Handwritten Digit Recognition using Python

Import the libraries and load the dataset

In [2]:
```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

print(x_train.shape, y_train.shape)
```

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

Preprocess the data

In [4]:
```
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
input_shape = (28, 28, 1)

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train)
y_test = keras.utils.to_categorical(y_test)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

Create the Model

In [5]:
```
batch_size = 128
num_classes = 10
epochs = 10

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,optimizer=keras.optimizers.Ad
```

Train the model

In [6]:
```
hist = model.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,verbose=1,valida
print("The model has successfully trained")

model.save('mnist.h5')
print("Saving the model as mnist.h5")
```

```
Epoch 1/10
469/469 [==============================] - 134s 279ms/step - loss: 2.2608 - accuracy:
0.1652 - val_loss: 2.1833 - val_accuracy: 0.4514
Epoch 2/10
469/469 [==============================] - 126s 268ms/step - loss: 2.1409 - accuracy:
0.3403 - val_loss: 2.0373 - val_accuracy: 0.6663
Epoch 3/10
469/469 [==============================] - 126s 268ms/step - loss: 1.9880 - accuracy:
0.4845 - val_loss: 1.8437 - val_accuracy: 0.7367
Epoch 4/10
469/469 [==============================] - 125s 267ms/step - loss: 1.7914 - accuracy:
0.5745 - val_loss: 1.6024 - val_accuracy: 0.7747
Epoch 5/10
469/469 [==============================] - 147s 314ms/step - loss: 1.5654 - accuracy:
0.6325 - val_loss: 1.3430 - val_accuracy: 0.7949
Epoch 6/10
469/469 [==============================] - 147s 313ms/step - loss: 1.3487 - accuracy:
0.6668 - val_loss: 1.1094 - val_accuracy: 0.8138
Epoch 7/10
469/469 [==============================] - 155s 331ms/step - loss: 1.1638 - accuracy:
0.6949 - val_loss: 0.9265 - val_accuracy: 0.8276
Epoch 8/10
469/469 [==============================] - 144s 306ms/step - loss: 1.0251 - accuracy:
0.7181 - val_loss: 0.7933 - val_accuracy: 0.8384
Epoch 9/10
469/469 [==============================] - 130s 276ms/step - loss: 0.9208 - accuracy:
0.7384 - val_loss: 0.6966 - val_accuracy: 0.8478
Epoch 10/10
469/469 [==============================] - 129s 275ms/step - loss: 0.8421 - accuracy:
0.7564 - val_loss: 0.6261 - val_accuracy: 0.8570
The model has successfully trained
Saving the model as mnist.h5
```

Evaluate the model

In [7]:
```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.6260712146759033
Test accuracy: 0.8569999933242798
```

In [ ]: