

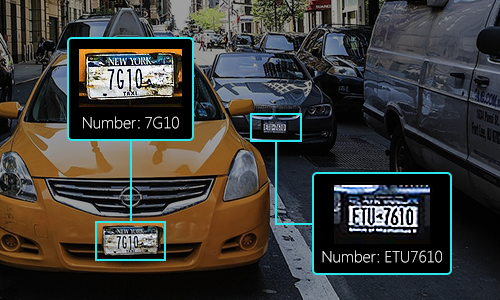
**REPORT**

|  |  |
| --- | --- |
| **STUDENT NAME** | **: NURUL YASMINE BINTI MOHD YAZID YAP** |
| **MATRIC NUMBER** | **: BS18110359** |
| **FACULTY** | **: SCIENCE AND NATURAL RESOURCES** |
| **COURSE** | **: MATHEMATICS WITH COMPUTER GRAPHICS** |
| **LECTURER** | **: PROF. MADYA DR. ABDULLAH BIN BADE** |
| **SUBJECT** | **: SC32303 FUNDAMENTAL OF COMPUTER GRAPHICS** |
| **ASSIGNMENT TITLE** | **: CAR PLATE DETECTOR** |
| **DATE OF SUBMISSION** | **: 6/1/2020** |
| **SIGNED:** |  |

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| 1.INTRODUCTION | 3 |
| 2.FRAMEWORK AND ARCHITECTURE | 3 |
| 3.OUTPUT | 4 |
| 4.ADVANTAGES AND UNIQUENESS | 5 |
| 6.CONCLUSION | 5 |
| 7.SOURCE CODE | 6 |

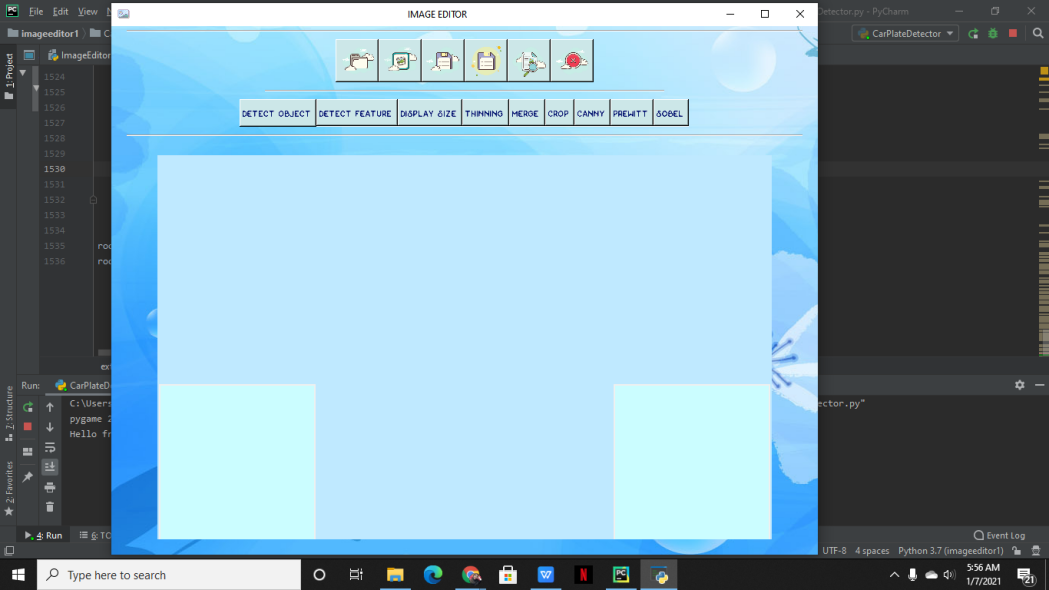
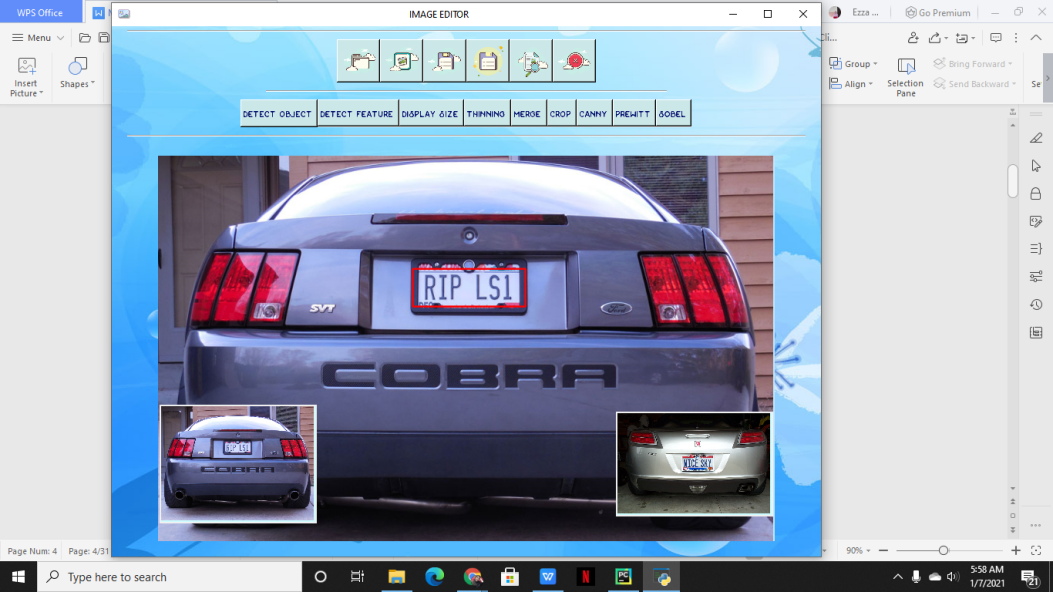
**INTRODUCTION**

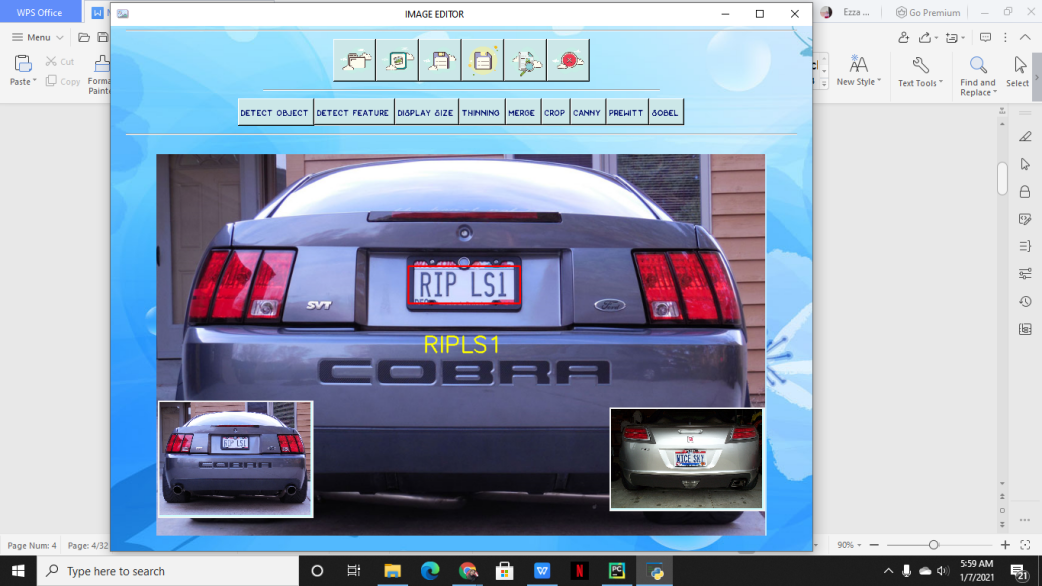
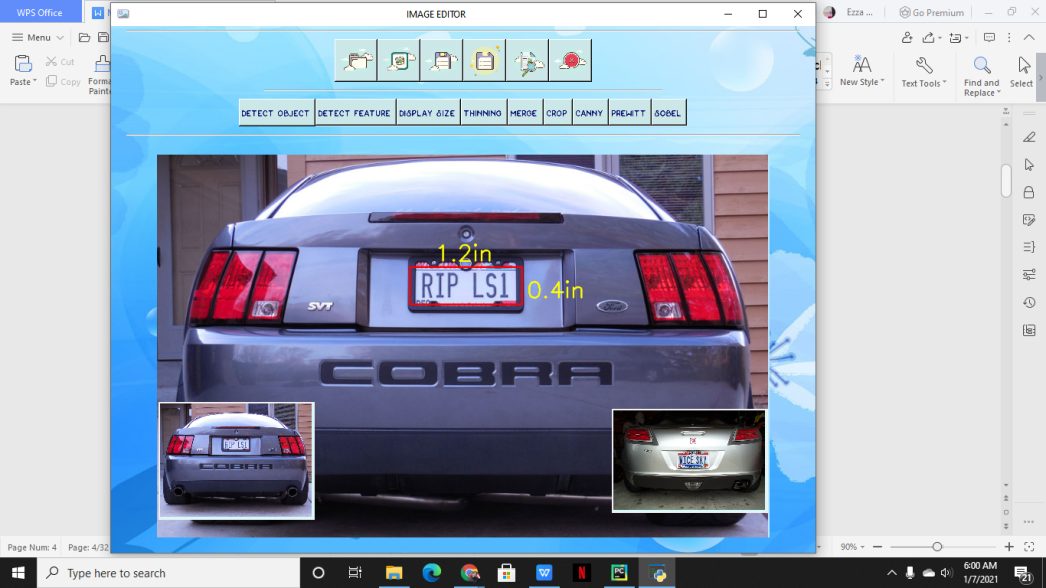
Car Plate Detector is written using basic opencv ,tkinter and other techniques in Digital Image Processing field of study. The concept is to build and develop a one-click solution object detection application. The application should be able to find the width,height and size of detected objects. The application should be also able to detect one character from the detected object, compare size of each detected object, apply thinning techniques and replace canny edge dtection techniques with prewitt and sobel. Thus, the theme used is a simple car plate detection application. The pictures below is two example of car plate detection application. 

