Sprint2

November 8, 2022

1 SPRINT 2 - Model Building

2 A Novel Method For Handwritten Digit Recognition System

3 Team ID : PNT2022TMID38424

4 Importing The Required Libraries

[ ]:
```
import numpy
import tensorflow
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.layers import Conv2D
from keras.optimizers import Adam
from keras.utils import np_utils
```

5 Loading The Data

[ ]:
```
(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 [==============================] - 0s 0us/step
```

[ ]:
```
#Shapeing data
print(x_train.shape)
print(x_test.shape)
```

```
(60000, 28, 28)
(10000, 28, 28)
```

6 Analyzing The Data

[ ]:
```
#printing the first image
x_train[0]
```

[ ]:
```
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],
```

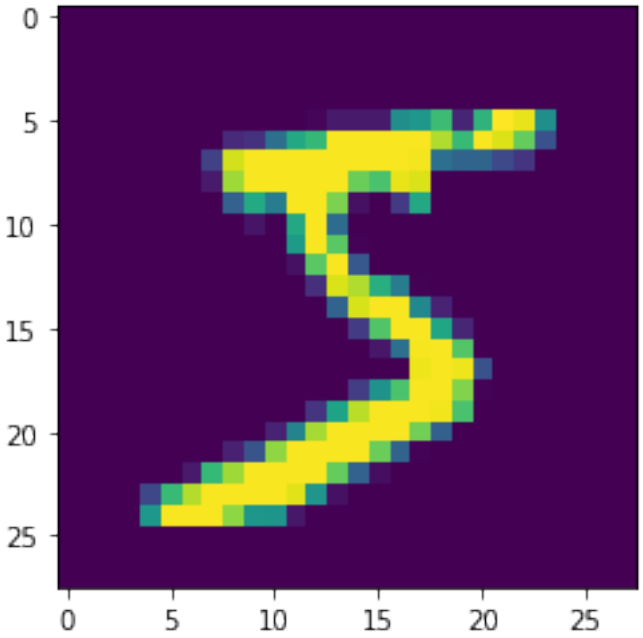
```
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  81, 240, 253, 253, 119,  25,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,  45, 186, 253, 253, 150,  27,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   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)
```

[ ]:
```
#printing lable of first image
y_train[0]
```

[ ]:
```
5
```

[ ]:
```
#Used for data visualization
import matplotlib.pyplot as plt
#Ploating the index = 0 image
plt.imshow(x_train[0])
```

[ ]:
```
<matplotlib.image.AxesImage at 0x7f817867d990>
```



7 Reshaping The Data

[ ]:
```
#Reshaping to format which CNN expects(batch, height, width, Channels)
x_train = x_train.reshape(60000, 28,28,1).astype('float32')
x_test = x_test.reshape(10000,28,28,1).astype('float32')
```

8 Applying One Hot Encoding

[ ]:
```
#One-Hot Encoding
number_of_classes = 10 # Storing the no. classes in a variable
#Converts the output in binary format
y_train = np_utils.to_categorical(y_train,number_of_classes)
y_test = np_utils.to_categorical(y_test,number_of_classes)
```

[ ]:
```
#Printing the new label
y_train[0]
```

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

9 Add CNN Layers

[ ]:
```
#Create Model
model = Sequential()
#adding model layer
model.add(Conv2D(64,(3,3),input_shape = (28,28,1),activation='relu'))
model.add(Conv2D(32,(3,3),activation='relu'))
#flatten the dimenstion of the image
model.add(Flatten())
model.add(Dense(number_of_classes,activation='softmax'))
```

10 Compiling The Model

[ ]:
```
model.
 ↪compile(loss='categorical_crossentropy',optimizer='Adam',metrics=['accuracy'])
```

