

**REPORT**

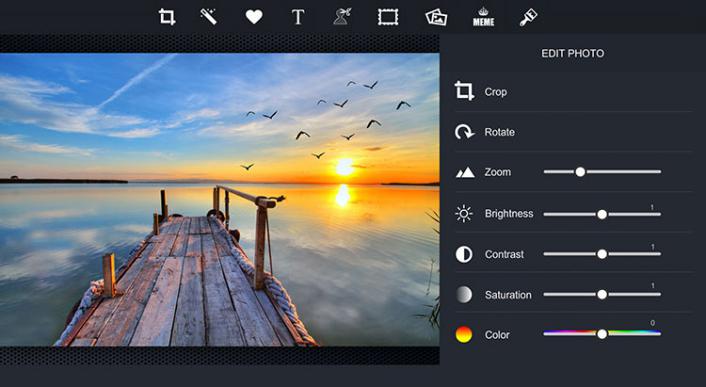
|  |  |
| --- | --- |
| **STUDENT NAME** | **: NURUL YASMINE BINTI MOHD YAZID YAP** |
| **MATRIC NUMBER** | **: BS18110359** |
| **FACULTY** | **: SCIENCE AND NATURAL RESOURCES** |
| **COURSE** | **: MATHEMATICS WITH COMPUTER GRAPHICS** |
| **LECTURER** | **: PROF. MADYA DR. ABDULLAH BIN BADE** |
| **SUBJECT** | **:SC32303 FUNDAMENTAL OF COMPUTER GRAPHICS** |
| **ASSIGNMENT TITLE** | **: IMAGE EDITOR** |
| **DATE OF SUBMISSION** | **:24/12/2020** |
| **SIGNED:** |  |

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| 1.INTRODUCTION | 3 |
| 2..FRAMEWORK AND ARCHITECTURE | 3 |
| 3.OUTPUT | 5 |
| 4.ADVANTAGES | 6 |
| 5.UNIQUENESS | 9 |
| 6.CONCLUSION | 9 |
| 7.SOURCE CODE | 10 |

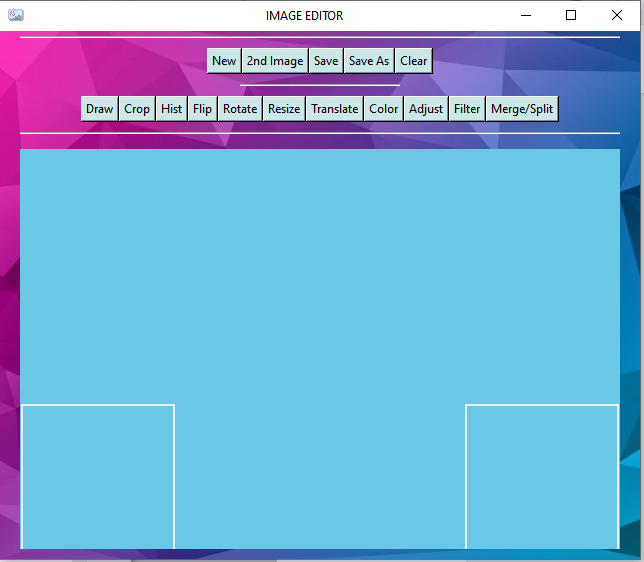
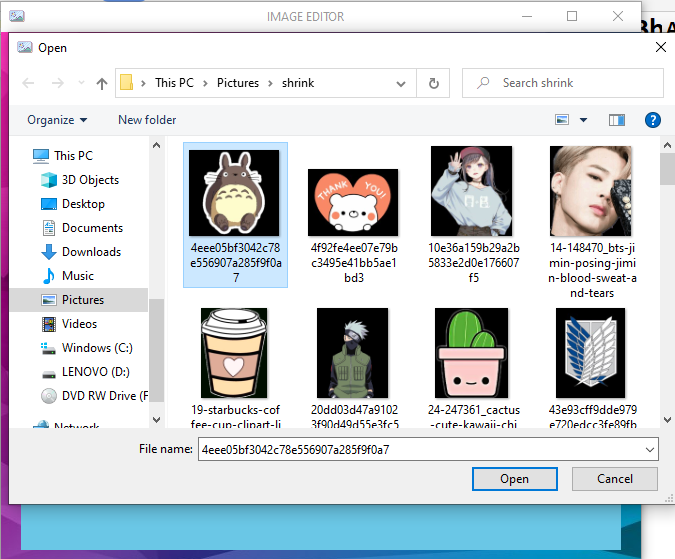
**INTRODUCTION**

Image Editor is written using basic opencv and tkinter in python. The concept is to build and develop an interactive 2D Image Manipulation Editor. The editor comprises several basic image manipulation features such as loading image, geometric transformation of image, color conversion such as gray scale to rgb color to hsv. Thus, the theme used is a simple paint adding only the color conversion features. The pictures below is two example of simple image editor.