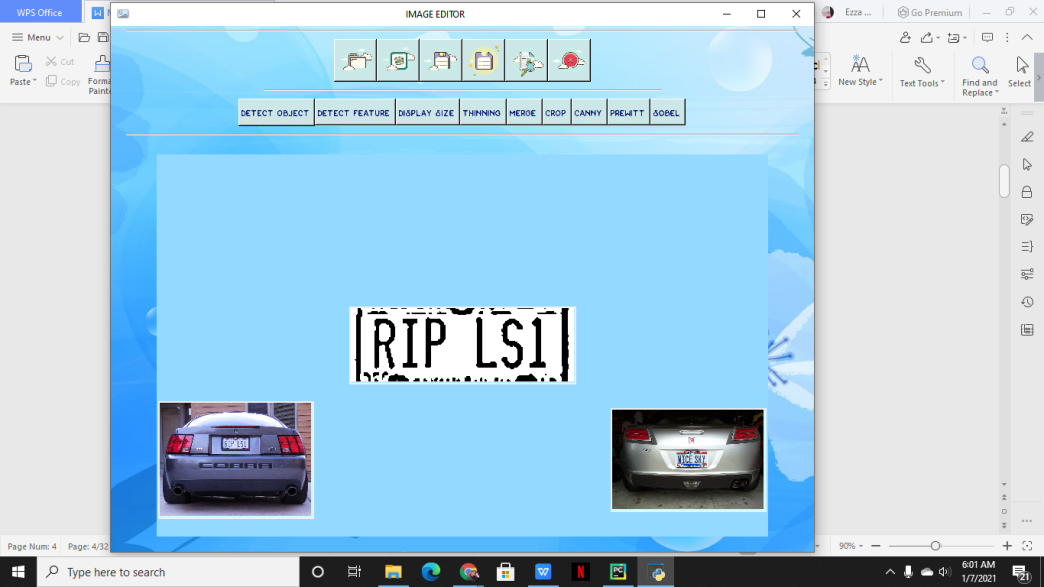
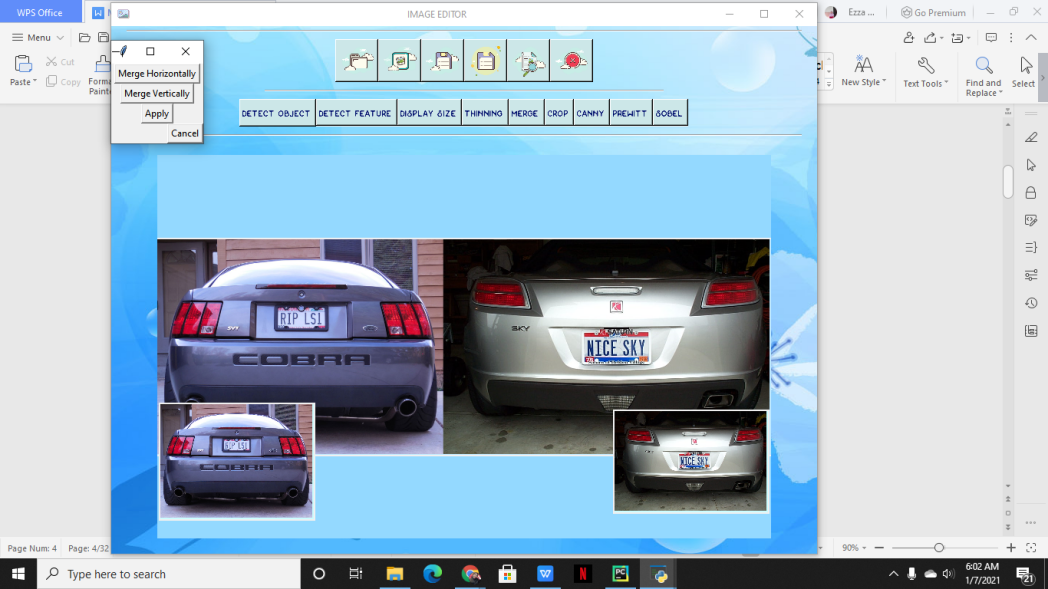
**FRAMEWORK AND ARCHITECTURE**

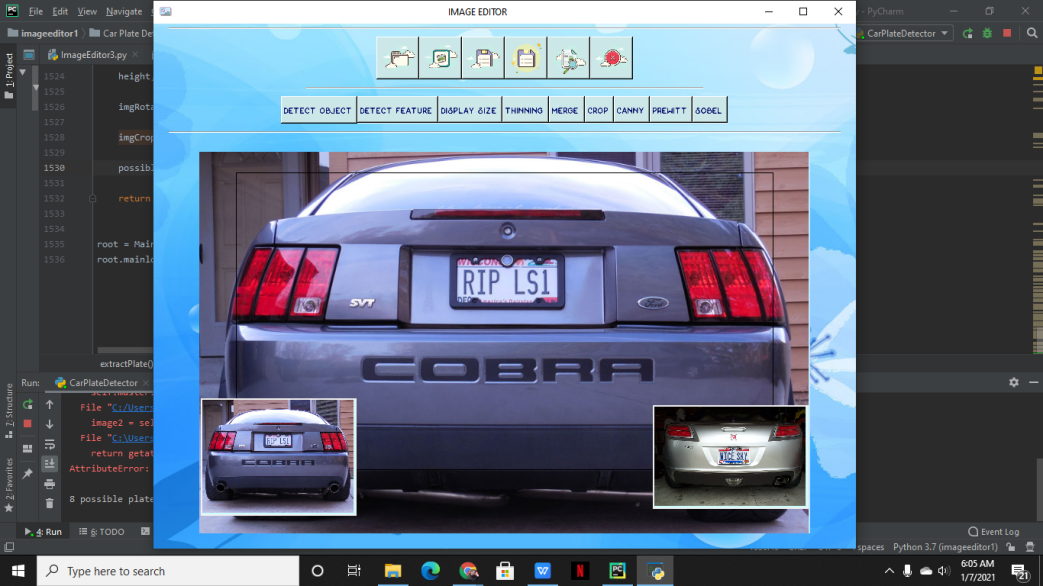
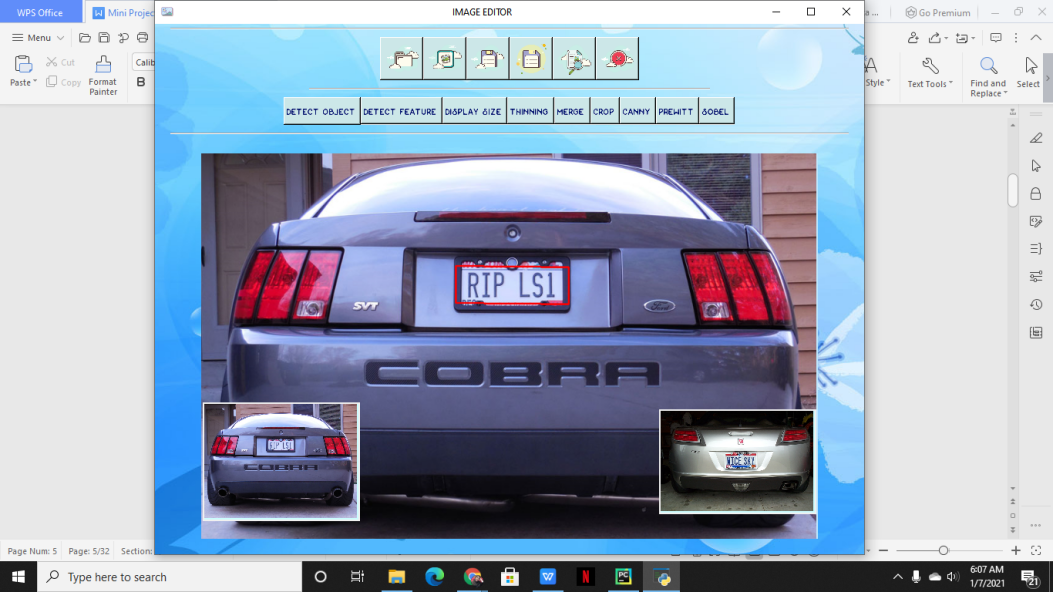
The application require user to insert image from computer to start using the button from the edit bar. The edit bar comprises of New, 2nd Image, Save, Save As, Save As Type, Clear, Detect Object, Detect Feature, Display Size, Thinning, Crop, Merge, Canny, Prewitt, and Sobel button. The New, Save, Save As and Clear button function to load image from file in computer , save the image during editing, save the image in the computer after editing and revert the image to its original image respectively. After inserting the image through New button, the image will resize according to window application size. Image can be save as four types of image file using the Save As Types button which consist of Bitmaps, JPEG, PNG and TIFF file image. User can detect object using four button which is Detect Object, Canny, Prewitt, and Sobel button. The Detect Object use threshold techniques to detect the car plate, the Canny, Prewitt and Sobel button uses Canny, Prewitt and Sobel techniques respectively to detect the car plate. Apply selecting either of the four button, the Detect Feature will show the text on plate, the Display Size will show the width and height of detected object and the Thinning button will show the detected object after thinning. User also able to crop image using the Crop button using mouse motion. The image can be merge using the Merge button. The button comprises horizontal and vertical merge. There is also Image Viewer to enable the image to be viewed and updated when edited. A sound system is also used where there is background music using Pygame. The functionality of Opencv and Tkinter in this game covered many areas. Opencv is used to manipulate the image while tkinter is used to create the interface of the image editor.

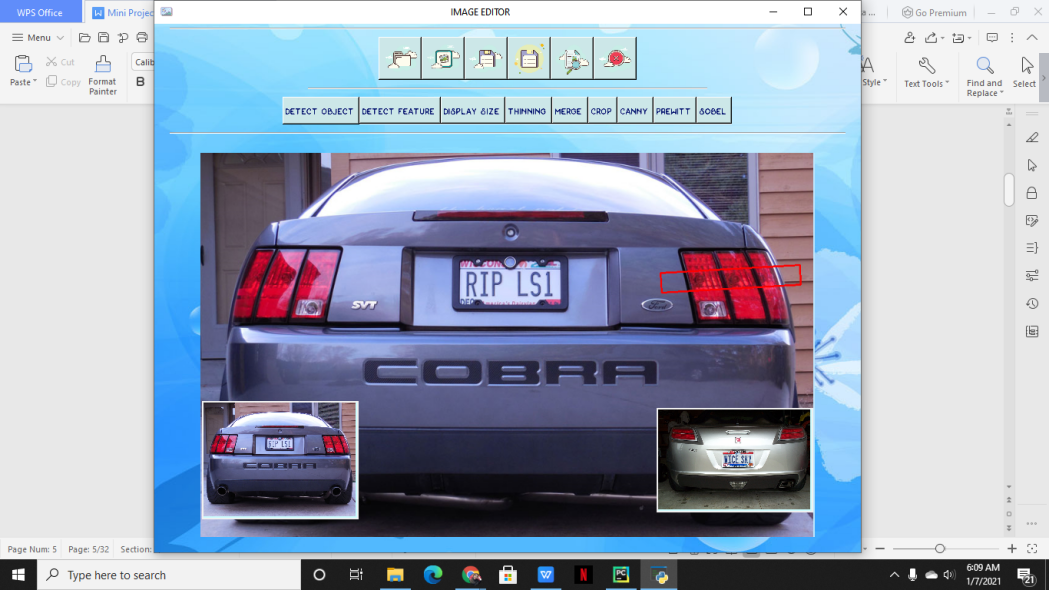
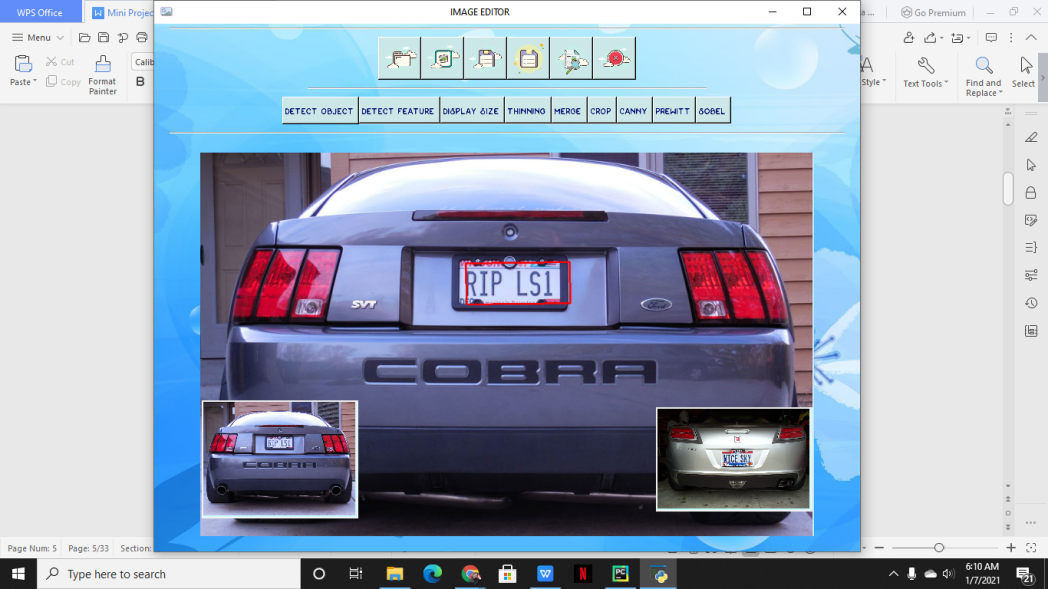
**OUTPUT**

**ADVANTAGES AND UNIQUENESS**

The advantages in application is that the user able to edit the image better using the mouse motion in the crop button. The editbar menu which has loading and save option allow user to choose a variety of image to edit and save the edited image in the computer. The advantages in this application is user able to choose to save different types of image file types. Other advantages is the detect feature, display size and thinning function can change according to the techniques used whether it is threshold,canny,prewitt or sobel. The uniqueness in this application compare to other is its cool space background,title, icon and button. The complementary of light blue colours can make the user more attracted to use the image editor. The other uniqueness in this image editor is its interactive and easy to use interface. This helps user in the process of editing the image. The slow background music enable the user to relax during editing process.