11 Train The Model

[ ]:
```
model.
 ↪fit(x_train,y_train,validation_data=(x_test,y_test),epochs=5,batch_size=32)
```

```
Epoch 1/5
1875/1875 [==============================] - 16s 4ms/step - loss: 0.2478 -
accuracy: 0.9504 - val_loss: 0.0881 - val_accuracy: 0.9750
Epoch 2/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0682 -
accuracy: 0.9788 - val_loss: 0.0758 - val_accuracy: 0.9779
Epoch 3/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0469 -
accuracy: 0.9851 - val_loss: 0.0791 - val_accuracy: 0.9797
Epoch 4/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0343 -
accuracy: 0.9896 - val_loss: 0.0874 - val_accuracy: 0.9783
Epoch 5/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0276 -
accuracy: 0.9912 - val_loss: 0.0923 - val_accuracy: 0.9797
```

[ ]:
```
<keras.callbacks.History at 0x7f81601b4d90>
```

12 Observing The Metrics

[ ]:
```
#Final Evaluation of the model
metrics = model.evaluate(x_test,y_test,verbose=0)
print("Metrics(Test loss & Test Accuracy):")
print(metrics)
```

```
Metrics(Test loss & Test Accuracy):
[0.09231072664260864, 0.9797000288963318]
```

13 Test The Model

[ ]:
```
prediction = model.predict(x_test[:4])
print(prediction)
```

```
1/1 [==============================] - 0s 114ms/step
[[1.8438075e-13 7.5036949e-16 2.6225452e-12 8.2185430e-13 1.2639869e-17
  5.8749660e-19 3.3369569e-22 1.0000000e+00 2.5833825e-13 8.3934092e-12]
 [6.7929811e-08 2.0299848e-09 9.9999845e-01 8.5986706e-08 1.4513716e-12
  8.8498392e-15 1.1440226e-07 3.2568449e-13 1.2355670e-06 1.4316574e-13]
 [8.4147281e-12 1.0000000e+00 1.3705131e-08 1.2463309e-14 3.5282783e-09
  1.0625666e-09 2.3752877e-10 1.8354391e-09 1.0467741e-08 4.7241001e-15]
 [1.0000000e+00 1.4227057e-18 3.7752104e-09 5.1879172e-14 8.8747482e-12
  2.1503955e-11 5.8140831e-10 7.4268670e-15 1.8976434e-09 2.7798863e-10]]
```

[ ]:
```
import numpy as np
#Printing our labels from first 4 images
print(np.argmax(prediction,axis=1))
#printing the actual label
print(y_test[:4])
```

```
[7 2 1 0]
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

14 Saving the model

[ ]:
```
#Save the model
model.save('models/mnistCNN.h5')
```

15 Test with Saved Model

[2]:
```
!unzip '/content/data.zip'
```

```
Archive:  /content/data.zip
   creating: data/
 extracting: data/1.png
 extracting: data/2.png
 extracting: data/8.png
 extracting: data/4.png
 extracting: data/3.png
 extracting: data/6.png
 extracting: data/7.png
 extracting: data/0.png
 extracting: data/5.png
```

[4]:
```
from tensorflow.keras.models import load_model
model = load_model(r'/content/mnistCNN.h5')
from PIL import Image
import numpy as np
for index in range(4):
  img = Image.open('/content/data/'+str(index)+'.png').convert("L")
  img = img.resize((28,28))
  im2arr = np.array(img)
  im2arr = im2arr.reshape(1,28,28,1)
  y_pred = model.predict(im2arr)
  print(y_pred)
```

```
1/1 [==============================] - 0s 48ms/step
[[0.0972627  0.10735983 0.09861368 0.09531128 0.09601787 0.09924947
  0.09819323 0.10249204 0.10377011 0.10172982]]
1/1 [==============================] - 0s 16ms/step
[[0.0972627  0.10735983 0.09861368 0.09531128 0.09601787 0.09924947
  0.09819323 0.10249204 0.10377011 0.10172982]]
1/1 [==============================] - 0s 18ms/step
[[0.0972627  0.10735983 0.09861368 0.09531128 0.09601787 0.09924947
  0.09819323 0.10249204 0.10377011 0.10172982]]
1/1 [==============================] - 0s 14ms/step
[[0.0972627  0.10735983 0.09861368 0.09531128 0.09601787 0.09924947
  0.09819323 0.10249204 0.10377011 0.10172982]]
```