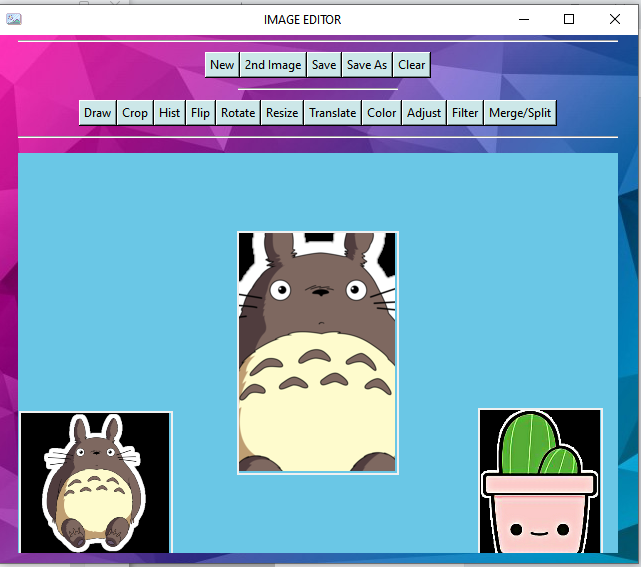


**FUNCTIONALITY**

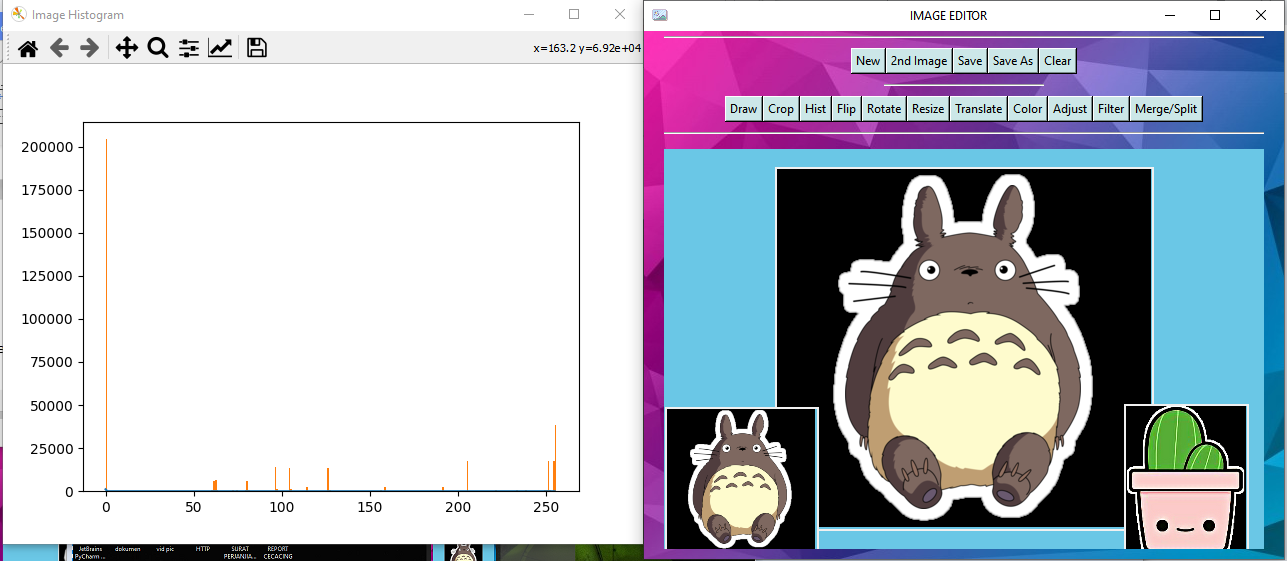
The application require user to insert image from computer to start using the button from the edit bar. The edit bar comprises of New, 2nd Image, Save, Save As, Clear, Draw, Crop, Hist, Rotate, Resize, Translate, Colour, Adjust, Filter, Merge/Split button button. The New, Save, Save As and Clear button 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. Button Draw allow user to draw red line on image using mouse motion. User also able to crop image using the Crop button using mouse motion. The button histogram will show a new window with the histogram of the image. Once selected, the Rotate button show a new window comprises of Rotate scale, Rotate button, Preview button and Cancel button. The Rotate button is used to rotate image using a scale starting from 0 to 360 degree. The user can preview the rotated image using Preview button. Once the desired rotated image is achieved, user can press the Rotate button to confirm the edited image. The Resize button enable user to resize the image according starting from a scale of 1 to 100 value for width and height,and apply the edited image once satisfied. Next, Translate button is used to move the object using a scale of x and y value that follows the image size. The colour button comprises of HSV and Black and White button. This allow user to change the image to HSV colour or Black and White colour. The Adjust button which comprises of brightness, red , green, and blue scale enable user to manipulate the image brightness and rgb colour. The scale starts at 0, the user able to to increase and decrease the value of the of brightness and rgb colour until 100. User able to enhance and restore image using the Filter button. The Filter button includes Negative, Sepia, Emboss, Gaussian Blur, Median Blur, Bilateral Blur, Average Blur, Box Filter Blur, Sharpen 1 and Sharpen 2 filter. The image can be split and merge using the Split/Merge button. The button comprises horizontal and vertical merge, horizontal and vertical split and splitImage which split the image into four parts. Image Segmentation is provided with four Edge Detection method techniques which is Canny, Laplacian, Sobel and Prewitt and two other image segmentation which is Threshold method and Cluster method. 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. 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. Numpy enable the image editor to translate the image.

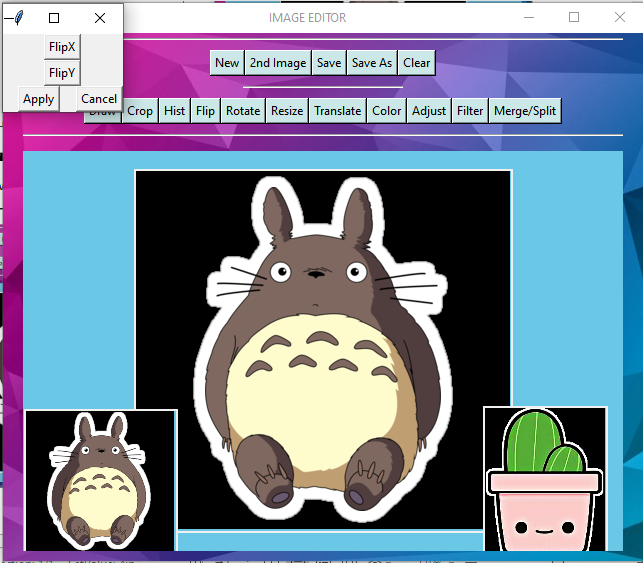
**FRAMEWORK AND ARCHITECTURE**

The picture shows the image editor without an image, new and 2nd Image button to load image from computer.

The picture shows the image editor after both image is loaded and a circle is drawn using draw button.

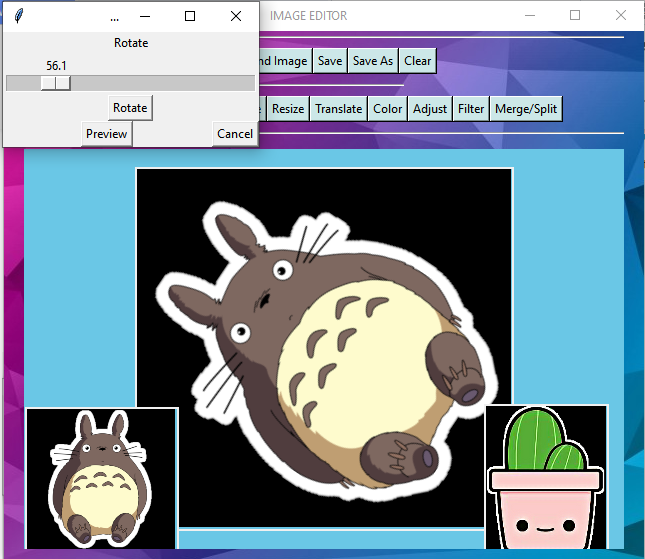
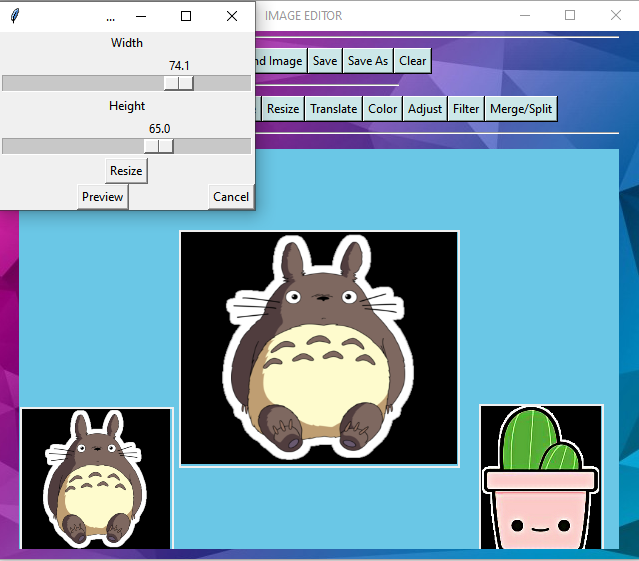
The picture show the image before and after the image is cropped.



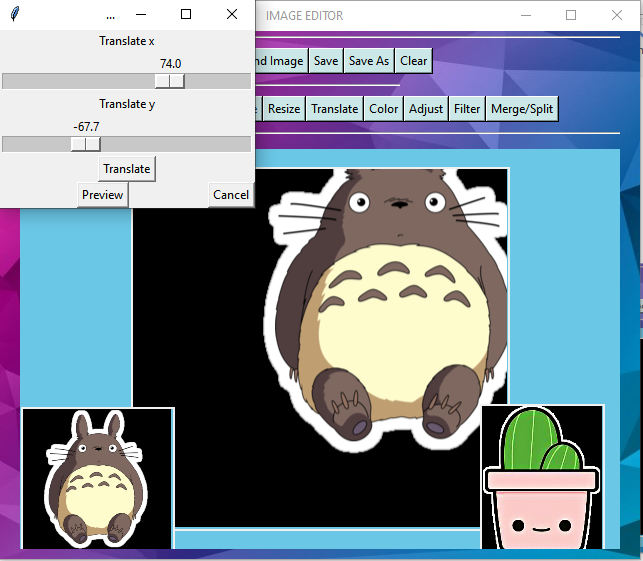
The picture shows an image histogram after the button hist was pressed.