The image editor also have uniqueness in its clear option. This enable the user to reverse to its original image or permanent edited image. Two canvas of original image is shown at bottom to help user identify the difference in the after and before edition. The second canvas also enable the user to see the second image loaded.

**CONCLUSION**

In conclusion,the uses of opencv held many advantages in digital image processing development. The simplicity of application of opencv and its variety of uses in graphics attract many programmers and image editor programmer. In the process of creating this image editor, I learn to use opencv to manipulate image. This widen my knowledge in programming and enable me to use this knowledge in future use. The process of programming of this image editor also helps me to learn to create a better interface using tkinter. I learn to position my button and create a colorful background.

**SOURCE CODE**

import tkinter as tk  
from tkinter import ttk  
from tkinter import TOP, LEFT, W, X, YES , N, S, E, W ,NE, NW, SE, SW  
from tkinter import filedialog  
from tkinter import Toplevel, Button, RIGHT  
from tkinter import Frame, Canvas, CENTER  
from PIL import Image, ImageTk  
import pygame as pg  
import numpy as np  
import cv2  
import os  
import math  
import random  
  
SCALAR\_BLACK = (0.0, 0.0, 0.0)  
SCALAR\_WHITE = (255.0, 255.0, 255.0)  
SCALAR\_YELLOW = (0.0, 255.0, 255.0)  
SCALAR\_GREEN = (0.0, 255.0, 0.0)  
SCALAR\_RED = (0.0, 0.0, 255.0)  
GAUSSIAN\_SMOOTH\_FILTER\_SIZE = (5, 5)  
ADAPTIVE\_THRESH\_BLOCK\_SIZE = 19  
ADAPTIVE\_THRESH\_WEIGHT = 9  
kNearest = cv2.ml.KNearest\_create()  
MIN\_PIXEL\_WIDTH = 2  
MIN\_PIXEL\_HEIGHT = 8  
MIN\_ASPECT\_RATIO = 0.25  
MAX\_ASPECT\_RATIO = 1.0  
MIN\_PIXEL\_AREA = 80  
MIN\_DIAG\_SIZE\_MULTIPLE\_AWAY = 0.3  
MAX\_DIAG\_SIZE\_MULTIPLE\_AWAY = 5.0  
MAX\_CHANGE\_IN\_AREA = 0.5  
MAX\_CHANGE\_IN\_WIDTH = 0.8  
MAX\_CHANGE\_IN\_HEIGHT = 0.2  
MAX\_ANGLE\_BETWEEN\_CHARS = 12.0  
MIN\_NUMBER\_OF\_MATCHING\_CHARS = 3  
RESIZED\_CHAR\_IMAGE\_WIDTH = 20  
RESIZED\_CHAR\_IMAGE\_HEIGHT = 30  
MIN\_CONTOUR\_AREA = 100  
PLATE\_WIDTH\_PADDING\_FACTOR = 1.3  
PLATE\_HEIGHT\_PADDING\_FACTOR = 1.5  
choice = None  
  
class Main(tk.Tk):  
  
 def \_\_init\_\_(self):  
 tk.Tk.\_\_init\_\_(self)  
  
 pg.init()  
 pg.mixer.music.load('Still With You.wav')  
 pg.mixer.music.play(-1)  
 pg.mixer.music.set\_volume(.1)  
  
 self.filename = ""  
 self.original\_image = None  
 self.original2\_image = None  
 self.processed\_image = None  
 self.save\_file\_type\_frame = None  
 self.is\_image\_selected = False  
 self.is\_canny\_state = False  
 self.is\_prewitt\_state = False  
 self.is\_sobel\_state = False  
 self.is\_detectObj\_state = False  
 self.is\_thinning\_state = False  
 self.is\_detectFeature\_state = False  
 self.is\_size\_state = False  
 self.is\_crop\_state = False  
 self.merge\_frame = None  
  
 self.flip\_frame = None  
 self.rotate\_frame = None  
 self.resize\_frame = None  
 self.translate\_frame = None  
 self.color\_frame = None  
 self.adjust\_frame = None  
 self.filter\_frame = None  
 self.MergeSplit\_frame = None  
 self.segment\_frame = None  
 self.save\_as\_type\_frame = None  
  
 def center(e):  
 w = int(self.winfo\_width() / 3.5) # get root width and scale it ( in pixels )  
 s = 'IMAGE EDITOR'.rjust(w // 2)  
 self.title(s)  
  
 self.bind("<Configure>", center) # called when window resized  
 # self.title("Image Editor")  
 self.iconphoto(False, tk.PhotoImage(file='icon.png'))  
 # self.configure(bg="blue")  
 load = Image.open('bg5.jpg')  
 render = ImageTk.PhotoImage(load)  
  
 # labels can be text or images  
 background\_label = tk.Label(self, image=render)  
 background\_label.image = render  
 background\_label.place(x=0, y=0, relwidth=1, relheight=1)  
  
 self.editbar1 = EditBar1(master=self)  
 self.editbar2 = EditBar2(master=self)  
 separator = ttk.Separator(master=self, orient=tk.HORIZONTAL)  
 separator1 = ttk.Separator(master=self, orient=tk.HORIZONTAL)  
 separator2 = ttk.Separator(master=self, orient=tk.HORIZONTAL)  
 self.image\_viewer = ImageViewer(master=self)  
  
 separator.pack(fill=tk.X, padx=20, pady=5)  
 self.editbar1.pack(pady=5)  
 separator1.pack(fill=tk.X, padx=200, pady=5)  
 self.editbar2.pack(pady=5)  
 separator2.pack(fill=tk.X, padx=20, pady=5)  
 self.image\_viewer.pack(fill=tk.BOTH, padx=60, pady=20, expand=1)  
  
  
class EditBar1(Frame):  
  
 def \_\_init\_\_(self, master=None):  
 Frame.\_\_init\_\_(self, master=master)  
  
 newicon = tk.PhotoImage(file='new.png').subsample(4,4)  
 new2icon = tk.PhotoImage(file='new22.png').subsample(2,2)  
 saveicon = tk.PhotoImage(file='save.png').subsample(4,4)  
 saveasicon = tk.PhotoImage(file='save as.png').subsample(4,4)  
 saveastypeicon = tk.PhotoImage(file='save as type.png').subsample(4,4)  
 clearicon = tk.PhotoImage(file='clear.png').subsample(4,4)  
  
 self.new\_button = Button(self, image=newicon,bg='#cce7e8')  
 self.new2\_button = Button(self, image=new2icon,bg='#cce7e8')  
 self.save\_button = Button(self, image=saveicon,bg='#cce7e8')  
 self.save\_as\_button = Button(self, image=saveasicon,bg='#cce7e8')  
 self.save\_as\_type\_button = Button(self, image=saveastypeicon,bg='#cce7e8')  
 self.clear\_button = Button(self, image=clearicon,bg='#cce7e8')  
  
 self.new\_button.image = newicon  
 self.new2\_button.image = new2icon  
 self.save\_button.image = saveicon  
 self.save\_as\_button.image = saveasicon  
 self.save\_as\_type\_button.image = saveastypeicon  
 self.clear\_button.image = clearicon  
  
 self.new\_button.bind("<ButtonRelease>", self.new\_button\_released)  
 self.new2\_button.bind("<ButtonRelease>", self.new2\_button\_released)  
 self.save\_button.bind("<ButtonRelease>", self.save\_button\_released)  
 self.save\_as\_button.bind("<ButtonRelease>", self.save\_as\_button\_released)  
 self.save\_as\_type\_button.bind("<ButtonRelease>", self.save\_as\_type\_button\_released)  
 self.clear\_button.bind("<ButtonRelease>", self.clear\_button\_released)  
  
 self.new\_button.pack(side=LEFT)  
 self.new2\_button.pack(side=LEFT)  
 self.save\_button.pack(side=LEFT)  
 self.save\_as\_button.pack(side=LEFT)  
 self.save\_as\_type\_button.pack(side=LEFT)  
 self.clear\_button.pack()  
  
 def new\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.new\_button:  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_canny\_state:  
 self.master.image\_viewer.deactivate\_canny()  
 if self.master.is\_prewitt\_state:  
 self.master.image\_viewer.deactivate\_prewitt()  
 if self.master.is\_sobel\_state:  
 self.master.image\_viewer.deactivate\_sobel()  
 if self.master.is\_thinning\_state:  
 self.master.image\_viewer.deactivate\_thinning()  
 if self.master.is\_detectFeature\_state:  
 self.master.image\_viewer.deactivate\_detectFeature()  
 if self.master.is\_size\_state:  
 self.master.image\_viewer.deactivate\_size()  
 if self.master.is\_detectObj\_state:  
 self.master.image\_viewer.deactivate\_detectObj()  
  
  
 filename = filedialog.askopenfilename()  
 image = cv2.imread(filename)  
  
 if image is not None:  
 self.master.filename = filename  
 self.master.original\_image = image.copy()  
 self.master.processed\_image = image.copy()  
 self.master.image\_viewer.show\_image()  
 self.master.is\_image\_selected = True  
  
 def new2\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.new2\_button:  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_canny\_state:  
 self.master.image\_viewer.deactivate\_canny()  
 if self.master.is\_prewitt\_state:  
 self.master.image\_viewer.deactivate\_prewitt()  
 if self.master.is\_sobel\_state:  
 self.master.image\_viewer.deactivate\_sobel()  
 if self.master.is\_thinning\_state:  
 self.master.image\_viewer.deactivate\_thinning()  
 if self.master.is\_detectFeature\_state:  
 self.master.image\_viewer.deactivate\_detectFeature()  
 if self.master.is\_size\_state:  
 self.master.image\_viewer.deactivate\_size()  
 if self.master.is\_detectObj\_state:  
 self.master.image\_viewer.deactivate\_detectObj()  
  
 filename = filedialog.askopenfilename()  
 image2 = cv2.imread(filename)  
  
 if image2 is not None:  
 self.master.filename = filename  
 self.master.original2\_image = image2.copy()  
 self.master.processed2\_image = image2.copy()  
 self.master.image\_viewer.show\_image()  
 self.master.is\_image\_selected = True  
  
 def save\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.save\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_canny\_state:  
 self.master.image\_viewer.deactivate\_canny()  
 if self.master.is\_prewitt\_state:  
 self.master.image\_viewer.deactivate\_prewitt()  
 if self.master.is\_sobel\_state:  
 self.master.image\_viewer.deactivate\_sobel()  
 if self.master.is\_thinning\_state:  
 self.master.image\_viewer.deactivate\_thinning()  
 if self.master.is\_detectFeature\_state:  
 self.master.image\_viewer.deactivate\_detectFeature()  
 if self.master.is\_size\_state:  
 self.master.image\_viewer.deactivate\_size()  
 if self.master.is\_detectObj\_state:  
 self.master.image\_viewer.deactivate\_detectObj()  
  
 save\_image = self.master.processed\_image  
 image\_filename = self.master.filename  
 cv2.imwrite(image\_filename, save\_image)  
  
