Level :3 - Task 1

- Objective : To develop a neural network that can read handwriting.
- dataset : https://www.kaggle.com/datasets/xainano/handwrittenmathsymbols
- Author : Mayuri Arun Pathak, Data Science Intern
- Email :mayuripathak2024@gmail.com

- Setting Wroking directory

In [1]:
```
import os
os.chdir("H:\\Data Science\\Internship\\Spark")
```

- Import required libraries

In [2]:
```
import numpy as np
from tensorflow.keras.datasets import mnist
import tensorflow as tf
import matplotlib.pyplot as plt
```

- Loading the Minist dataset

- The MNIST Dataset is conatined in the Tensorflow library and can be loaded using Keras. The dataset is then further divided into the training and test sets

In [4]:
```
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 3s 0us/step
```

- Visualizing the dataset

In [5]:
```
X_train.shape
```
Out[5]:
```
(60000, 28, 28)
```

- The training dataset contains 60,000 images where each is 28x28 pixel greyscale image.

In [6]:
```
Y_train.shape
```
Out[6]:
```
(60000,)
```

In [7]:
```
X_test.shape
```
Out[7]:
```
(10000, 28, 28)
```

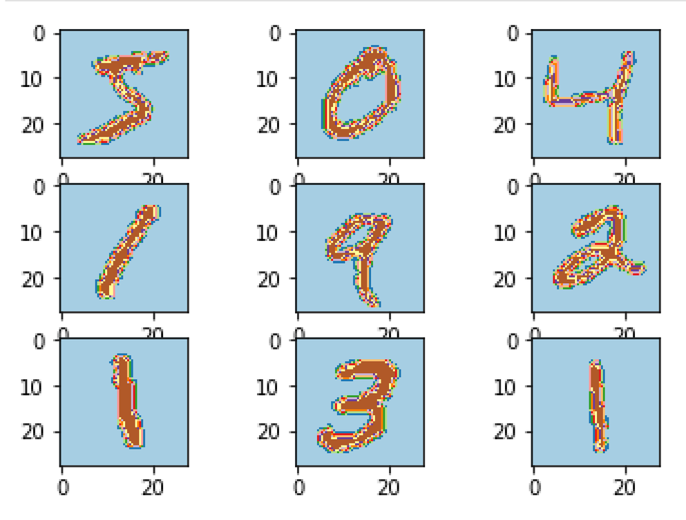
- Training dataset contains 10,000 images where each is 28x28 pixel greyscale image

- Checking the values of each pixel before the normalization process

In [8]:
```
X_train[0]
```
Out[8]:
```
array([[  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]], dtype=uint8)
```

Visualising the dataset

In [11]:
```
for i in range(9):
    plt.subplot(330 + 1 + i)
    plt.imshow(X_train[i], cmap=plt.get_cmap('Paired'))
plt.show()
```



- Reshaping the data

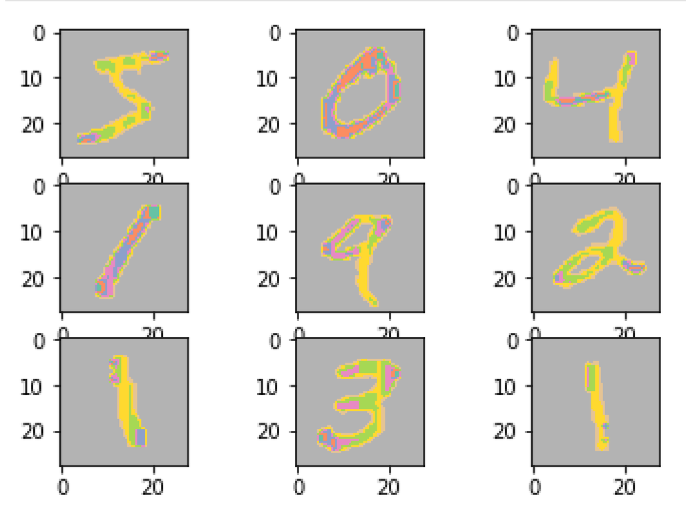
In [12]:
```
X_train_r = X_train.reshape(X_train.shape[0], X_train.shape[2], X_train.shape[2], 1)
X_test_r = X_test.reshape(X_test.shape[0], X_test.shape[2], X_test.shape[2], 1)
```

- Normalizing the data

In [13]:
```
X_train_r = tf.keras.utils.normalize(X_train_r, axis = 1)
X_test_r = tf.keras.utils.normalize(X_test_r, axis = 1)
```

- Visualising the data after normalization

In [23]:
```
for i in range(9):
  plt.subplot(330 + 1 + i)
  plt.imshow(np.squeeze(X_train_r[i]), cmap=plt.get_cmap('Set2_r'))
plt.show()
```



- Creating a simple CNN Model

In [24]:
```
image_shape = X_train_r.shape[1:]
print(image_shape)
```
```
(28, 28, 1)
```