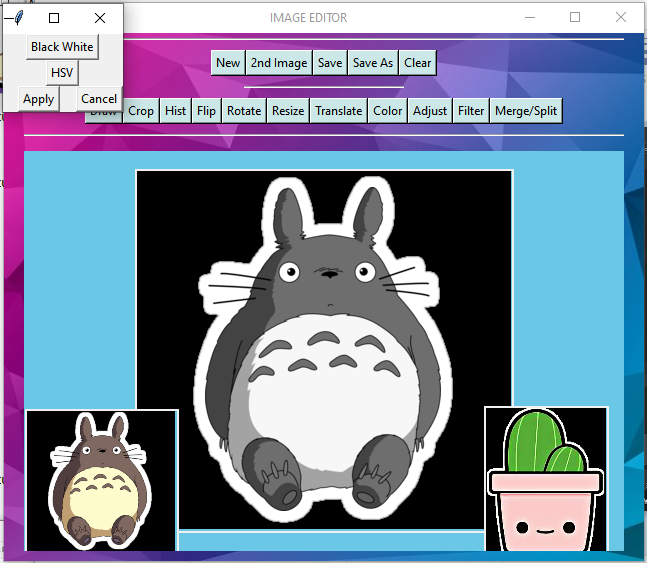
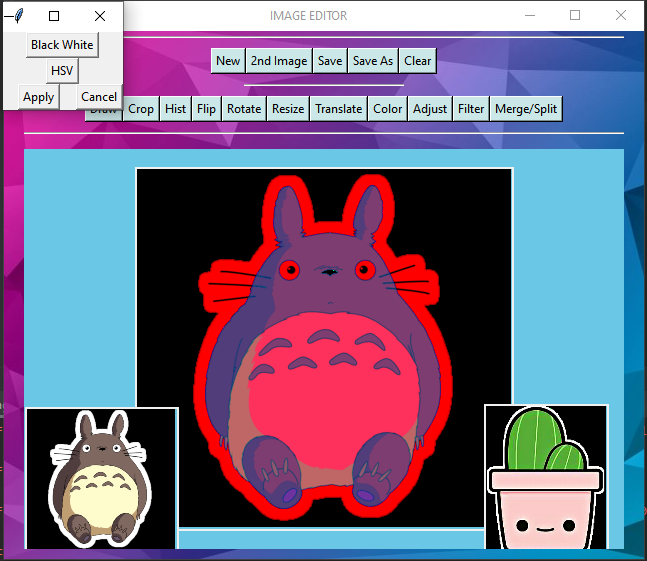


The picture shows the the image is mirrored using FlipX and FlipY respectively from the Flip button.

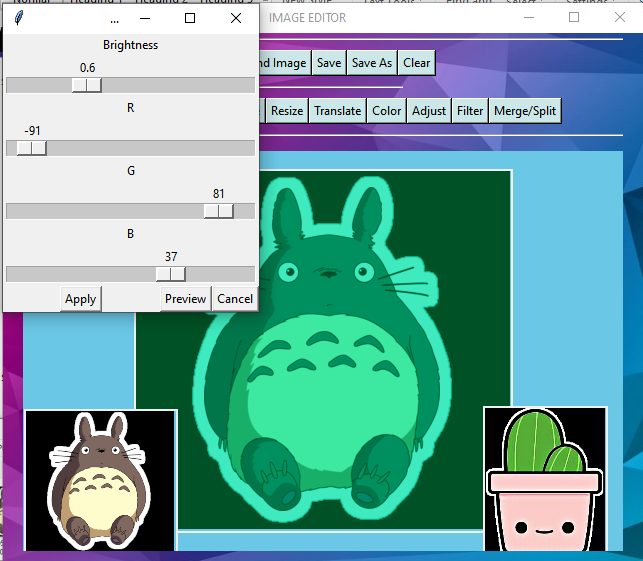
 

The picture shows a rotated and resized image usng the Rotate ad Resize button.

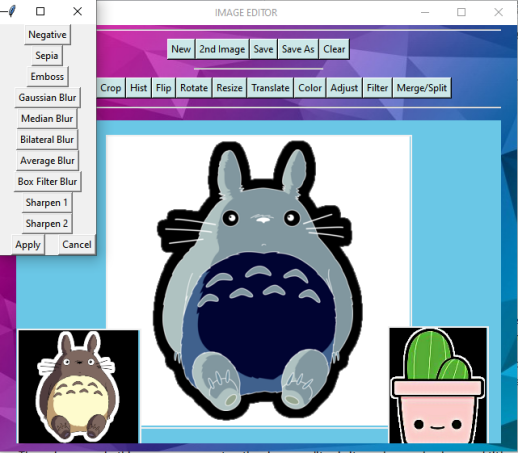
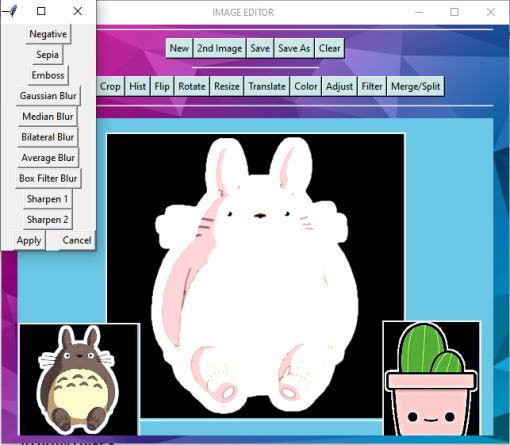
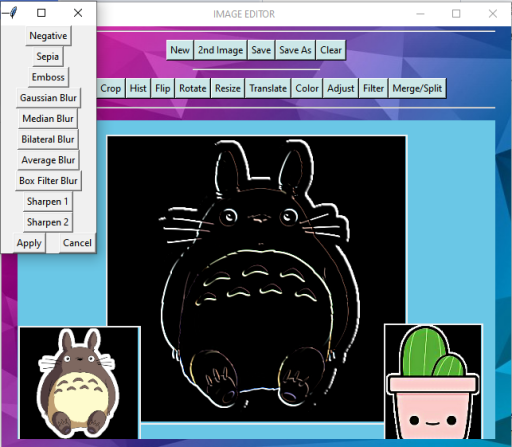


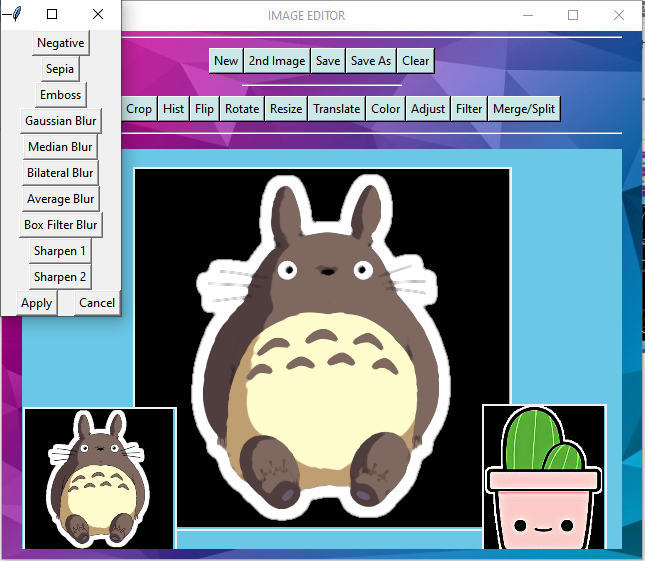
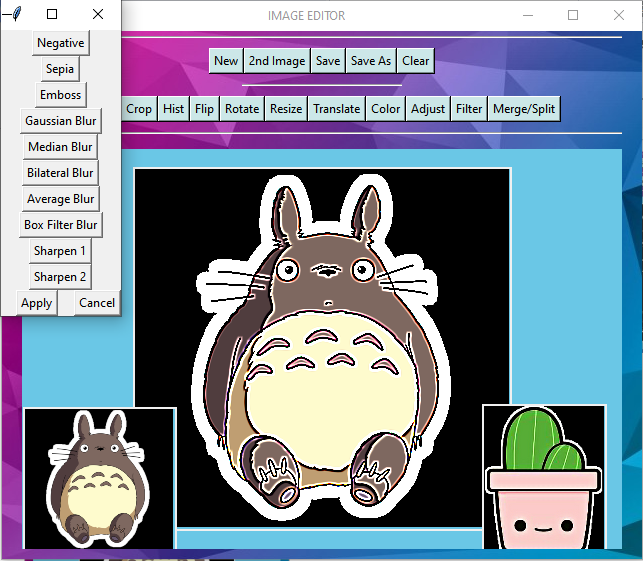
The picture show a translated image when move in x and y direction.