 def save\_as\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.save\_as\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_canny\_state:  
 self.master.image\_viewer.deactivate\_canny()  
 if self.master.is\_prewitt\_state:  
 self.master.image\_viewer.deactivate\_prewitt()  
 if self.master.is\_sobel\_state:  
 self.master.image\_viewer.deactivate\_sobel()  
 if self.master.is\_thinning\_state:  
 self.master.image\_viewer.deactivate\_thinning()  
 if self.master.is\_detectFeature\_state:  
 self.master.image\_viewer.deactivate\_detectFeature()  
 if self.master.is\_size\_state:  
 self.master.image\_viewer.deactivate\_size()  
 if self.master.is\_detectObj\_state:  
 self.master.image\_viewer.deactivate\_detectObj()  
  
 original\_file\_type = self.master.filename.split('.')[-1]  
 filename = filedialog.asksaveasfilename()  
 filename = filename + "." + original\_file\_type  
  
 save\_image = self.master.processed\_image  
 cv2.imwrite(filename, save\_image)  
  
 self.master.filename = filename  
  
 def save\_as\_type\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.save\_as\_type\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_canny\_state:  
 self.master.image\_viewer.deactivate\_canny()  
 if self.master.is\_prewitt\_state:  
 self.master.image\_viewer.deactivate\_prewitt()  
 if self.master.is\_sobel\_state:  
 self.master.image\_viewer.deactivate\_sobel()  
 if self.master.is\_thinning\_state:  
 self.master.image\_viewer.deactivate\_thinning()  
 if self.master.is\_detectFeature\_state:  
 self.master.image\_viewer.deactivate\_detectFeature()  
 if self.master.is\_size\_state:  
 self.master.image\_viewer.deactivate\_size()  
 if self.master.is\_detectObj\_state:  
 self.master.image\_viewer.deactivate\_detectObj()  
  
 self.master.save\_as\_type\_frame = FileTypeFrame(master=self.master)  
 self.master.save\_as\_type\_frame.grab\_set()  
  
 def clear\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.clear\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_canny\_state:  
 self.master.image\_viewer.deactivate\_canny()  
 if self.master.is\_prewitt\_state:  
 self.master.image\_viewer.deactivate\_prewitt()  
 if self.master.is\_sobel\_state:  
 self.master.image\_viewer.deactivate\_sobel()  
 if self.master.is\_thinning\_state:  
 self.master.image\_viewer.deactivate\_thinning()  
 if self.master.is\_detectFeature\_state:  
 self.master.image\_viewer.deactivate\_detectFeature()  
 if self.master.is\_size\_state:  
 self.master.image\_viewer.deactivate\_size()  
 if self.master.is\_detectObj\_state:  
 self.master.image\_viewer.deactivate\_detectObj()  
  
 self.master.processed\_image = self.master.original\_image.copy()  
 self.master.image\_viewer.show\_image()  
 self.master.processed2\_image = self.master.original2\_image.copy()  
 self.master.image\_viewer.show\_image()  
  
class EditBar2(Frame):  
  
 def \_\_init\_\_(self, master=None):  
 Frame.\_\_init\_\_(self, master=master)  
  
 detectObjicon = tk.PhotoImage(file='detect object.png').subsample(2,2)  
 detectFeatureicon = tk.PhotoImage(file='detect feature.png').subsample(2,2)  
 sizeicon = tk.PhotoImage(file='size.png').subsample(2,2)  
 thinningicon = tk.PhotoImage(file='thinning.png').subsample(2,2)  
 mergeicon = tk.PhotoImage(file='merge.png').subsample(2,2)  
 cropicon = tk.PhotoImage(file='crop.png').subsample(2,2)  
 cannyicon = tk.PhotoImage(file='canny.png').subsample(2,2)  
 prewitticon = tk.PhotoImage(file='prewitt.png').subsample(2,2)  
 sobelicon = tk.PhotoImage(file='sobel.png').subsample(2,2)  
  
 self.detectObj\_button = Button(self, image=detectObjicon,bg='#cce7e8')  
 self.detectFeature\_button = Button(self, image=detectFeatureicon,bg='#cce7e8')  
 self.size\_button = Button(self, image=sizeicon,bg='#cce7e8')  
 self.thinning\_button = Button(self, image=thinningicon,bg='#cce7e8')  
 self.merge\_button = Button(self, image=mergeicon,bg='#cce7e8')  
 self.crop\_button = Button(self, image=cropicon,bg='#cce7e8')  
 self.canny\_button = Button(self, image=cannyicon,bg='#cce7e8')  
 self.prewitt\_button = Button(self, image=prewitticon,bg='#cce7e8')  
 self.sobel\_button = Button(self, image=sobelicon,bg='#cce7e8')  
  
 self.detectObj\_button.image = detectObjicon  
 self.detectFeature\_button.image = detectFeatureicon  
 self.size\_button.image = sizeicon  
 self.thinning\_button.image = thinningicon  
 self.merge\_button.image = mergeicon  
 self.crop\_button.image = cropicon  
 self.canny\_button.image = cannyicon  
 self.prewitt\_button.image = prewitticon  
 self.sobel\_button.image = sobelicon  
  
 self.detectObj\_button.bind("<ButtonRelease>", self.detectObj\_button\_released)  
 self.detectFeature\_button.bind("<ButtonRelease>", self.detectFeature\_button\_released)  
 self.size\_button.bind("<ButtonRelease>", self.size\_button\_released)  
 self.thinning\_button.bind("<ButtonRelease>", self.thinning\_button\_released)  
 self.merge\_button.bind("<ButtonRelease>", self.merge\_button\_released)  
 self.crop\_button.bind("<ButtonRelease>", self.crop\_button\_released)  
 self.canny\_button.bind("<ButtonRelease>", self.canny\_button\_released)  
 self.prewitt\_button.bind("<ButtonRelease>", self.prewitt\_button\_released)  
 self.sobel\_button.bind("<ButtonRelease>", self.sobel\_button\_released)  
  
 self.detectObj\_button.pack(side=LEFT, anchor=W, fill=X, expand=YES)  
 self.detectFeature\_button.pack(side=LEFT, anchor=W, fill=X, expand=YES)  
 self.size\_button.pack(side=LEFT, anchor=W, fill=X, expand=YES)  
 self.thinning\_button.pack(side=LEFT, anchor=W, fill=X, expand=YES)  
 self.merge\_button.pack(side=LEFT, anchor=W, fill=X, expand=YES)  
 self.crop\_button.pack(side=LEFT, anchor=W, fill=X, expand=YES)  
 self.canny\_button.pack(side=LEFT, anchor=W, fill=X, expand=YES)  
 self.prewitt\_button.pack(side=LEFT, anchor=W, fill=X, expand=YES)  
 self.sobel\_button.pack(side=LEFT, anchor=W, fill=X, expand=YES)  
  
 def detectObj\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.detectObj\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_canny\_state:  
 self.master.image\_viewer.deactivate\_canny()  
 if self.master.is\_prewitt\_state:  
 self.master.image\_viewer.deactivate\_prewitt()  
 if self.master.is\_sobel\_state:  
 self.master.image\_viewer.deactivate\_sobel()  
 if self.master.is\_thinning\_state:  
 self.master.image\_viewer.deactivate\_thinning()  
 if self.master.is\_detectFeature\_state:  
 self.master.image\_viewer.deactivate\_detectFeature()  
 if self.master.is\_size\_state:  
 self.master.image\_viewer.deactivate\_size()  
 if self.master.is\_detectObj\_state:  
 self.master.image\_viewer.deactivate\_detectObj()  
  
 else:  
 self.master.image\_viewer.activate\_detectObj()  
  
 def detectFeature\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.detectFeature\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_canny\_state:  
 self.master.image\_viewer.deactivate\_canny()  
 if self.master.is\_prewitt\_state:  
 self.master.image\_viewer.deactivate\_prewitt()  
 if self.master.is\_sobel\_state:  
 self.master.image\_viewer.deactivate\_sobel()  
 if self.master.is\_thinning\_state:  
 self.master.image\_viewer.deactivate\_thinning()  
 if self.master.is\_detectFeature\_state:  
 self.master.image\_viewer.deactivate\_detectFeature()  
 if self.master.is\_size\_state:  
 self.master.image\_viewer.deactivate\_size()  
 if self.master.is\_detectObj\_state:  
 self.master.image\_viewer.deactivate\_detectObj()  
 else:  
 self.master.image\_viewer.activate\_detectFeature()  
  
 def size\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.size\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_canny\_state:  
 self.master.image\_viewer.deactivate\_canny()  
 if self.master.is\_prewitt\_state:  
 self.master.image\_viewer.deactivate\_prewitt()  
 if self.master.is\_sobel\_state:  
 self.master.image\_viewer.deactivate\_sobel()  
 if self.master.is\_thinning\_state:  
 self.master.image\_viewer.deactivate\_thinning()  
 if self.master.is\_detectFeature\_state:  
 self.master.image\_viewer.deactivate\_detectFeature()  
 if self.master.is\_size\_state:  
 self.master.image\_viewer.deactivate\_size()  
 if self.master.is\_detectObj\_state:  
 self.master.image\_viewer.deactivate\_detectObj()  
  
 else:  
 self.master.image\_viewer.activate\_size()  
  
 def thinning\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.thinning\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_canny\_state:  
 self.master.image\_viewer.deactivate\_canny()  
 if self.master.is\_prewitt\_state:  
 self.master.image\_viewer.deactivate\_prewitt()  
 if self.master.is\_sobel\_state:  
 self.master.image\_viewer.deactivate\_sobel()  
 if self.master.is\_thinning\_state:  
 self.master.image\_viewer.deactivate\_thinning()  
 if self.master.is\_detectFeature\_state:  
 self.master.image\_viewer.deactivate\_detectFeature()  
 if self.master.is\_size\_state:  
 self.master.image\_viewer.deactivate\_size()  
 if self.master.is\_detectObj\_state:  
 self.master.image\_viewer.deactivate\_detectObj()  
  
 else:  
 self.master.image\_viewer.activate\_thinning()  
  
 def crop\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.crop\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_canny\_state:  
 self.master.image\_viewer.deactivate\_canny()  
 if self.master.is\_prewitt\_state:  
 self.master.image\_viewer.deactivate\_prewitt()  
 if self.master.is\_sobel\_state:  
 self.master.image\_viewer.deactivate\_sobel()  
 if self.master.is\_detectObj\_state:  
 self.master.image\_viewer.deactivate\_detectObj()  
 if self.master.is\_thinning\_state:  
 self.master.image\_viewer.deactivate\_thinning()  
 if self.master.is\_detectFeature\_state:  
 self.master.image\_viewer.deactivate\_detectFeature()  
 if self.master.is\_size\_state:  
 self.master.image\_viewer.deactivate\_size()  
 else:  
 self.master.image\_viewer.activate\_crop()  
  
 def merge\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.merge\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_canny\_state:  
 self.master.image\_viewer.deactivate\_canny()  
 if self.master.is\_prewitt\_state:  
 self.master.image\_viewer.deactivate\_prewitt()  
 if self.master.is\_sobel\_state:  
 self.master.image\_viewer.deactivate\_sobel()  
 if self.master.is\_thinning\_state:  
 self.master.image\_viewer.deactivate\_thinning()  
 if self.master.is\_detectFeature\_state:  
 self.master.image\_viewer.deactivate\_detectFeature()  
 if self.master.is\_size\_state:  
 self.master.image\_viewer.deactivate\_size()  
 if self.master.is\_detectObj\_state:  
 self.master.image\_viewer.deactivate\_detectObj()  
  
 self.master.merge\_frame = MergeFrame(master=self.master)  
 self.master.merge\_frame.grab\_set()  
  
 def canny\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.canny\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_canny\_state:  
 self.master.image\_viewer.deactivate\_canny()  
 if self.master.is\_prewitt\_state:  
 self.master.image\_viewer.deactivate\_prewitt()  
 if self.master.is\_sobel\_state:  
 self.master.image\_viewer.deactivate\_sobel()  
 if self.master.is\_thinning\_state:  
 self.master.image\_viewer.deactivate\_thinning()  
 if self.master.is\_detectFeature\_state:  
 self.master.image\_viewer.deactivate\_detectFeature()  
 if self.master.is\_size\_state:  
 self.master.image\_viewer.deactivate\_size()  
 if self.master.is\_detectObj\_state:  
 self.master.image\_viewer.deactivate\_detectObj()  
  
