from keras.models import load\_model  
from tkinter import \*  
import tkinter as tk  
import win32gui  
import os  
import cv2  
from PIL import ImageGrab, Image  
import numpy as np  
  
model = load\_model('mnist.h5')  
print('Trained weights loaded')  
  
def get\_handle():  
 """This function uses the wingui library to get the window handles of all the active windows.  
 Then, the window with the name as 'tk' is selected and its handle is returned."""  
 toplist = []  
 windows\_list = []  
 canvas = 0  
 def enum\_win(hwnd, result):  
 win\_text = win32gui.GetWindowText(hwnd)  
 windows\_list.append((hwnd, win\_text))  
 win32gui.EnumWindows(enum\_win, toplist)  
 for (hwnd, win\_text) in windows\_list:  
 if 'tk' == win\_text:  
 canvas = hwnd  
 return canvas  
  
def preprocessing\_image():  
 """function to preprocess the image to"""  
 image = cv2.imread('test.jpg')  
 grey = cv2.cvtColor(image.copy(), cv2.COLOR\_BGR2GRAY)  
 ret, thresh = cv2.threshold(grey.copy(), 75, 255, cv2.THRESH\_BINARY\_INV)  
 contours, \_ = cv2.findContours(thresh, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 cv2.drawContours(image, contours, -1, (0, 255, 0), 3)  
 for c in contours:  
 x,y,w,h = cv2.boundingRect(c)  
 # Creating a rectangle around the digit in the original image (for displaying the digits fetched via contours)  
 cv2.rectangle(image, (x,y), (x+w, y+h), color=(0, 255, 0), thickness=2)  
 # Cropping out the digit from the image corresponding to the current contours in the for loop  
 digit = thresh[y:y+h, x:x+w]  
 # Resizing that digit to (18, 18)  
 resized\_digit = cv2.resize(digit, (18,18))  
 # Padding the digit with 5 pixels of black color (zeros) in each side to finally produce the image of (28, 28)  
 padded\_digit = np.pad(resized\_digit, ((5,5),(5,5)), "constant", constant\_values=0)  
 # Adding the preprocessed digit to the list of preprocessed digits  
 preprocessed\_digit = (padded\_digit)  
 return preprocessed\_digit  
  
def predict\_digit(img):  
 """function to predict the digit.  
 Argument of function is PIL Image"""  
 img.save('test.jpg')  
 preprocessed\_image = preprocessing\_image()  
 img = preprocessed\_image.reshape(1, 28, 28, 1)  
 img = img/255.0  
 #predicting the digit  
 result = model.predict([img])[0]  
 os.remove('test.jpg')  
 return np.argmax(result), max(result)  
  
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="cross")  
 self.label = tk.Label(self, text="Thinking..", font=("Helvetica", 48))  
 self.classify\_btn = tk.Button(self, text = "Recognise", command = self.classify\_handwriting)  
 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)  
 def clear\_all(self):  
 self.canvas.delete("all")  
 def classify\_handwriting(self):  
 HWND = self.canvas.winfo\_id() # get the handle of the canvas  
 hwnd = get\_handle()  
 rect = win32gui.GetWindowRect(HWND) # get the coordinate of the canvas  
 x1, y1, x2, y2 = rect  
 # print(x1,x2, y1,y2)  
 im = ImageGrab.grab((x1+40, y1+40, x2+100, y2+100))  
 digit, acc = predict\_digit(im)  
 print(digit)  
 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, fill='black')  
  
app = App()  
mainloop()