The picture show a Black and White and HSV edited image using Color button.

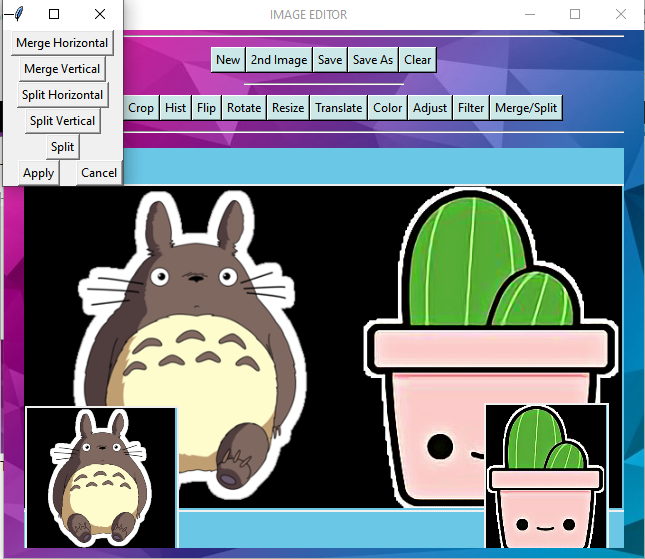
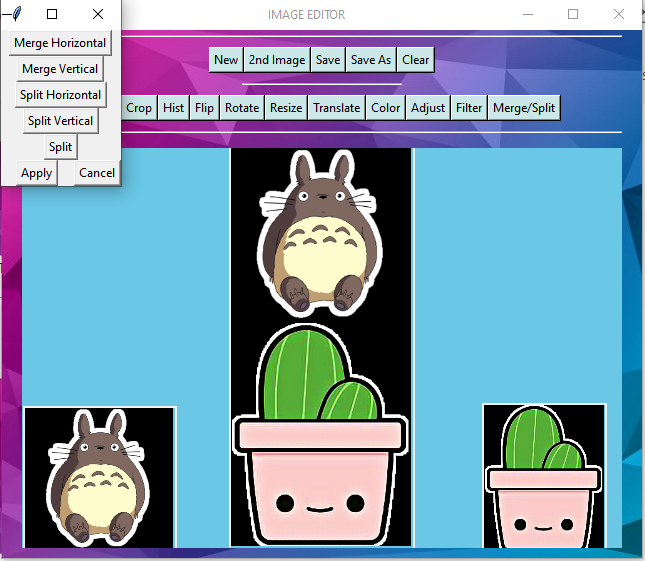


The picture show an edited image using adjust button.

The picture show filter image using three filter in filter button.



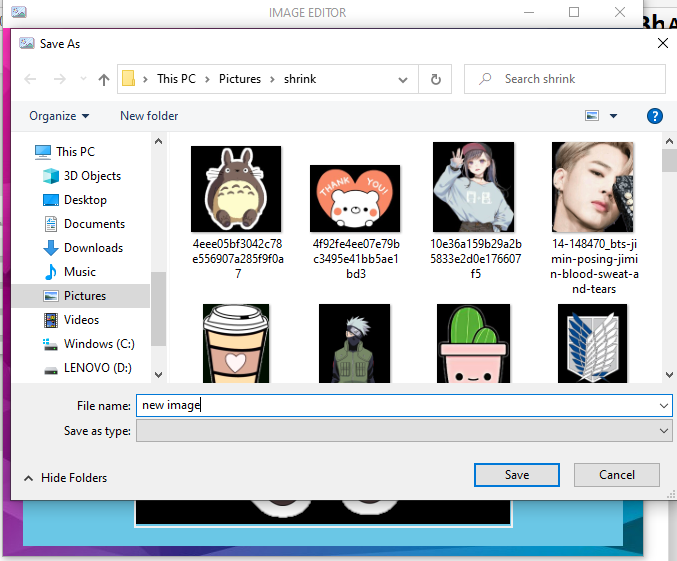
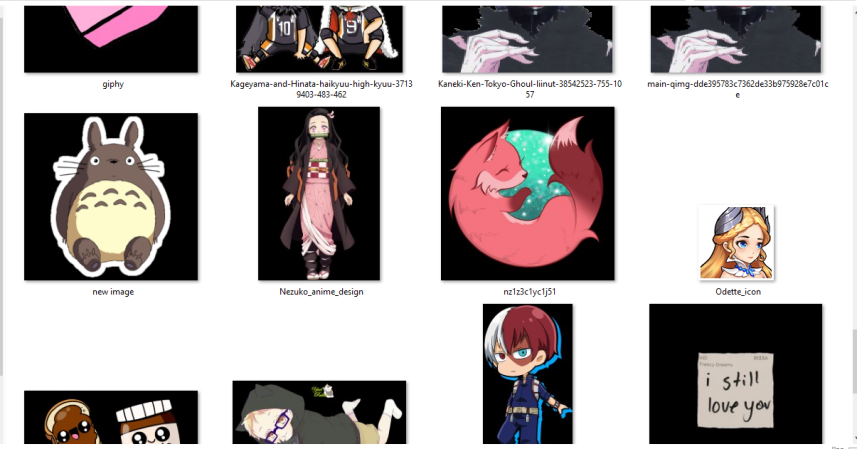
The picture show median blur,one of the five blur filter and sharpen1 filter,one of the two sharpen filter.

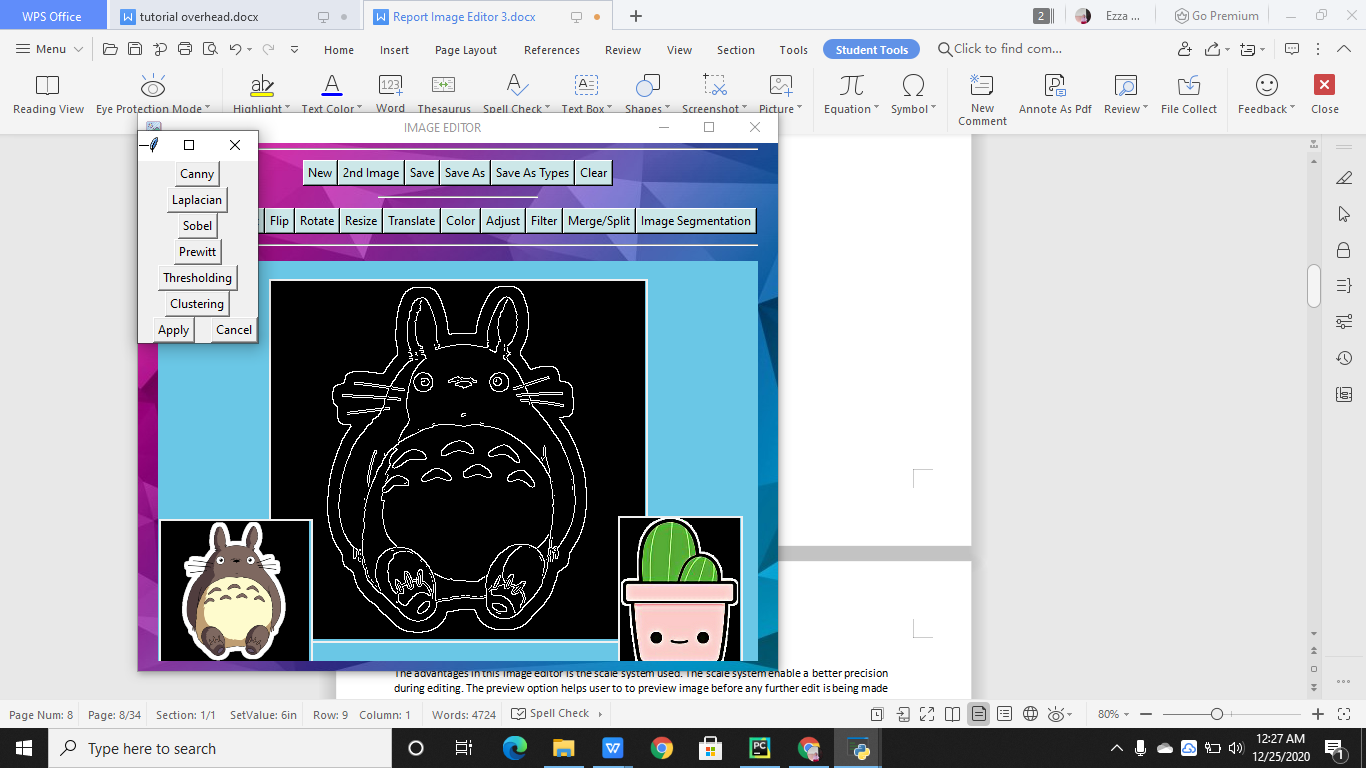
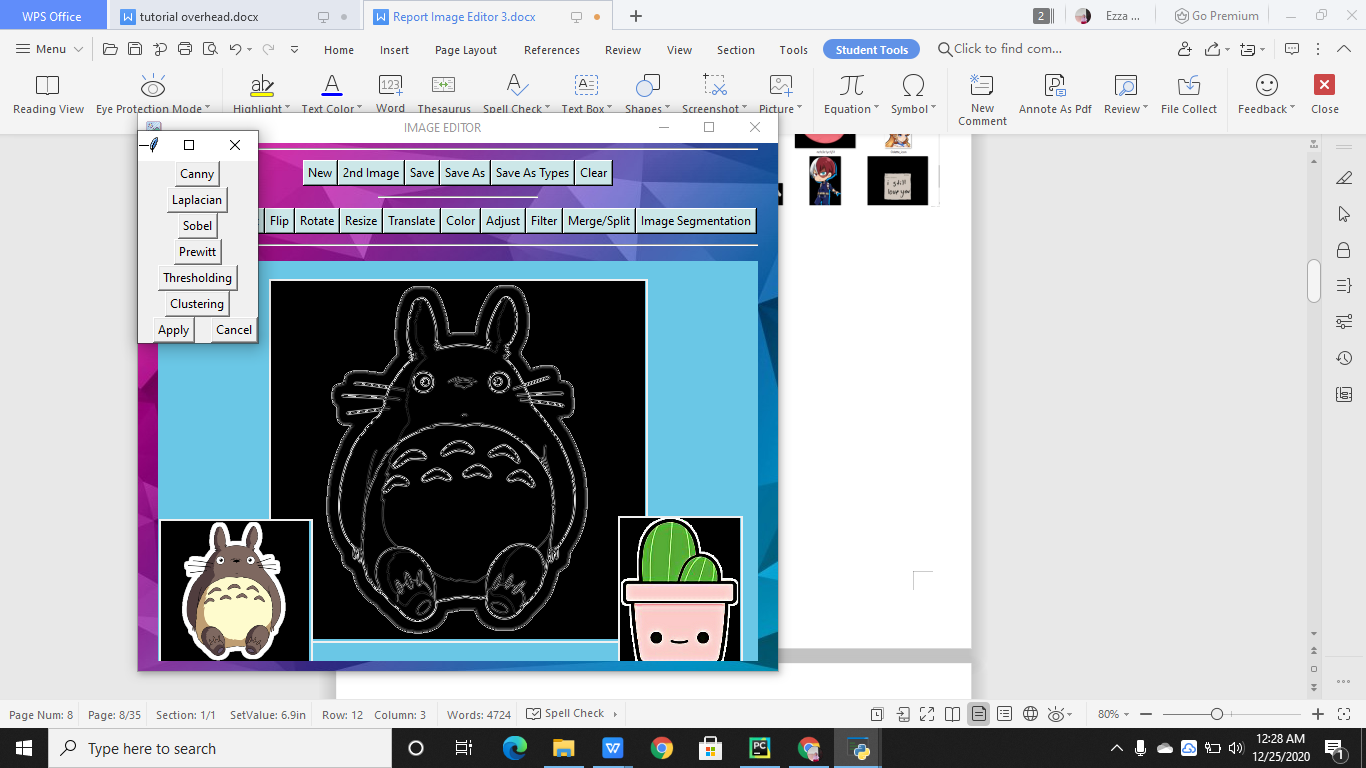
The picture show merge morizontal and merge vertical found in Merge/Split button.

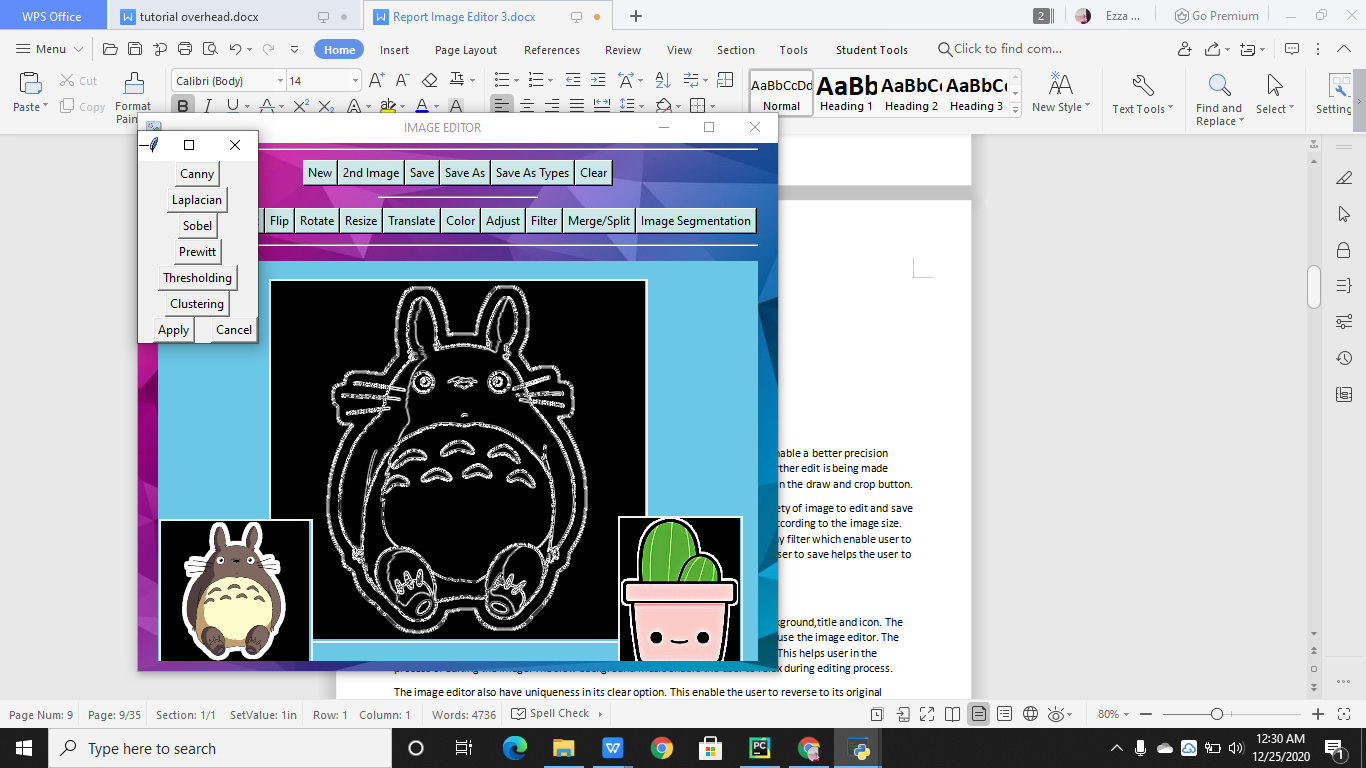
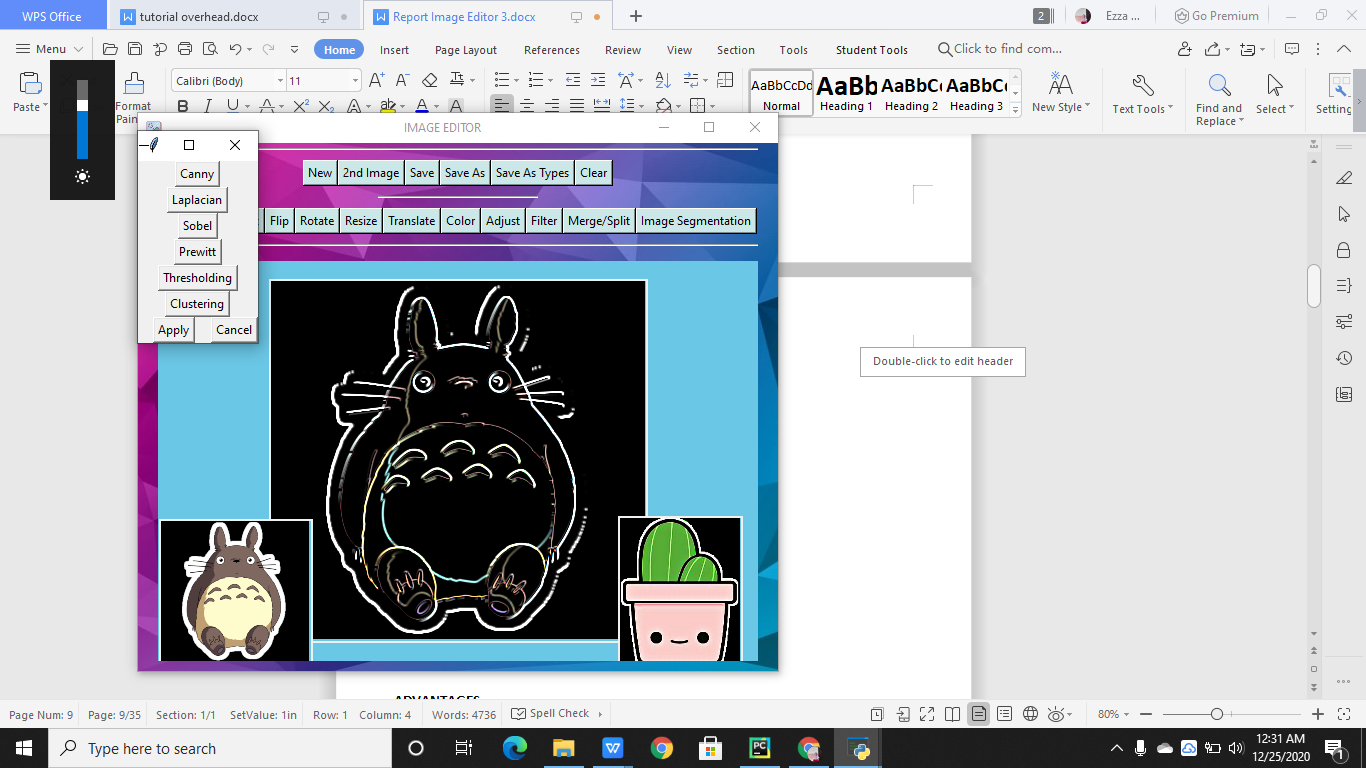
The picture show three split image using split horizontal,vertical and splitImage in Merge/Split button.



