In [*]:
```
pip install numpy tensorflow keras pillow
```

In [11]:
```
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,)
```

In [12]:
```
import keras
from keras.utils import to_categorical

# Define the number of classes
num_classes = 10  # Adjust this based on your dataset

# Rest of your code
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)
y_train = to_categorical(y_train, num_classes)
y_test = to_categorical(y_test, num_classes)
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
```

In [13]:
```
batch_size = 128
num_classes = 10
epochs = 10

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',input_shape=input_sh
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.optim
```

In [13]:
```
hist = model.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,verbose=
print("The model has successfully trained")

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

```
Epoch 1/10
469/469 [==============================] - 109s 228ms/step - loss: 2.2775 - a
ccuracy: 0.1511 - val_loss: 2.2197 - val_accuracy: 0.3650
Epoch 2/10
469/469 [==============================] - 112s 239ms/step - loss: 2.1837 - a
ccuracy: 0.3010 - val_loss: 2.1033 - val_accuracy: 0.5870
Epoch 3/10
469/469 [==============================] - 105s 223ms/step - loss: 2.0614 - a
ccuracy: 0.4453 - val_loss: 1.9471 - val_accuracy: 0.6706
Epoch 4/10
469/469 [==============================] - 105s 223ms/step - loss: 1.8968 - a
ccuracy: 0.5436 - val_loss: 1.7416 - val_accuracy: 0.7184
Epoch 5/10
469/469 [==============================] - 104s 222ms/step - loss: 1.6920 - a
ccuracy: 0.6048 - val_loss: 1.4949 - val_accuracy: 0.7526
Epoch 6/10
469/469 [==============================] - 104s 222ms/step - loss: 1.4677 - a
ccuracy: 0.6467 - val_loss: 1.2437 - val_accuracy: 0.7815
Epoch 7/10
469/469 [==============================] - 104s 222ms/step - loss: 1.2615 - a
ccuracy: 0.6804 - val_loss: 1.0295 - val_accuracy: 0.8025
Epoch 8/10
469/469 [==============================] - 104s 221ms/step - loss: 1.0995 - a
ccuracy: 0.7058 - val_loss: 0.8670 - val_accuracy: 0.8199
Epoch 9/10
469/469 [==============================] - 104s 222ms/step - loss: 0.9754 - a
ccuracy: 0.7252 - val_loss: 0.7512 - val_accuracy: 0.8311
Epoch 10/10
469/469 [==============================] - 105s 224ms/step - loss: 0.8880 - a
ccuracy: 0.7418 - val_loss: 0.6691 - val_accuracy: 0.8433
The model has successfully trained
Saving the model as mnist.h5
```

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

In [ ]:

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

model = load_model('mnist.h5')

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)

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", cur
        self.label = tk.Label(self, text="Thinking..", font=("Helvetica", 48))
        self.classify_btn = tk.Button(self, text = "Recognise", command =
        self.button_clear = tk.Button(self, text = "Clear", command = self.cle

        # 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)

    def clear_all(self):
        self.canvas.delete("all")

    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
        im = ImageGrab.grab(rect)

        digit, acc = predict_digit(im)
        self.label.configure(text= str(digit)+', '+ str(int(acc*100))+'%')

    def draw_lines(self, event):
        self.x = event.x
```

```
        self.y = event.y
        r=8
        self.canvas.create_oval(self.x-r, self.y-r, self.x + r, self.y + r, fi

app = App()
mainloop()
```

In [ ]: