VIDYA PAYGUDE

Data Science Intern at LetsGrowMore Virtual Internship Program (APRIL-2022)

ADVANCED LEVEL TASK 7 - Develop A Neural Network That Can Read Handwriting

In [ ]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Sequential
from tensorflow.keras.utils import plot_model
from tensorflow.keras.layers import *
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from keras.datasets import mnist
import random
from numpy import argmax
```

```
Using TensorFlow backend.
```

In [ ]:
```
(x_train,y_train),(x_test,y_test) = mnist.load_data()
```

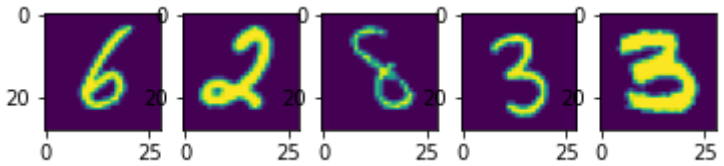
```
Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 7s 1us/step
```

60,000 images as input data with shape 28 x 28 each

In [ ]:
```
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

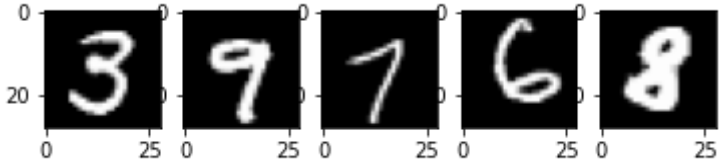
Out[ ]:
```
((60000, 28, 28), (60000,), (10000, 28, 28), (10000,))
```

In [ ]:
```
# sample images
for i in range (1,6):
    plt.subplot(1,5,i)
    plt.imshow(x_train[random.randint(0,5000)])
```



In [ ]:
```
x_train = x_train.reshape( (x_train.shape[0] , x_train.shape[1] , x_train.shape[2] , 1) )
x_test = x_test.reshape( (x_test.shape[0] , x_test.shape[1] , x_test.shape[2] , 1) )
x_train =  x_train.astype('float32') / 255.0
x_test =  x_test.astype('float32') / 255.0
```

In [ ]:
```
for i in range (1,6):
    plt.subplot(1,5,i)
    plt.imshow(x_train[random.randint(0,5000)] , cmap ="gray")
```



In [ ]:
```
model= Sequential()
model.add(Conv2D(32, (3,3) , activation='relu' , input_shape=(28,28,1)))
model.add(MaxPool2D((2,2)))
model.add(Conv2D(48, (3,3) , activation='relu' ))
model.add(MaxPool2D((2,2)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(500,activation='relu' ))
model.add(Dense(10,activation='softmax' ))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
```

```
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 48)        13872
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 48)          0
_________________________________________________________________
dropout (Dropout)            (None, 5, 5, 48)          0
_________________________________________________________________
flatten (Flatten)            (None, 1200)              0
_________________________________________________________________
dense (Dense)                (None, 500)               600500
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5010
=================================================================
Total params: 619,702
Trainable params: 619,702
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10, verbose =1 , batch_size=128, validation_split=0.1)
```

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/10
54000/54000 [==============================] - 28s 513us/sample - loss: 0.2437 - accuracy: 0.9256 - val_loss: 0.0584 - val_accurac
y: 0.9823
Epoch 2/10
54000/54000 [==============================] - 35s 647us/sample - loss: 0.0821 - accuracy: 0.9747 - val_loss: 0.0431 - val_accurac
y: 0.9875
Epoch 3/10
54000/54000 [==============================] - 31s 574us/sample - loss: 0.0621 - accuracy: 0.9811 - val_loss: 0.0359 - val_accurac
y: 0.9907
Epoch 4/10
54000/54000 [==============================] - 30s 564us/sample - loss: 0.0504 - accuracy: 0.9841 - val_loss: 0.0349 - val_accurac
y: 0.9900
Epoch 5/10
54000/54000 [==============================] - 28s 511us/sample - loss: 0.0428 - accuracy: 0.9859 - val_loss: 0.0267 - val_accurac
y: 0.9928
Epoch 6/10
54000/54000 [==============================] - 28s 518us/sample - loss: 0.0372 - accuracy: 0.9877 - val_loss: 0.0300 - val_accurac
y: 0.9910
Epoch 7/10
54000/54000 [==============================] - 31s 567us/sample - loss: 0.0340 - accuracy: 0.9893 - val_loss: 0.0247 - val_accurac
y: 0.9933
```

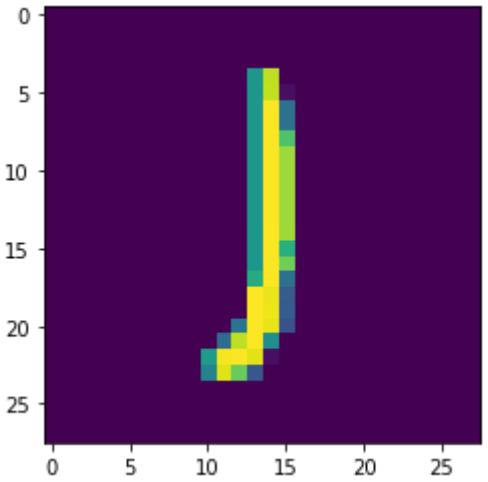
```
Epoch 8/10
54000/54000 [==============================] - 33s 608us/sample - loss: 0.0269 - accuracy: 0.9907 - val_loss: 0.0245 - val_accurac
y: 0.9940
Epoch 9/10
54000/54000 [==============================] - 29s 534us/sample - loss: 0.0259 - accuracy: 0.9918 - val_loss: 0.0251 - val_accurac
y: 0.9927
Epoch 10/10
54000/54000 [==============================] - 31s 566us/sample - loss: 0.0236 - accuracy: 0.9922 - val_loss: 0.0274 - val_accurac
y: 0.9922
```

Out[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x22d6f03a748>
```

In [ ]:
```
loss,acc = model.evaluate(x_test,y_test,verbose=0)
print("accuracy = ",acc*100, "%")
```

```
accuracy =  99.09999966621399 %
```

In [ ]:
```
n = random.randint(0,5000)
test_img = x_train[n]
plt.imshow(test_img)
plt.show()
```



In [ ]:
```
test_img = test_img.reshape(1,28,28,1 )

p = model.predict([test_img])
print("prediction = {} ".format(argmax(p)))
```

```
prediction = 1
```