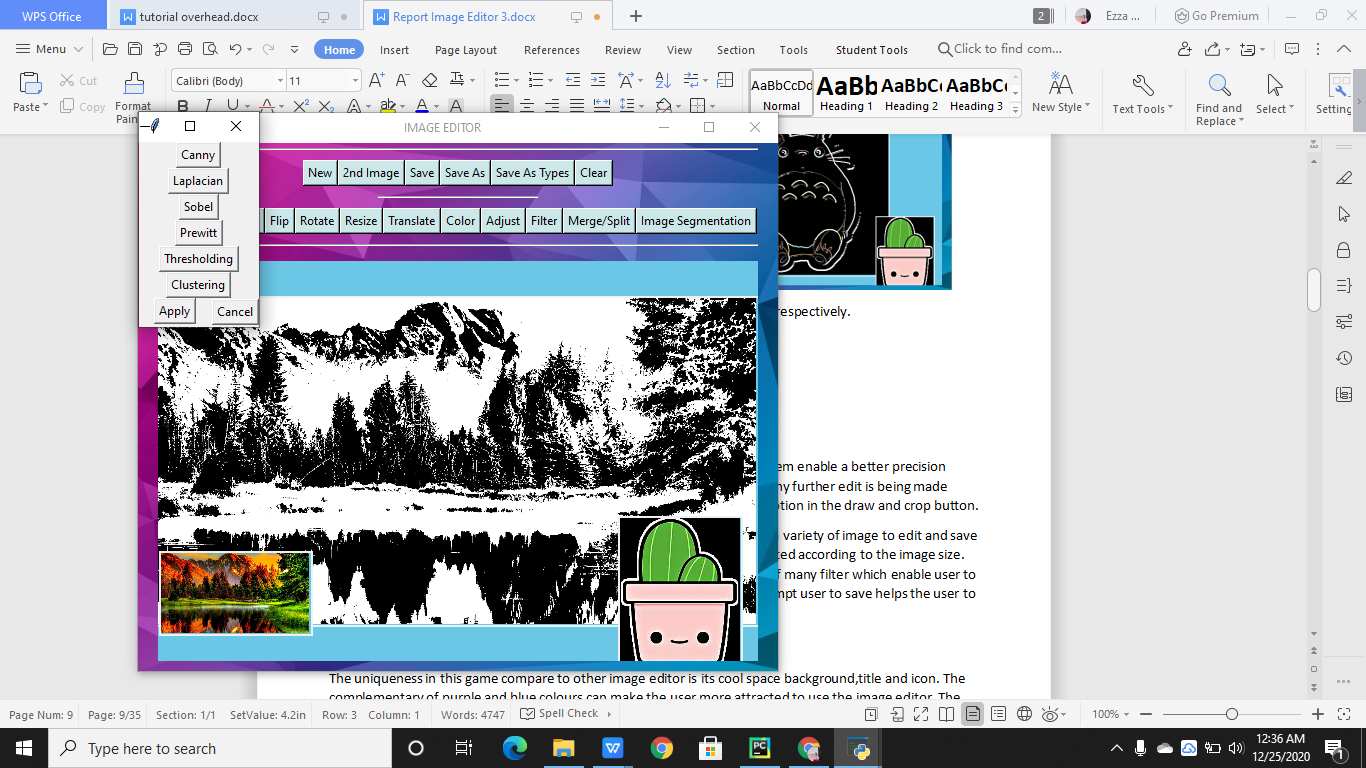
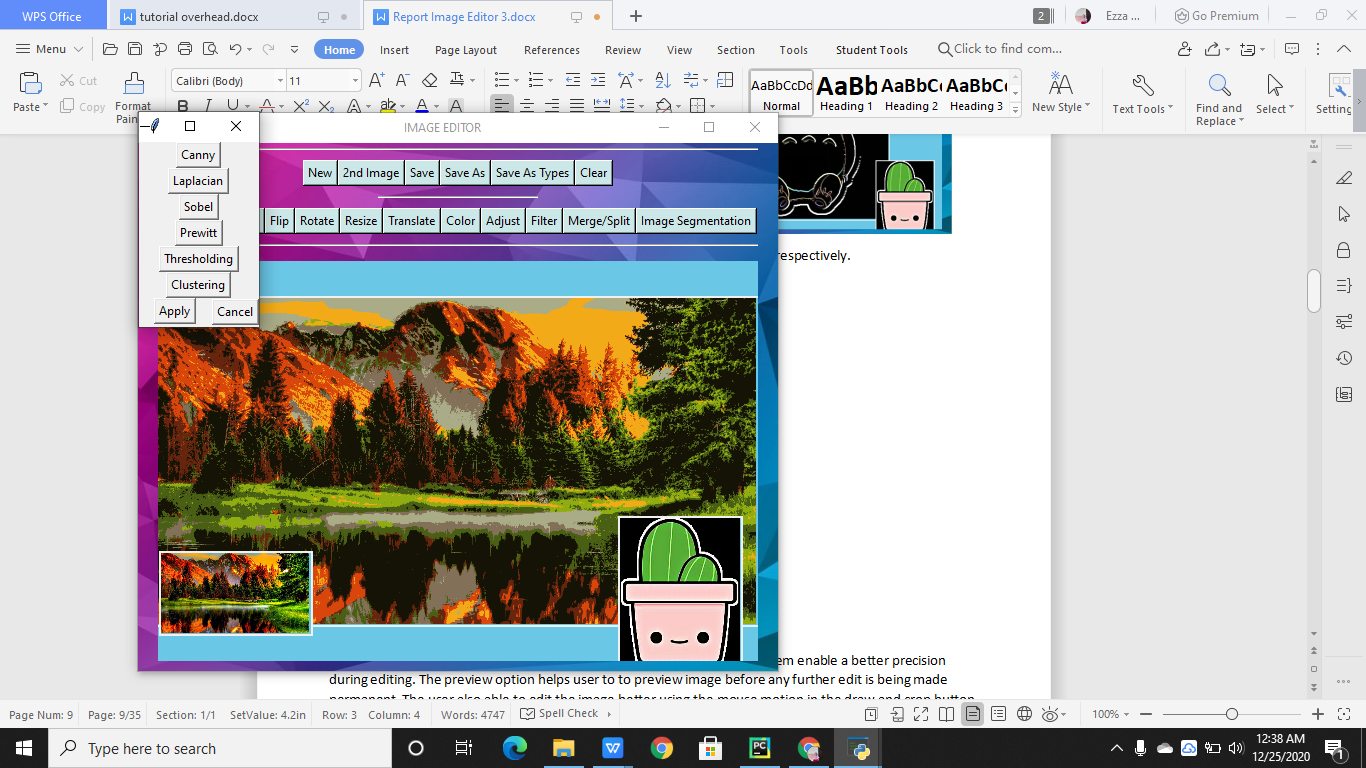
The picture show the Save As button used to save the image as ‘new image’.

