773
Epoch 6/12
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0151 - accuracy: 0.9948 - val_loss: 0.1241 -
val_accuracy: 0.9752
Epoch 7/12
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0121 - accuracy: 0.9959 - val_loss: 0.1189 -
val_accuracy: 0.9775
Epoch 8/12
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0152 - accuracy: 0.9952 - val_loss: 0.1121 -
val_accuracy: 0.9771
Epoch 9/12
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0117 - accuracy: 0.9965 - val_loss: 0.1323 -
val_accuracy: 0.9749
Epoch 10/12
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0115 - accuracy: 0.9961 - val_loss: 0.1200 -
val_accuracy: 0.9799
Epoch 11/12
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0100 - accuracy: 0.9969 - val_loss: 0.1282 -
val_accuracy: 0.9778
Epoch 12/12
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0124 - accuracy: 0.9959 - val_loss: 0.1238 -
val_accuracy: 0.9780
```

Out[33]:
```
<keras.callbacks.History at 0x206dece8400>
```

Testing Model with custom image

In [35]:
```
import cv2
```

In [60]:
```
img  = cv2.imread('5.png')
plt.imshow(img)
```

Out[60]:
```
<matplotlib.image.AxesImage at 0x206fc2abc10>
```

In [62]:
```
img.shape
```

Out[62]:
```
(237, 218, 3)
```

In [63]:
```
img = cv2.resize((cv2.cvtColor(img , cv2.COLOR_BGR2GRAY)), (28,28 ) , interpolation=cv2.INTER_AREA)
```

In [64]:
```
img.shape
```

Out[64]:
```
(28, 28)
```

In [69]:
```
import tensorflow
```

In [70]:
```
img = tensorflow.keras.utils.normalize(img, axis=1)
```

In [71]:
```
img = np.array(img).reshape(-1,28,28,1)
```

In [72]:
```
img.shape
```

Out[72]:
```
(1, 28, 28, 1)
```

In [74]:
```
predictions = model.predict(img)
```

In [81]:
```
print("Number Predicted:")
print(np.argmax(model.predict(img)))
```

```
Number Predicted:
5
```

Model Accuracy

AS we can see from above model testing our model accuracy is 99%