In [14]:

```
#import pandas as pd
import keras # to implement deep learning & high level nueral network
from keras.datasets import mnist # Modified National Institute of Standards and Techno
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten # dense to layer
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
import numpy as np


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

print(x_train.shape, y_train.shape) # training img 60,000 28,28
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1) #img size 28*28
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1) #reshape data
input_shape = (28, 28, 1)


# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, 10) #to convert matrices
y_test = keras.utils.to_categorical(y_test, 10)

x_train = x_train.astype('float32') #data type to divided 255
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')

batch_size = 128
num_classes = 10
epochs = 10
# architcutre for deep learning
model = Sequential() # sequential model is a linear stack of layers
model.add(Conv2D(32, kernel_size=(5, 5),activation='relu',input_shape=input_shape)) #
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu')) # output = 128
model.add(Dropout(0.3))
model.add(Dense(64, 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

hist = model.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,verbose=1,valida
print("The model has successfully trained")

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

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

```
(60000, 28, 28) (60000,)
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
Epoch 1/10
469/469 [==============================] - 14s 28ms/step - loss: 2.3007 - accuracy:
0.1175 - val_loss: 2.2904 - val_accuracy: 0.1632
Epoch 2/10
469/469 [==============================] - 11s 24ms/step - loss: 2.2889 - accuracy:
0.1441 - val_loss: 2.2789 - val_accuracy: 0.2701
Epoch 3/10
469/469 [==============================] - 12s 26ms/step - loss: 2.2791 - accuracy:
0.1661 - val_loss: 2.2677 - val_accuracy: 0.3729
Epoch 4/10
469/469 [==============================] - 12s 25ms/step - loss: 2.2690 - accuracy:
0.1891 - val_loss: 2.2556 - val_accuracy: 0.4569
Epoch 5/10
469/469 [==============================] - 12s 26ms/step - loss: 2.2574 - accuracy:
0.2132 - val_loss: 2.2417 - val_accuracy: 0.5188
Epoch 6/10
469/469 [==============================] - 12s 26ms/step - loss: 2.2449 - accuracy:
0.2340 - val_loss: 2.2262 - val_accuracy: 0.5498
Epoch 7/10
469/469 [==============================] - 11s 24ms/step - loss: 2.2308 - accuracy:
0.2523 - val_loss: 2.2086 - val_accuracy: 0.5683
Epoch 8/10
469/469 [==============================] - 12s 26ms/step - loss: 2.2151 - accuracy:
0.2693 - val_loss: 2.1887 - val_accuracy: 0.5781
Epoch 9/10
469/469 [==============================] - 14s 29ms/step - loss: 2.1981 - accuracy:
0.2836 - val_loss: 2.1660 - val_accuracy: 0.5890
Epoch 10/10
469/469 [==============================] - 14s 30ms/step - loss: 2.1784 - accuracy:
0.3018 - val_loss: 2.1397 - val_accuracy: 0.5997
The model has successfully trained
313/313 [==============================] - 1s 5ms/step - loss: 2.1397 - accuracy: 0.5
997
Test loss: 2.139695644378662
Test accuracy: 0.5996999740600586
Saving the model as mnist.h5
```

In [15]:

```
#GUI_digit_recognizer
from keras.models import load_model
from tkinter import *
import tkinter as tk
import win32gui # to img
from PIL import ImageGrab, Image
import numpy as np
```

In [17]:

```
model = load_model('mnist.h5') #loading to width i stored before

def predict_digit(img):
    #resize image to 28x28 pixels
    img = img.resize((28,28))
    #convert rgb to grayscale
    img = img.convert('L')
    img = np.array(img)
    #reshaping to support our model input and normalizing
    img = img.reshape(1,28,28,1)
    img = img/255.0
```

```
    #predicting the class
    res = model.predict([img])[0]
    return np.argmax(res), max(res) # argmax to get digit , max to get acuraccy


class App(tk.Tk):
    def __init__(self):
        tk.Tk.__init__(self)

        self.x = self.y = 0

        # Creating elements
        self.canvas = tk.Canvas(self, width=300, height=300, bg = "white", cursor="cro
        self.label = tk.Label(self, text="Draw..", font=("Helvetica", 48))
        self.classify_btn = tk.Button(self, text = "Recognise", command = self.classif
        self.button_clear = tk.Button(self, text = "Clear", command = self.clear_all)

        # Grid structure
        self.canvas.grid(row=0, column=0, pady=2, sticky=W, )
        self.label.grid(row=0, column=1,pady=2, padx=2)
        self.classify_btn.grid(row=1, column=1, pady=2, padx=2)
        self.button_clear.grid(row=1, column=0, pady=2)

        #self.canvas.bind("<Motion>", self.start_pos)
        self.canvas.bind("<B1-Motion>", self.draw_lines) #B1- to click or not click wh

    def clear_all(self):
        self.canvas.delete("all") # to clear

    def classify_handwriting(self):
        HWND = self.canvas.winfo_id()  # get the handle of the canvas
        rect = win32gui.GetWindowRect(HWND)  # get the coordinate of the canvas
        a,b,c,d = rect
        rect=(a+4,b+4,c-4,d-4)
        im = ImageGrab.grab(rect)

        digit, acc = predict_digit(im) # to get acuraccy and digit
        self.label.configure(text= str(digit)+', '+ str(int(acc*100))+'%')#to get acur

    def draw_lines(self, event): # when to write digit go on my way
        self.x = event.x
        self.y = event.y
        r=8 # font size
        self.canvas.create_oval(self.x-r, self.y-r, self.x + r, self.y+ r, fill='black

app = App()
mainloop()
```

```
1/1 [==============================] - 0s 88ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
```