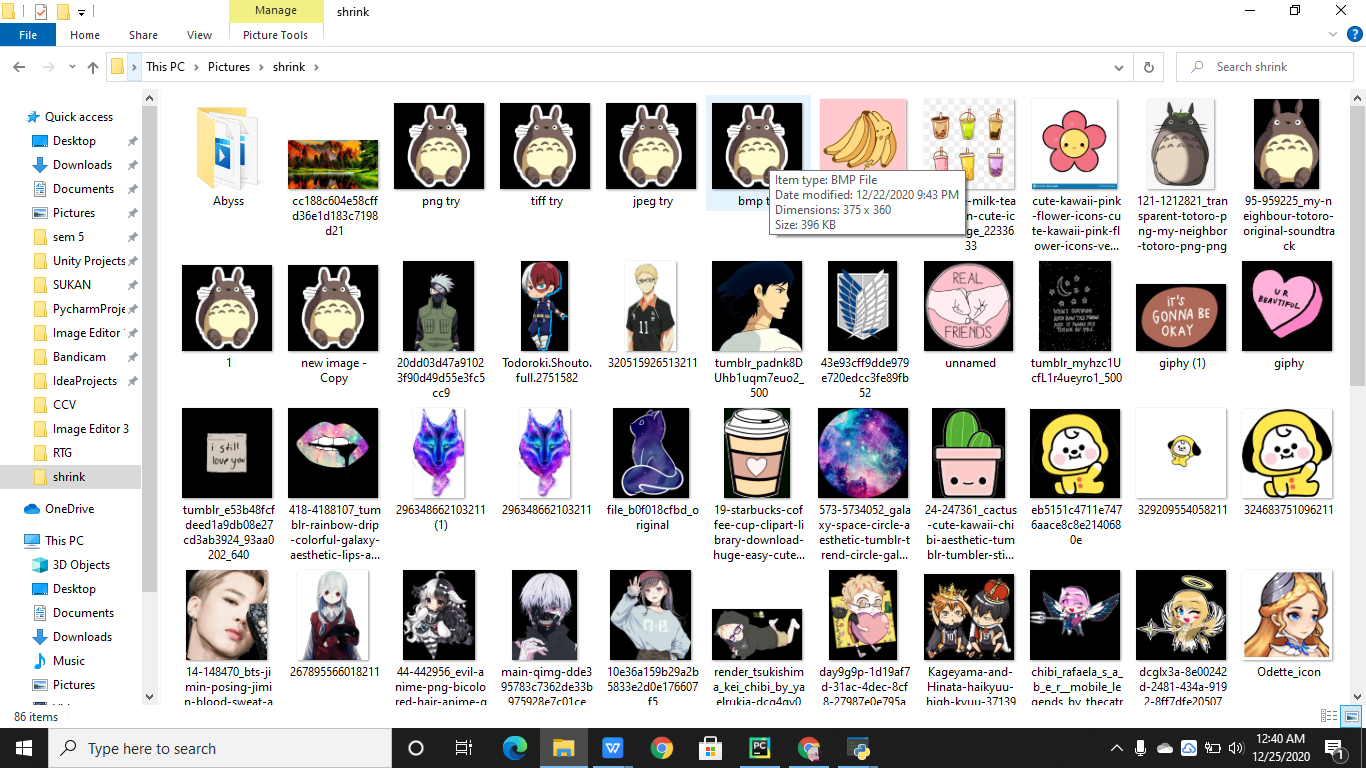
The picture show the Canny and Laplacian edge detection techniques application respectively.

The picture show Sobel and Prewitt edge detection techniques application respectively.

The picture shows Threshold and Cluster image segmentation method application respectively.



The picture shows PNG, TIFF, JPEG, BMP file types image with BMP file type properties shown as proof.

**ADVANTAGES**

The advantages in this image editor is the scale system used. The scale system enable a better precision during editing. The preview option helps user to to preview image before any further edit is being made permanent. The user also able to edit the image better using the mouse motion in the draw and 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 scale in the translate button is updated according to the image size. This allow for better translation to the image. The filter button comprises of many filter which enable user to have variety of choices. The split option which automatically show and prompt user to save helps the user to differentiate and choose which splited image to use. The advantages in this image editor is user able to choose to save different types of image file types.

**UNIQUENESS**

The uniqueness in this game compare to other image editor is its cool space background,title and icon. The complementary of purple and 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 LEFT  
from tkinter import filedialog  
from tkinter import Toplevel, Label, Scale, Button, HORIZONTAL, RIGHT  
from tkinter import Frame, Canvas, CENTER, ROUND  
from PIL import Image, ImageTk  
import matplotlib.pyplot as plt  
import pygame as pg  
import numpy as np  
import cv2  
  
  
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.rotated\_image = None  
 self.is\_image\_selected = False  
 self.is\_draw\_state = False  
 self.is\_crop\_state = False  
 self.is\_hist\_state = False  
  
 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('bg.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=240, pady=5)  
 self.editbar2.pack(pady=5)  
 separator2.pack(fill=tk.X, padx=20, pady=5)  
 self.image\_viewer.pack(fill=tk.BOTH, padx=20, pady=10, expand=1)  
  
  
class EditBar1(Frame):  
  
 def \_\_init\_\_(self, master=None):  
 Frame.\_\_init\_\_(self, master=master)  
  
 self.new\_button = Button(self, text="New", bg='#cce7e8', fg='#000000')  
 self.new2\_button = Button(self, text="2nd Image", bg='#cce7e8', fg='#000000')  
 self.save\_button = Button(self, text="Save", bg='#cce7e8', fg='#000000')  
 self.save\_as\_button = Button(self, text="Save As", bg='#cce7e8', fg='#000000')  
 self.save\_as\_type\_button = Button(self, text="Save As Types", bg='#cce7e8', fg='#000000')  
 self.clear\_button = Button(self, text="Clear", bg='#cce7e8', fg='#000000')  
  
 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\_draw\_state:  
 self.master.image\_viewer.deactivate\_draw()  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_hist\_state:  
 self.master.image\_viewer.deactivate\_hist()  
  