 else:  
 self.master.image\_viewer.activate\_canny()  
  
 def prewitt\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.prewitt\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_canny\_state:  
 self.master.image\_viewer.deactivate\_canny()  
 if self.master.is\_prewitt\_state:  
 self.master.image\_viewer.deactivate\_prewitt()  
 if self.master.is\_sobel\_state:  
 self.master.image\_viewer.deactivate\_sobel()  
 if self.master.is\_thinning\_state:  
 self.master.image\_viewer.deactivate\_thinning()  
 if self.master.is\_detectFeature\_state:  
 self.master.image\_viewer.deactivate\_detectFeature()  
 if self.master.is\_size\_state:  
 self.master.image\_viewer.deactivate\_size()  
 if self.master.is\_detectObj\_state:  
 self.master.image\_viewer.deactivate\_detectObj()  
  
 else:  
 self.master.image\_viewer.activate\_prewitt()  
  
 def sobel\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.sobel\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_canny\_state:  
 self.master.image\_viewer.deactivate\_canny()  
 if self.master.is\_prewitt\_state:  
 self.master.image\_viewer.deactivate\_prewitt()  
 if self.master.is\_sobel\_state:  
 self.master.image\_viewer.deactivate\_sobel()  
 if self.master.is\_thinning\_state:  
 self.master.image\_viewer.deactivate\_thinning()  
 if self.master.is\_detectFeature\_state:  
 self.master.image\_viewer.deactivate\_detectFeature()  
 if self.master.is\_size\_state:  
 self.master.image\_viewer.deactivate\_size()  
 if self.master.is\_detectObj\_state:  
 self.master.image\_viewer.deactivate\_detectObj()  
  
 else:  
 self.master.image\_viewer.activate\_sobel()  
  
  
  
class FileTypeFrame(Toplevel):  
  
 def \_\_init\_\_(self, master=None):  
 Toplevel.\_\_init\_\_(self, master=master)  
  
 self.original\_image = self.master.processed\_image  
 self.filtered\_image = None  
  
 self.bmp\_button = Button(master=self, text="Bitmaps Type")  
 self.jpeg\_button = Button(master=self, text="JPEG Type")  
 self.tiff\_button = Button(master=self, text="TIFF Type")  
 self.png\_button = Button(master=self, text="PNG Type")  
 self.cancel\_button = Button(master=self, text="Cancel")  
  
 self.bmp\_button.bind("<ButtonRelease>", self.bmp\_button\_released)  
 self.jpeg\_button.bind("<ButtonRelease>", self.jpeg\_button\_released)  
 self.tiff\_button.bind("<ButtonRelease>", self.tiff\_button\_released)  
 self.png\_button.bind("<ButtonRelease>", self.png\_button\_released)  
 self.cancel\_button.bind("<ButtonRelease>", self.cancel\_button\_released)  
  
 self.bmp\_button.pack()  
 self.jpeg\_button.pack()  
 self.tiff\_button.pack()  
 self.png\_button.pack()  
 self.cancel\_button.pack(side=RIGHT)  
  
 def bmp\_button\_released(self, event):  
 self.bmp()  
  
 def jpeg\_button\_released(self, event):  
 self.jpeg()  
  
 def tiff\_button\_released(self, event):  
 self.tiff()  
  
 def png\_button\_released(self, event):  
 self.png()  
  
 def cancel\_button\_released(self, event):  
 self.master.image\_viewer.show\_image()  
 self.close()  
  
 def bmp(self):  
 type\_filename = filedialog.asksaveasfilename()  
 type\_filename = type\_filename + ".bmp"  
  
 save\_image = self.master.processed\_image  
 cv2.imwrite(type\_filename, save\_image)  
  
 self.master.filename = type\_filename  
  
 def jpeg(self):  
 type\_filename = filedialog.asksaveasfilename()  
 type\_filename = type\_filename + ".jpeg"  
 save\_image = self.master.processed\_image  
 cv2.imwrite(type\_filename, save\_image)  
  
 self.master.filename = type\_filename  
  
 def tiff(self):  
 type\_filename = filedialog.asksaveasfilename()  
 type\_filename = type\_filename + ".tiff"  
  
 save\_image = self.master.processed\_image  
 cv2.imwrite(type\_filename, save\_image)  
  
 self.master.filename = type\_filename  
  
 def png(self):  
 type\_filename = filedialog.asksaveasfilename()  
 type\_filename = type\_filename + ".png"  
  
 save\_image = self.master.processed\_image  
 cv2.imwrite(type\_filename, save\_image)  
  
 self.master.filename = type\_filename  
  
 def close(self):  
 self.destroy()  
  
  
  
class MergeFrame(Toplevel):  
  
 def \_\_init\_\_(self, master=None):  
 Toplevel.\_\_init\_\_(self, master=master)  
  
 self.original\_image = self.master.processed\_image  
 self.original2\_image = self.master.processed2\_image  
 self.edited\_image = None  
  
 self.mergeH\_button = Button(master=self, text="Merge Horizontally")  
 self.mergeV\_button = Button(master=self, text="Merge Vertically")  
 self.cancel\_button = Button(master=self, text="Cancel")  
 self.apply\_button = Button(master=self, text="Apply")  
  
 self.mergeH\_button.bind("<ButtonRelease>", self.mergeH\_button\_released)  
 self.mergeV\_button.bind("<ButtonRelease>", self.mergeV\_button\_released)  
 self.apply\_button.bind("<ButtonRelease>", self.apply\_button\_released)  
 self.cancel\_button.bind("<ButtonRelease>", self.cancel\_button\_released)  
  
 self.mergeH\_button.pack()  
 self.mergeV\_button.pack()  
 self.apply\_button.pack()  
 self.cancel\_button.pack(side=RIGHT)  
  
 def mergeH\_button\_released(self, event):  
 self.mergeH()  
 self.show\_image(self.edited\_image)  
  
 def mergeV\_button\_released(self, event):  
 self.mergeV()  
 self.show\_image(self.edited\_image)  
  
 def apply\_button\_released(self, event):  
 self.master.processed\_image = self.edited\_image  
 self.close()  
  
 def cancel\_button\_released(self, event):  
 self.master.image\_viewer.show\_image()  
 self.close()  
  
 def show\_image(self, img=None):  
 self.master.image\_viewer.show\_image(img=img)  
  
 def mergeH(self, interpolation=cv2.INTER\_CUBIC):  
 img1 = self.original\_image  
 img2 = self.original2\_image  
 img\_list = [img1, img2]  
 h\_min = min(img.shape[0]  
 for img in img\_list)  
  
 # image resizing  
 im\_list\_hresize = [  
 cv2.resize(img, (int(img.shape[1] \* h\_min / img.shape[0]), h\_min), interpolation=interpolation) for img  
 in  
 img\_list]  
  
 self.edited\_image = cv2.hconcat(im\_list\_hresize)  
  
 def mergeV(self,interpolation=cv2.INTER\_CUBIC):  
 img1 = self.original\_image  
 img2 = self.original2\_image  
 img\_list = [img1, img2]  
  
 w\_min = min(img.shape[1]  
 for img in img\_list)  
  
 # resizing images  
 im\_list\_vresize = [  
 cv2.resize(img, (w\_min, int(img.shape[0] \* w\_min / img.shape[1])), interpolation=interpolation) for img in  
 img\_list]  
  
 self.edited\_image = cv2.vconcat(im\_list\_vresize)  
  
 def close(self):  
 self.destroy()  
  
  
class ImageViewer(Frame):  
  
 def \_\_init\_\_(self, master=None):  
 Frame.\_\_init\_\_(self, master=master, bg="#94D9FF", width=800, height=500)  
  
 self.shown\_image = None  
 self.x = 0  
 self.y = 0  
 self.crop\_start\_x = 0  
 self.crop\_start\_y = 0  
 self.crop\_end\_x = 0  
 self.crop\_end\_y = 0  
 self.draw\_ids = list()  
 self.rectangle\_id = 0  
 self.ratio = 0  
 self.canvas = Canvas(self, bg="#BFE9FF", width=800, height=500)  
 self.canvas.place(relx=0.5, rely=0.5, anchor=CENTER)  
 self.canvas2 = Canvas(self, bg="#CBFDFF", width=200, height=200)  
 self.canvas2.place(relx=0.87, rely=0.8, anchor=CENTER)  
 self.canvas3 = Canvas(self, bg="#CBFDFF", width=200, height=200)  
 self.canvas3.place(relx=0.13, rely=0.8, anchor=CENTER)  
  
 def show\_image(self, img=None):  
 self.clear\_canvas()  
  
 if img is None:  
 image3 = self.master.original\_image.copy()  
 image = self.master.processed\_image.copy()  
 image2 = self.master.processed2\_image.copy()  
 else:  
 image3 = self.master.original\_image.copy()  
 image = img  
 image2 = self.master.processed2\_image.copy()  
  
 image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  
 height, width, channels = image.shape  
 ratio = height / width  
  
 image2 = cv2.cvtColor(image2, cv2.COLOR\_BGR2RGB)  
 height2, width2, channels2 = image2.shape  
 ratio2 = height2 / width2  
  
 image3 = cv2.cvtColor(image3, cv2.COLOR\_BGR2RGB)  
 height3, width3, channels3 = image3.shape  
 ratio3 = height3 / width3  
  
 new\_width = width  
 new\_height = height  
 new\_width2 = width2  
 new\_height2 = height2  
 new\_width3 = width3  
 new\_height3 = height3  
  
 if height > 600 or width > 800:  
 if ratio < 1:  
 new\_width = 800  
 new\_height = int(new\_width \* ratio)  
 else:  
 new\_height = 600  
 new\_width = int(new\_height \* (width / height))  
  
 if height2 > 200 or width2 > 200:  
 if ratio2 < 1:  
 new\_width2 = 200  
 new\_height2 = int(new\_width2 \* ratio2)  
 else:  
 new\_height2 = 200  
 new\_width2 = int(new\_height2 \* (width2 / height2))  
  
 if height3 > 200 or width3 > 200:  
 if ratio3 < 1:  
 new\_width3 = 200  
 new\_height3 = int(new\_width3 \* ratio3)  
 else:  
 new\_height3 = 200  
 new\_width3 = int(new\_height3 \* (width3 / height3))  
  
 self.shown\_image = cv2.resize(image, (new\_width, new\_height))  
 self.shown\_image = ImageTk.PhotoImage(Image.fromarray(self.shown\_image))  
 self.shown2\_image = cv2.resize(image2, (new\_width2, new\_height2))  
 self.shown2\_image = ImageTk.PhotoImage(Image.fromarray(self.shown2\_image))  
 self.shown3\_image = cv2.resize(image3, (new\_width3, new\_height3))  
 self.shown3\_image = ImageTk.PhotoImage(Image.fromarray(self.shown3\_image))  
  
 self.ratio = height / new\_height  
 self.ratio2 = height2 / new\_height2  
 self.ratio3 = height3 / new\_height3  
  
 self.canvas.config(width=new\_width, height=new\_height)  
 self.canvas.create\_image(new\_width / 2, new\_height / 2, anchor=CENTER, image=self.shown\_image)  
 self.canvas2.config(width=new\_width2, height=new\_height2)  
 self.canvas2.create\_image(new\_width2 / 2, new\_height2 / 2, anchor=CENTER, image=self.shown2\_image)  
 self.canvas3.config(width=new\_width3, height=new\_height3)  
 self.canvas3.create\_image(new\_width3 / 2, new\_height3 / 2, anchor=CENTER, image=self.shown3\_image)  
  