In [25]:
```
model = model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(48, (3,3), activation = tf.nn.relu, input_shape = image_shape))
model.add(tf.keras.layers.MaxPool2D((2,2)))
model.add(tf.keras.layers.Conv2D(64, (3,3), activation = tf.nn.relu))
model.add(tf.keras.layers.MaxPool2D((2,2)))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(256, activation = tf.nn.relu))
model.add(tf.keras.layers.Dense(10, activation = tf.nn.softmax))
```

In [26]:
```
model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics = ['accuracy'])
model.fit(X_train_r, Y_train, epochs = 10)
```
```
Epoch 1/10
1875/1875 [==============================] - 134s 70ms/step - loss: 0.1864 - accuracy: 0.9420
Epoch 2/10
1875/1875 [==============================] - 133s 71ms/step - loss: 0.0712 - accuracy: 0.9780
Epoch 3/10
1875/1875 [==============================] - 120s 64ms/step - loss: 0.0556 - accuracy: 0.9825
Epoch 4/10
1875/1875 [==============================] - 154s 82ms/step - loss: 0.0461 - accuracy: 0.9850
Epoch 5/10
1875/1875 [==============================] - 145s 77ms/step - loss: 0.0379 - accuracy: 0.9879
Epoch 6/10
1875/1875 [==============================] - 127s 68ms/step - loss: 0.0346 - accuracy: 0.9888
Epoch 7/10
1875/1875 [==============================] - 131s 70ms/step - loss: 0.0297 - accuracy: 0.9903
Epoch 8/10
1875/1875 [==============================] - 131s 70ms/step - loss: 0.0260 - accuracy: 0.9915
Epoch 9/10
1875/1875 [==============================] - 127s 68ms/step - loss: 0.0238 - accuracy: 0.9920
Epoch 10/10
1875/1875 [==============================] - 125s 67ms/step - loss: 0.0234 - accuracy: 0.9924
```
Out[26]:
```
<keras.callbacks.History at 0x207bdf80dc0>
```

- Summary of the CNN Model

In [27]:
```
model.summary()
```
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 48)        480

 max_pooling2d (MaxPooling2D  (None, 13, 13, 48)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        27712

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 dropout (Dropout)           (None, 5, 5, 64)          0

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 256)               409856

 dense_1 (Dense)             (None, 10)                2570

=================================================================
Total params: 440,618
Trainable params: 440,618
Non-trainable params: 0
_________________________________________________________________
```

- Evaluating the model based on validation loss and validation accuracy

In [64]:
```
val_loss, val_accuracy = model.evaluate(X_test, Y_test)
print("Validation loss : ", val_loss*100, "%")
print("Validation accuracy : ", val_accuracy*100,"%")
```
```
313/313 [==============================] - 3s 8ms/step - loss: 34.4281 - accuracy: 0.9636
Validation loss :  3442.8115844726562 %
Validation accuracy :  96.35999798774719 %
```

- Saving the Model

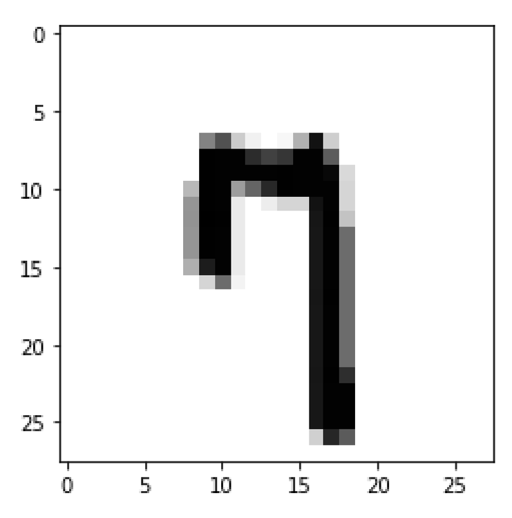
In [36]:
```
model.save('/content/drive/MyDrive/Machine Learning Projects/Mnist.model')
```
```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _update_step_xla while saving (showing 3 of 3). These functions will not b
e directly callable after loading.
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Machine Learning Projects/Mnist.model\assets
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Machine Learning Projects/Mnist.model\assets
```

Predicting a random test case

In [41]:
```
new_model = tf.keras.models.load_model("/content/drive/MyDrive/Machine Learning Projects/Mnist.model")
```

In [42]:
```
predictions = new_model.predict(X_test)
```
```
313/313 [==============================] - 5s 15ms/step
```

In [43]:
```
plt.imshow(X_test[80], cmap = plt.get_cmap('binary'))
```
Out[43]:
```
<matplotlib.image.AxesImage at 0x207a25031f0>
```



In [44]:
```
print("The prediction is : ", np.argmax(predictions[80]))
```
```
The prediction is :  7
```

- The Prediction is :7

Suggestions are always Welcome!

Thank You!