 def activate\_crop(self):  
 self.canvas.bind("<ButtonPress>", self.start\_crop)  
 self.canvas.bind("<B1-Motion>", self.crop)  
 self.canvas.bind("<ButtonRelease>", self.end\_crop)  
  
 self.master.is\_crop\_state = True  
  
 def deactivate\_crop(self):  
 self.canvas.unbind("<ButtonPress>")  
 self.canvas.unbind("<B1-Motion>")  
 self.canvas.unbind("<ButtonRelease>")  
  
 self.master.is\_crop\_state = False  
  
 def start\_crop(self, event):  
 self.crop\_start\_x = event.x  
 self.crop\_start\_y = event.y  
  
 def crop(self, event):  
 if self.rectangle\_id:  
 self.canvas.delete(self.rectangle\_id)  
  
 self.crop\_end\_x = event.x  
 self.crop\_end\_y = event.y  
  
 self.rectangle\_id = self.canvas.create\_rectangle(self.crop\_start\_x, self.crop\_start\_y,  
 self.crop\_end\_x, self.crop\_end\_y, width=1)  
  
 def end\_crop(self, event):  
 if self.crop\_start\_x <= self.crop\_end\_x and self.crop\_start\_y <= self.crop\_end\_y:  
 start\_x = int(self.crop\_start\_x \* self.ratio)  
 start\_y = int(self.crop\_start\_y \* self.ratio)  
 end\_x = int(self.crop\_end\_x \* self.ratio)  
 end\_y = int(self.crop\_end\_y \* self.ratio)  
 elif self.crop\_start\_x > self.crop\_end\_x and self.crop\_start\_y <= self.crop\_end\_y:  
 start\_x = int(self.crop\_end\_x \* self.ratio)  
 start\_y = int(self.crop\_start\_y \* self.ratio)  
 end\_x = int(self.crop\_start\_x \* self.ratio)  
 end\_y = int(self.crop\_end\_y \* self.ratio)  
 elif self.crop\_start\_x <= self.crop\_end\_x and self.crop\_start\_y > self.crop\_end\_y:  
 start\_x = int(self.crop\_start\_x \* self.ratio)  
 start\_y = int(self.crop\_end\_y \* self.ratio)  
 end\_x = int(self.crop\_end\_x \* self.ratio)  
 end\_y = int(self.crop\_start\_y \* self.ratio)  
 else:  
 start\_x = int(self.crop\_end\_x \* self.ratio)  
 start\_y = int(self.crop\_end\_y \* self.ratio)  
 end\_x = int(self.crop\_start\_x \* self.ratio)  
 end\_y = int(self.crop\_start\_y \* self.ratio)  
  
 x = slice(start\_x, end\_x, 1)  
 y = slice(start\_y, end\_y, 1)  
  
 self.master.processed\_image = self.master.processed\_image[y, x]  
  
 self.show\_image()  
  
 def activate\_size(self):  
 self.detectObj()  
  
 # cv2.imshow("imgPlate", licPlate.imgPlate) # show crop of plate and threshold of plate  
 # cv2.imshow("imgThresh", licPlate.imgThresh)  
  
 self.drawRedRectangleAroundPlate(self.imgOriginalScene, self.licPlate)  
  
 self.writeSizeOnImage(self.imgOriginalScene, self.licPlate)  
  
 self.master.processed\_image = self.imgOriginalScene  
  
 self.show\_image()  
  
 def deactivate\_size(self):  
 pass  
  
 def activate\_detectFeature(self):  
 self.detectObj()  
  
 # cv2.imshow("imgPlate", licPlate.imgPlate) # show crop of plate and threshold of plate  
 # cv2.imshow("imgThresh", licPlate.imgThresh)  
  
 self.drawRedRectangleAroundPlate(self.imgOriginalScene, self.licPlate)  
  
 self.writeLicensePlateCharsOnImage(self.imgOriginalScene, self.licPlate)  
  
 self.master.processed\_image = self.imgOriginalScene  
  
 self.show\_image()  
  
 def deactivate\_detectFeature(self):  
 pass  
  
 def activate\_detectObj(self):  
 global choice  
 choice = 'Choice 4'  
 self.detectObj()  
  
 # cv2.imshow("imgPlate", licPlate.imgPlate) # show crop of plate and threshold of plate  
 # cv2.imshow("imgThresh", licPlate.imgThresh)  
  
 self.drawRedRectangleAroundPlate(self.imgOriginalScene, self.licPlate)  
 self.show\_image()  
  
 def deactivate\_detectObj(self):  
 pass  
  
 def activate\_thinning(self):  
 retval,imgThresh2=cv2.threshold(self.licPlate.imgThresh,62,255,cv2.THRESH\_BINARY\_INV)  
 self.master.processed\_image = imgThresh2  
 self.show\_image()  
  
 def deactivate\_thinning(self):  
 pass  
  
 def activate\_canny(self):  
 global choice  
 choice = 'Choice 1'  
 self.detectObj()  
 self.drawRedRectangleAroundPlate(self.imgOriginalScene, self.licPlate)  
 self.show\_image()  
  
 def activate\_prewitt(self):  
 global choice  
 choice = 'Choice 2'  
 self.detectObj()  
  
 self.drawRedRectangleAroundPlate(self.imgOriginalScene, self.licPlate)  
 self.show\_image()  
  
 def activate\_sobel(self):  
 global choice  
 choice = 'Choice 3'  
 self.detectObj()  
  
 self.drawRedRectangleAroundPlate(self.imgOriginalScene, self.licPlate)  
 self.show\_image()  
  
 def clear\_canvas(self):  
 self.canvas.delete("all")  
  
 def detectObj(self):  
 blnKNNTrainingSuccessful = loadKNNDataAndTrainKNN()  
  
 if blnKNNTrainingSuccessful == False:  
 print("\nerror: KNN traning was not successful\n")  
 return  
  
 self.imgOriginalScene = self.master.processed\_image  
  
 if self.imgOriginalScene is None:  
 print("\nerror: image not read from file \n\n")  
 os.system("pause")  
 return  
  
 listOfPossiblePlates = detectPlatesInScene(self.imgOriginalScene)  
  
 listOfPossiblePlates = detectCharsInPlates(listOfPossiblePlates)  
  
 if len(listOfPossiblePlates) == 0:  
 print("\nno license plates were detected\n")  
 else:  
  
 listOfPossiblePlates.sort(key=lambda possiblePlate: len(possiblePlate.strChars), reverse=True)  
  
 self.licPlate = listOfPossiblePlates[0]  
  
 def drawRedRectangleAroundPlate(self, imgOriginalScene, licPlate):  
  
 p2fRectPoints = cv2.boxPoints(licPlate.rrLocationOfPlateInScene)  
  
 cv2.line(imgOriginalScene, tuple(p2fRectPoints[0]), tuple(p2fRectPoints[1]), SCALAR\_RED, 2)  
 cv2.line(imgOriginalScene, tuple(p2fRectPoints[1]), tuple(p2fRectPoints[2]), SCALAR\_RED, 2)  
 cv2.line(imgOriginalScene, tuple(p2fRectPoints[2]), tuple(p2fRectPoints[3]), SCALAR\_RED, 2)  
 cv2.line(imgOriginalScene, tuple(p2fRectPoints[3]), tuple(p2fRectPoints[0]), SCALAR\_RED, 2)  
  
 self.master.processed\_image = imgOriginalScene  
  
 def writeLicensePlateCharsOnImage(self, imgOriginalScene, licPlate):  
  
 sceneHeight, sceneWidth, sceneNumChannels = imgOriginalScene.shape  
 plateHeight, plateWidth, plateNumChannels = licPlate.imgPlate.shape  
  
 intFontFace = cv2.FONT\_HERSHEY\_SIMPLEX  
 fltFontScale = float(plateHeight) / 50.0  
 intFontThickness = int(round(fltFontScale \* 1.5))  
  
 textSize, baseline = cv2.getTextSize(licPlate.strChars, intFontFace, fltFontScale,intFontThickness)  
  
 ((intPlateCenterX, intPlateCenterY), (intPlateWidth, intPlateHeight),fltCorrectionAngleInDeg) = licPlate.rrLocationOfPlateInScene  
  
 intPlateCenterX = int(intPlateCenterX)  
 intPlateCenterY = int(intPlateCenterY)  
  
 ptCenterOfTextAreaX = int(intPlateCenterX)  
  
 if intPlateCenterY < (sceneHeight \* 0.75):  
 ptCenterOfTextAreaY = int(round(intPlateCenterY)) + int(round(plateHeight \* 1.6))  
 else:  
 ptCenterOfTextAreaY = int(round(intPlateCenterY)) - int(round(plateHeight \* 1.6))  
  
 textSizeWidth, textSizeHeight = textSize  
 ptLowerLeftTextOriginX = int(  
 ptCenterOfTextAreaX - (textSizeWidth / 2))  
 ptLowerLeftTextOriginY = int(  
 ptCenterOfTextAreaY + (textSizeHeight / 2))  
  
 cv2.putText(imgOriginalScene, licPlate.strChars, (ptLowerLeftTextOriginX, ptLowerLeftTextOriginY), intFontFace, fltFontScale, SCALAR\_YELLOW, intFontThickness)  
 # cv2.putText(imgOriginalScene, "{:.1f}in".format(wid), (ptLowerLeftTextOriginX+20, ptLowerLeftTextOriginY+40), intFontFace,  
 # fltFontScale, SCALAR\_YELLOW, intFontThickness)  
 # cv2.putText(imgOriginalScene, "{:.1f}in".format(ht), ((ptLowerLeftTextOriginX+280), (ptLowerLeftTextOriginY+100)),intFontFace,  
 # fltFontScale, SCALAR\_YELLOW, intFontThickness)  
  
 def writeSizeOnImage(self, imgOriginalScene, licPlate):  
 ptCenterOfTextAreaX = 0 # this will be the center of the area the text will be written to  
 ptCenterOfTextAreaY = 0  
  
 ptLowerLeftTextOriginX = 0 # this will be the bottom left of the area that the text will be written to  
 ptLowerLeftTextOriginY = 0  
  
 sceneHeight, sceneWidth, sceneNumChannels = imgOriginalScene.shape  
 plateHeight, plateWidth, plateNumChannels = licPlate.imgPlate.shape  
  
 intFontFace = cv2.FONT\_HERSHEY\_SIMPLEX  
 fltFontScale = float(plateHeight) / 50.0  
 intFontThickness = int(round(fltFontScale \* 1.5))  
  
 textSize, baseline = cv2.getTextSize(licPlate.strChars, intFontFace, fltFontScale,intFontThickness)  
  
 ((intPlateCenterX, intPlateCenterY), (intPlateWidth, intPlateHeight),fltCorrectionAngleInDeg) = licPlate.rrLocationOfPlateInScene  
  
 intPlateCenterX = int(intPlateCenterX)  
 intPlateCenterY = int(intPlateCenterY)  
  
 ptCenterOfTextAreaX = int(intPlateCenterX)  
  
 if intPlateCenterY < (sceneHeight \* 0.75):  
 ptCenterOfTextAreaY = int(round(intPlateCenterY)) + int(round(plateHeight \* 1.6))  
 else:  
 ptCenterOfTextAreaY = int(round(intPlateCenterY)) - int(round(plateHeight \* 1.6))  
  
 textSizeWidth, textSizeHeight = textSize  
 ptLowerLeftTextOriginX = int(  
 ptCenterOfTextAreaX - (textSizeWidth / 2))  
 ptLowerLeftTextOriginY = int(  
 ptCenterOfTextAreaY + (textSizeHeight / 2))  
  
 pixels\_per\_metric = 150 / 0.955  
 # pixels\_per\_metric = 150 / (0.955 \* 2.54)  
 wid = plateWidth / pixels\_per\_metric  
 ht = plateHeight / pixels\_per\_metric  
  
 # cv2.putText(imgOriginalScene, licPlate.strChars, (ptLowerLeftTextOriginX, ptLowerLeftTextOriginY), intFontFace, fltFontScale, SCALAR\_YELLOW, intFontThickness)  
 cv2.putText(imgOriginalScene, "{:.1f}in".format(wid), (intPlateCenterX - 50, intPlateCenterY - 40), intFontFace,  
 fltFontScale, SCALAR\_YELLOW, intFontThickness)  
 cv2.putText(imgOriginalScene, "{:.1f}in".format(ht), ((intPlateCenterX + 100), (intPlateCenterY + 20)),  
 intFontFace,fltFontScale, SCALAR\_YELLOW, intFontThickness)  
  
class PossibleChar:  
  
 def \_\_init\_\_(self, \_contour):  
 self.contour = \_contour  
  
 self.boundingRect = cv2.boundingRect(self.contour)  
  
 [intX, intY, intWidth, intHeight] = self.boundingRect  
  
 self.intBoundingRectX = intX  
 self.intBoundingRectY = intY  
 self.intBoundingRectWidth = intWidth  
 self.intBoundingRectHeight = intHeight  
  
 self.intBoundingRectArea = self.intBoundingRectWidth \* self.intBoundingRectHeight  
  
 self.intCenterX = (self.intBoundingRectX + self.intBoundingRectX + self.intBoundingRectWidth) / 2  
 self.intCenterY = (self.intBoundingRectY + self.intBoundingRectY + self.intBoundingRectHeight) / 2  
  
 self.fltDiagonalSize = math.sqrt((self.intBoundingRectWidth \*\* 2) + (self.intBoundingRectHeight \*\* 2))  
  
 self.fltAspectRatio = float(self.intBoundingRectWidth) / float(self.intBoundingRectHeight)  
  
class PossiblePlate:  
  
 # constructor #################################################################################  
 def \_\_init\_\_(self):  
 self.imgPlate = None  
 self.imgGrayscale = None  
 self.imgThresh = None  
  
 self.rrLocationOfPlateInScene = None  
  
 self.strChars = ""  
  
def preprocess(imgOriginal):  
 imgGrayscale = extractValue(imgOriginal)  
  
 imgMaxContrastGrayscale = maximizeContrast(imgGrayscale)  
  
 height, width = imgGrayscale.shape  
  
 imgBlurred = np.zeros((height, width, 1), np.uint8)  
  
 imgBlurred = cv2.GaussianBlur(imgMaxContrastGrayscale, GAUSSIAN\_SMOOTH\_FILTER\_SIZE, 0)  
 imgThresh = cv2.adaptiveThreshold(imgBlurred, 255.0, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv2.THRESH\_BINARY\_INV, ADAPTIVE\_THRESH\_BLOCK\_SIZE, ADAPTIVE\_THRESH\_WEIGHT)  
  
 return imgGrayscale, imgThresh  
  
def canny( imgOriginal):  
 # imgGrayscale = extractValue(imgOriginal)  
 # imgMaxContrastGrayscale = maximizeContrast(imgGrayscale)  
 # height, width = imgGrayscale.shape  
 # imgBlurred = np.zeros((height, width, 1), np.uint8)  
 # imgBlurred = cv2.GaussianBlur(imgMaxContrastGrayscale, GAUSSIAN\_SMOOTH\_FILTER\_SIZE, 0)  
 imgGrayscale = extractValue(imgOriginal)  
 imgMaxContrastGrayscale = maximizeContrast(imgGrayscale)  
 imgBlurred = cv2.GaussianBlur(imgMaxContrastGrayscale, (9, 9), 0)  
 edged = cv2.Canny(imgBlurred, 50, 100)  
 edged = cv2.dilate(edged, None, iterations=1)  
 imgThresh = cv2.erode(edged, None, iterations=1)  
  
 return imgGrayscale, imgThresh  
  
def prewitt( imgOriginal):  
 # imgGrayscale = extractValue(imgOriginal)  
 # imgMaxContrastGrayscale = maximizeContrast(imgGrayscale)  
 # height, width = imgGrayscale.shape  
 # imgBlurred = np.zeros((height, width, 1), np.uint8)  
 # imgBlurred = cv2.GaussianBlur(imgMaxContrastGrayscale, GAUSSIAN\_SMOOTH\_FILTER\_SIZE, 0)  
  
 imgGrayscale = extractValue(imgOriginal)  
 imgMaxContrastGrayscale = maximizeContrast(imgGrayscale)  
 imgBlurred = cv2.GaussianBlur(imgMaxContrastGrayscale, (9, 9), 0)  
  
 kernelx = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]])  
 kernely = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])  
 img\_prewittx = cv2.filter2D(imgBlurred, -1, kernelx)  
 img\_prewitty = cv2.filter2D(imgBlurred, -1, kernely)  
 edged = img\_prewittx + img\_prewitty  
 edged = cv2.dilate(edged, None, iterations=1)  
 kernel\_sharpening = np.array(  
 [[-1, -1, -1, -1, -1], [-1, -1, -1, -1, -1], [-1, -1, 25, -1, -1], [-1, -1, -1, -1, -1],  
 [-1, -1, -1, -1, -1]])  
 sharpen=cv2.filter2D(edged, -1, kernel\_sharpening)  
 imgThresh = cv2.erode(sharpen, None, iterations=1)  
  
 return imgGrayscale, imgThresh  
  
def sobel( imgOriginal):  
 # imgGrayscale = extractValue(imgOriginal)  
 # imgMaxContrastGrayscale = maximizeContrast(imgGrayscale)  
 # height, width = imgGrayscale.shape  
 # imgBlurred = np.zeros((height, width, 1), np.uint8)  
 # imgBlurred = cv2.GaussianBlur(imgMaxContrastGrayscale, GAUSSIAN\_SMOOTH\_FILTER\_SIZE, 0)  
  
 imgGrayscale = extractValue(imgOriginal)  
 imgMaxContrastGrayscale = maximizeContrast(imgGrayscale)  
 imgBlurred = cv2.GaussianBlur(imgMaxContrastGrayscale, (9, 9), 0)  
 sobelx = cv2.Sobel(imgBlurred,cv2.CV\_8U,1,0,ksize=3)  
 sobely = cv2.Sobel(imgBlurred,cv2.CV\_8U,0,1,ksize=3)  
 edged = sobelx + sobely  
 \_, img2 = cv2.threshold(edged, 0, 255, cv2.THRESH\_BINARY + cv2.THRESH\_OTSU)  
 imgThresh = img2.copy()  
 # edged = cv2.dilate(edged, None, iterations=1)  
 # imgThresh = cv2.erode(edged, None, iterations=1)  
  
 return imgGrayscale, imgThresh  
  
def getThreshGray (imgOriginal):  
 global choice  
 if choice == 'Choice 1':  
 imgThresh, imgGrayscale = canny(imgOriginal)  
 elif choice == 'Choice 2':  
 imgThresh, imgGrayscale = prewitt(imgOriginal)  
 elif choice == 'Choice 3':  
 imgThresh, imgGrayscale = sobel(imgOriginal)  
 elif choice == 'Choice 4':  
 imgThresh, imgGrayscale = preprocess(imgOriginal)  
 else:  
 imgThresh, imgGrayscale=preprocess(imgOriginal)  
  
 return imgThresh, imgGrayscale  
  
def extractValue(imgOriginal):  
 height, width, numChannels = imgOriginal.shape  
  
 imgHSV = np.zeros((height, width, 3), np.uint8)  
  
 imgHSV = cv2.cvtColor(imgOriginal, cv2.COLOR\_BGR2HSV)  
  
 imgHue, imgSaturation, imgValue = cv2.split(imgHSV)  
  
 return imgValue  
  
def maximizeContrast(imgGrayscale):  
  
 height, width = imgGrayscale.shape  
  
 imgTopHat = np.zeros((height, width, 1), np.uint8)  
 imgBlackHat = np.zeros((height, width, 1), np.uint8)  
  
 structuringElement = cv2.getStructuringElement(cv2.MORPH\_RECT, (3, 3))  
  
 imgTopHat = cv2.morphologyEx(imgGrayscale, cv2.MORPH\_TOPHAT, structuringElement)  
 imgBlackHat = cv2.morphologyEx(imgGrayscale, cv2.MORPH\_BLACKHAT, structuringElement)  
  
 imgGrayscalePlusTopHat = cv2.add(imgGrayscale, imgTopHat)  
 imgGrayscalePlusTopHatMinusBlackHat = cv2.subtract(imgGrayscalePlusTopHat, imgBlackHat)  
  
 return imgGrayscalePlusTopHatMinusBlackHat  
  
def loadKNNDataAndTrainKNN():  
 allContoursWithData = []  
 validContoursWithData = []  
  
 try:  
 npaClassifications = np.loadtxt("classifications.txt", np.float32)  
 except:  
 print("error, unable to open classifications.txt, exiting program\n")  
 os.system("pause")  
 return False  
  
 try:  
 npaFlattenedImages = np.loadtxt("flattened\_images.txt", np.float32)  
 except:  
 print("error, unable to open flattened\_images.txt, exiting program\n")  
 os.system("pause")  
 return False  
 # end try  
  
 npaClassifications = npaClassifications.reshape((npaClassifications.size, 1))  
  
 kNearest.setDefaultK(1)  
  
 kNearest.train(npaFlattenedImages, cv2.ml.ROW\_SAMPLE, npaClassifications)  
  
 return True  
  
def detectCharsInPlates(listOfPossiblePlates):  
 intPlateCounter = 0  
 imgContours = None  
 contours = []  
  
 if len(listOfPossiblePlates) == 0:  
 return listOfPossiblePlates  
  
 for possiblePlate in listOfPossiblePlates:  
 possiblePlate.imgGrayscale, possiblePlate.imgThresh = getThreshGray(possiblePlate.imgPlate)  
  
 possiblePlate.imgThresh = cv2.resize(possiblePlate.imgThresh, (0, 0), fx = 1.6, fy = 1.6)  
  
 thresholdValue, possiblePlate.imgThresh = cv2.threshold(possiblePlate.imgThresh, 0.0, 255.0, cv2.THRESH\_BINARY |cv2.THRESH\_OTSU)  
  
 listOfPossibleCharsInPlate = findPossibleCharsInPlate(possiblePlate.imgGrayscale, possiblePlate.imgThresh)  
  
 listOfListsOfMatchingCharsInPlate = findListOfListsOfMatchingChars(listOfPossibleCharsInPlate)  
  
 if (len(listOfListsOfMatchingCharsInPlate) == 0):  
  
 possiblePlate.strChars = ""  
 continue  
  
 for i in range(0, len(listOfListsOfMatchingCharsInPlate)):  
 listOfListsOfMatchingCharsInPlate[i].sort(key = lambda matchingChar: matchingChar.intCenterX)  
 listOfListsOfMatchingCharsInPlate[i] = removeInnerOverlappingChars(listOfListsOfMatchingCharsInPlate[i])  
  
 intLenOfLongestListOfChars = 0  
 intIndexOfLongestListOfChars = 0  
  
 for i in range(0, len(listOfListsOfMatchingCharsInPlate)):  
 if len(listOfListsOfMatchingCharsInPlate[i]) > intLenOfLongestListOfChars:  
 intLenOfLongestListOfChars = len(listOfListsOfMatchingCharsInPlate[i])  
 intIndexOfLongestListOfChars = i  
  
 longestListOfMatchingCharsInPlate = listOfListsOfMatchingCharsInPlate[intIndexOfLongestListOfChars]  
  
 possiblePlate.strChars = recognizeCharsInPlate(possiblePlate.imgThresh, longestListOfMatchingCharsInPlate)  
  
 return listOfPossiblePlates  
  
def findPossibleCharsInPlate(imgGrayscale, imgThresh):  
 listOfPossibleChars = []  
 imgThreshCopy = imgThresh.copy()  
  
 contours, npaHierarchy = cv2.findContours(imgThreshCopy, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)  
  
 for contour in contours:  
 possibleChar = PossibleChar(contour)  
  
 if checkIfPossibleChar(possibleChar):  
 listOfPossibleChars.append(possibleChar)  
  
 return listOfPossibleChars  
  
def checkIfPossibleChar(possibleChar):  
 if (possibleChar.intBoundingRectArea > MIN\_PIXEL\_AREA and  
 possibleChar.intBoundingRectWidth > MIN\_PIXEL\_WIDTH and possibleChar.intBoundingRectHeight > MIN\_PIXEL\_HEIGHT and  
 MIN\_ASPECT\_RATIO < possibleChar.fltAspectRatio and possibleChar.fltAspectRatio < MAX\_ASPECT\_RATIO):  
 return True  
 else:  
 return False  
  
def findListOfListsOfMatchingChars(listOfPossibleChars):  
 listOfListsOfMatchingChars = []  
  
 for possibleChar in listOfPossibleChars:  
 listOfMatchingChars = findListOfMatchingChars(possibleChar, listOfPossibleChars)  
 listOfMatchingChars.append(possibleChar)  
  
 if len(listOfMatchingChars) < MIN\_NUMBER\_OF\_MATCHING\_CHARS:  
 continue  
 listOfListsOfMatchingChars.append(listOfMatchingChars)  
  
 listOfPossibleCharsWithCurrentMatchesRemoved = []  
  
 listOfPossibleCharsWithCurrentMatchesRemoved = list(set(listOfPossibleChars) - set(listOfMatchingChars))  
  
 recursiveListOfListsOfMatchingChars = findListOfListsOfMatchingChars(listOfPossibleCharsWithCurrentMatchesRemoved)  
  
 for recursiveListOfMatchingChars in recursiveListOfListsOfMatchingChars:  
 listOfListsOfMatchingChars.append(recursiveListOfMatchingChars)  
  
 break  
  
 return listOfListsOfMatchingChars  
  
def findListOfMatchingChars(possibleChar, listOfChars):  
 listOfMatchingChars = []  
  
 for possibleMatchingChar in listOfChars:  
 if possibleMatchingChar == possibleChar:  
 continue  
  
 fltDistanceBetweenChars = distanceBetweenChars(possibleChar, possibleMatchingChar)  
  
 fltAngleBetweenChars = angleBetweenChars(possibleChar, possibleMatchingChar)  
  
 fltChangeInArea = float(abs(possibleMatchingChar.intBoundingRectArea - possibleChar.intBoundingRectArea)) / float(possibleChar.intBoundingRectArea)  
  
 fltChangeInWidth = float(abs(possibleMatchingChar.intBoundingRectWidth - possibleChar.intBoundingRectWidth)) / float(possibleChar.intBoundingRectWidth)  
 fltChangeInHeight = float(abs(possibleMatchingChar.intBoundingRectHeight - possibleChar.intBoundingRectHeight)) / float(possibleChar.intBoundingRectHeight)  
  
  
 if (fltDistanceBetweenChars < (possibleChar.fltDiagonalSize \* MAX\_DIAG\_SIZE\_MULTIPLE\_AWAY) and  
 fltAngleBetweenChars < MAX\_ANGLE\_BETWEEN\_CHARS and  
 fltChangeInArea < MAX\_CHANGE\_IN\_AREA and  
 fltChangeInWidth < MAX\_CHANGE\_IN\_WIDTH and  
 fltChangeInHeight < MAX\_CHANGE\_IN\_HEIGHT):  
  
 listOfMatchingChars.append(possibleMatchingChar)  
  
 return listOfMatchingChars  
  
def distanceBetweenChars(firstChar, secondChar):  
 intX = abs(firstChar.intCenterX - secondChar.intCenterX)  
 intY = abs(firstChar.intCenterY - secondChar.intCenterY)  
  
 return math.sqrt((intX \*\* 2) + (intY \*\* 2))  
  
def angleBetweenChars(firstChar, secondChar):  
 fltAdj = float(abs(firstChar.intCenterX - secondChar.intCenterX))  
 fltOpp = float(abs(firstChar.intCenterY - secondChar.intCenterY))  
  
 if fltAdj != 0.0:  
 fltAngleInRad = math.atan(fltOpp / fltAdj)  
 else:  
 fltAngleInRad = 1.5708  
  
 fltAngleInDeg = fltAngleInRad \* (180.0 / math.pi)  
  
 return fltAngleInDeg  
  
def removeInnerOverlappingChars(listOfMatchingChars):  
 listOfMatchingCharsWithInnerCharRemoved = list(listOfMatchingChars)  
 for currentChar in listOfMatchingChars:  
 for otherChar in listOfMatchingChars:  
 if currentChar != otherChar:  
 if distanceBetweenChars(currentChar, otherChar) < (currentChar.fltDiagonalSize \* MIN\_DIAG\_SIZE\_MULTIPLE\_AWAY):  
 if currentChar.intBoundingRectArea < otherChar.intBoundingRectArea:  
 if currentChar in listOfMatchingCharsWithInnerCharRemoved:  
 listOfMatchingCharsWithInnerCharRemoved.remove(currentChar)  
 else:  
 if otherChar in listOfMatchingCharsWithInnerCharRemoved:  
 listOfMatchingCharsWithInnerCharRemoved.remove(otherChar)  
  
 return listOfMatchingCharsWithInnerCharRemoved  
  
def recognizeCharsInPlate(imgThresh, listOfMatchingChars):  
 strChars = ""  
  
 height, width = imgThresh.shape  
  
 imgThreshColor = np.zeros((height, width, 3), np.uint8)  
  
 listOfMatchingChars.sort(key = lambda matchingChar: matchingChar.intCenterX)  
  
 cv2.cvtColor(imgThresh, cv2.COLOR\_GRAY2BGR, imgThreshColor)  
 for currentChar in listOfMatchingChars:  
 pt1 = (currentChar.intBoundingRectX, currentChar.intBoundingRectY)  
 pt2 = ((currentChar.intBoundingRectX + currentChar.intBoundingRectWidth), (currentChar.intBoundingRectY + currentChar.intBoundingRectHeight))  
  
 cv2.rectangle(imgThreshColor, pt1, pt2, SCALAR\_GREEN, 2)  
  
 # crop char out of threshold image  
 imgROI = imgThresh[currentChar.intBoundingRectY : currentChar.intBoundingRectY + currentChar.intBoundingRectHeight,  
 currentChar.intBoundingRectX : currentChar.intBoundingRectX + currentChar.intBoundingRectWidth]  
  
 imgROIResized = cv2.resize(imgROI, (RESIZED\_CHAR\_IMAGE\_WIDTH, RESIZED\_CHAR\_IMAGE\_HEIGHT))  
  
 npaROIResized = imgROIResized.reshape((1, RESIZED\_CHAR\_IMAGE\_WIDTH \* RESIZED\_CHAR\_IMAGE\_HEIGHT))  
  
 npaROIResized = np.float32(npaROIResized)  
  
 retval, npaResults, neigh\_resp, dists = kNearest.findNearest(npaROIResized, k = 1)  
  
 strCurrentChar = str(chr(int(npaResults[0][0])))  
  
 strChars = strChars + strCurrentChar  
  
 return strChars  
  
def detectPlatesInScene(imgOriginalScene):  
 listOfPossiblePlates = []  
  
 height, width, numChannels = imgOriginalScene.shape  
  
 imgGrayscaleScene = np.zeros((height, width, 1), np.uint8)  
 imgThreshScene = np.zeros((height, width, 1), np.uint8)  
 imgContours = np.zeros((height, width, 3), np.uint8)  
  
 cv2.destroyAllWindows()  
  
 imgGrayscaleScene, imgThreshScene = getThreshGray(imgOriginalScene)  
  
 listOfPossibleCharsInScene = findPossibleCharsInScene(imgThreshScene)  
  
 listOfListsOfMatchingCharsInScene = findListOfListsOfMatchingChars(listOfPossibleCharsInScene)  
  
 for listOfMatchingChars in listOfListsOfMatchingCharsInScene:  
 possiblePlate = extractPlate(imgOriginalScene, listOfMatchingChars)  
  
 if possiblePlate.imgPlate is not None:  
 listOfPossiblePlates.append(possiblePlate)  
 # end if  
 # end for  
  
 print("\n" + str(len(listOfPossiblePlates)) + " possible plates found")  
  
 return listOfPossiblePlates  
  
def findPossibleCharsInScene(imgThresh):  
 listOfPossibleChars = []  
  
 intCountOfPossibleChars = 0  
  
 imgThreshCopy = imgThresh.copy()  
  
 contours, npaHierarchy = cv2.findContours(imgThreshCopy, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)  
  
 height, width = imgThresh.shape  
 imgContours = np.zeros((height, width, 3), np.uint8)  
  
 for i in range(0, len(contours)):  
  
 possibleChar = PossibleChar(contours[i])  
  
 if checkIfPossibleChar(possibleChar):  
 intCountOfPossibleChars = intCountOfPossibleChars + 1  
 listOfPossibleChars.append(possibleChar)  
  
 return listOfPossibleChars  
def extractPlate(imgOriginal, listOfMatchingChars):  
 possiblePlate = PossiblePlate()  
  
 listOfMatchingChars.sort(key = lambda matchingChar: matchingChar.intCenterX)  
  
 fltPlateCenterX = (listOfMatchingChars[0].intCenterX + listOfMatchingChars[len(listOfMatchingChars) - 1].intCenterX) / 2.0  
 fltPlateCenterY = (listOfMatchingChars[0].intCenterY + listOfMatchingChars[len(listOfMatchingChars) - 1].intCenterY) / 2.0  
  
 ptPlateCenter = fltPlateCenterX, fltPlateCenterY  
 intPlateWidth = int((listOfMatchingChars[len(listOfMatchingChars) - 1].intBoundingRectX + listOfMatchingChars[len(listOfMatchingChars) - 1].intBoundingRectWidth - listOfMatchingChars[0].intBoundingRectX) \* PLATE\_WIDTH\_PADDING\_FACTOR)  
  
 intTotalOfCharHeights = 0  
  
 for matchingChar in listOfMatchingChars:  
 intTotalOfCharHeights = intTotalOfCharHeights + matchingChar.intBoundingRectHeight  
 fltAverageCharHeight = intTotalOfCharHeights / len(listOfMatchingChars)  
  
 intPlateHeight = int(fltAverageCharHeight \* PLATE\_HEIGHT\_PADDING\_FACTOR)  
  
 fltOpposite = listOfMatchingChars[len(listOfMatchingChars) - 1].intCenterY - listOfMatchingChars[0].intCenterY  
 fltHypotenuse = distanceBetweenChars(listOfMatchingChars[0], listOfMatchingChars[len(listOfMatchingChars) - 1])  
 fltCorrectionAngleInRad = math.asin(fltOpposite / fltHypotenuse)  
 fltCorrectionAngleInDeg = fltCorrectionAngleInRad \* (180.0 / math.pi)  
  
 possiblePlate.rrLocationOfPlateInScene = ( tuple(ptPlateCenter), (intPlateWidth, intPlateHeight), fltCorrectionAngleInDeg )  
  
 rotationMatrix = cv2.getRotationMatrix2D(tuple(ptPlateCenter), fltCorrectionAngleInDeg, 1.0)  
  
 height, width, numChannels = imgOriginal.shape  
  
 imgRotated = cv2.warpAffine(imgOriginal, rotationMatrix, (width, height))  
  
 imgCropped = cv2.getRectSubPix(imgRotated, (intPlateWidth, intPlateHeight), tuple(ptPlateCenter))  
  
 possiblePlate.imgPlate = imgCropped  
  
 return possiblePlate  
  
  
root = Main()  
root.mainloop()