 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\_draw\_state:  
 self.master.image\_viewer.deactivate\_draw()  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_hist\_state:  
 self.master.image\_viewer.deactivate\_hist()  
  
 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\_draw\_state:  
 self.master.image\_viewer.deactivate\_draw()  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_hist\_state:  
 self.master.image\_viewer.deactivate\_hist()  
  
 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\_draw\_state:  
 self.master.image\_viewer.deactivate\_draw()  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_hist\_state:  
 self.master.image\_viewer.deactivate\_hist()  
  
 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\_draw\_state:  
 self.master.image\_viewer.deactivate\_draw()  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_hist\_state:  
 self.master.image\_viewer.deactivate\_hist()  
  
 self.master.save\_as\_type\_frame = TypeFrame(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\_draw\_state:  
 self.master.image\_viewer.deactivate\_draw()  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_hist\_state:  
 self.master.image\_viewer.deactivate\_hist()  
  
 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)  
  
 self.draw\_button = Button(self, text="Draw", bg='#cce7e8', fg='#000000')  
 self.crop\_button = Button(self, text="Crop", bg='#cce7e8', fg='#000000')  
 self.hist\_button = Button(self, text="Hist", bg='#cce7e8', fg='#000000')  
 self.flip\_button = Button(self, text="Flip", bg='#cce7e8', fg='#000000')  
 self.rotate\_button = Button(self, text="Rotate", bg='#cce7e8', fg='#000000')  
 self.resize\_button = Button(self, text="Resize", bg='#cce7e8', fg='#000000')  
 self.translate\_button = Button(self, text="Translate", bg='#cce7e8', fg='#000000')  
 self.color\_button = Button(self, text="Color", bg='#cce7e8', fg='#000000')  
 self.adjust\_button = Button(self, text="Adjust", bg='#cce7e8', fg='#000000')  
 self.filter\_button = Button(self, text="Filter", bg='#cce7e8', fg='#000000')  
 self.MergeSplit\_button = Button(self, text="Merge/Split", bg='#cce7e8', fg='#000000')  
 self.segment\_button = Button(self, text="Image Segmentation", bg='#cce7e8', fg='#000000')  
  
 self.draw\_button.bind("<ButtonRelease>", self.draw\_button\_released)  
 self.crop\_button.bind("<ButtonRelease>", self.crop\_button\_released)  
 self.hist\_button.bind("<ButtonRelease>", self.hist\_button\_released)  
 self.flip\_button.bind("<ButtonRelease>", self.flip\_button\_released)  
 self.rotate\_button.bind("<ButtonRelease>", self.rotate\_button\_released)  
 self.resize\_button.bind("<ButtonRelease>", self.resize\_button\_released)  
 self.translate\_button.bind("<ButtonRelease>", self.translate\_button\_released)  
 self.color\_button.bind("<ButtonRelease>", self.color\_button\_released)  
 self.adjust\_button.bind("<ButtonRelease>", self.adjust\_button\_released)  
 self.filter\_button.bind("<ButtonRelease>", self.filter\_button\_released)  
 self.MergeSplit\_button.bind("<ButtonRelease>", self.MergeSplit\_button\_released)  
 self.segment\_button.bind("<ButtonRelease>", self.segment\_button\_released)  
  
 self.draw\_button.pack(side=LEFT)  
 self.crop\_button.pack(side=LEFT)  
 self.hist\_button.pack(side=LEFT)  
 self.flip\_button.pack(side=LEFT)  
 self.rotate\_button.pack(side=LEFT)  
 self.resize\_button.pack(side=LEFT)  
 self.translate\_button.pack(side=LEFT)  
 self.color\_button.pack(side=LEFT)  
 self.adjust\_button.pack(side=LEFT)  
 self.filter\_button.pack(side=LEFT)  
 self.MergeSplit\_button.pack(side=LEFT)  
 self.segment\_button.pack(side=LEFT)  
  
 def draw\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.draw\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_draw\_state:  
 self.master.image\_viewer.deactivate\_draw()  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_hist\_state:  
 self.master.image\_viewer.deactivate\_hist()  
 else:  
 self.master.image\_viewer.activate\_draw()  
  
 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\_draw\_state:  
 self.master.image\_viewer.deactivate\_draw()  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_hist\_state:  
 self.master.image\_viewer.deactivate\_hist()  
 else:  
 self.master.image\_viewer.activate\_crop()  
  
 def flip\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.flip\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_draw\_state:  
 self.master.image\_viewer.deactivate\_draw()  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_hist\_state:  
 self.master.image\_viewer.deactivate\_hist()  
  
 self.master.flip\_frame = FlipFrame(master=self.master)  
 self.master.flip\_frame.grab\_set()  
  
 def rotate\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.rotate\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_draw\_state:  
 self.master.image\_viewer.deactivate\_draw()  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_hist\_state:  
 self.master.image\_viewer.deactivate\_hist()  
  
 self.master.rotate\_frame = RotateFrame(master=self.master)  
 self.master.rotate\_frame.grab\_set()  
  
 def resize\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.resize\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_draw\_state:  
 self.master.image\_viewer.deactivate\_draw()  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_hist\_state:  
 self.master.image\_viewer.deactivate\_hist()  
  
 self.master.resize\_frame = ResizeFrame(master=self.master)  
 self.master.resize\_frame.grab\_set()  
  
 def translate\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.translate\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_draw\_state:  
 self.master.image\_viewer.deactivate\_draw()  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_hist\_state:  
 self.master.image\_viewer.deactivate\_hist()  
  
 self.master.translate\_frame = TranslateFrame(master=self.master)  
 self.master.translate\_frame.grab\_set()  
  
 def color\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.color\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_draw\_state:  
 self.master.image\_viewer.deactivate\_draw()  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_hist\_state:  
 self.master.image\_viewer.deactivate\_hist()  
  
 self.master.color\_frame = ColorFrame(master=self.master)  
 self.master.color\_frame.grab\_set()  
  
 def hist\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.hist\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_draw\_state:  
 self.master.image\_viewer.deactivate\_draw()  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_hist\_state:  
 self.master.image\_viewer.deactivate\_hist()  
 else:  
 self.master.image\_viewer.activate\_hist()  
  
 def adjust\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.adjust\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_draw\_state:  
 self.master.image\_viewer.deactivate\_draw()  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_hist\_state:  
 self.master.image\_viewer.deactivate\_hist()  
  
 self.master.adjust\_frame = AdjustFrame(master=self.master)  
 self.master.adjust\_frame.grab\_set()  
  
 def filter\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.filter\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_draw\_state:  
 self.master.image\_viewer.deactivate\_draw()  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_hist\_state:  
 self.master.image\_viewer.deactivate\_hist()  
  
 self.master.filter\_frame = FilterFrame(master=self.master)  
 self.master.filter\_frame.grab\_set()  
  
 def MergeSplit\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.MergeSplit\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_draw\_state:  
 self.master.image\_viewer.deactivate\_draw()  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_hist\_state:  
 self.master.image\_viewer.deactivate\_hist()  
  
 self.master.MergeSplit\_frame = MergeSplitFrame(master=self.master)  
 self.master.MergeSplit\_frame.grab\_set()  
  
 def segment\_button\_released(self, event):  
 if self.winfo\_containing(event.x\_root, event.y\_root) == self.segment\_button:  
 if self.master.is\_image\_selected:  
 if self.master.is\_draw\_state:  
 self.master.image\_viewer.deactivate\_draw()  
 if self.master.is\_crop\_state:  
 self.master.image\_viewer.deactivate\_crop()  
 if self.master.is\_hist\_state:  
 self.master.image\_viewer.deactivate\_hist()  
  
 self.master.segment\_frame = SegmentFrame(master=self.master)  
 self.master.segment\_frame.grab\_set()  
  
  
class TypeFrame(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 RotateFrame(Toplevel):  
  
 def \_\_init\_\_(self, master=None):  
 Toplevel.\_\_init\_\_(self, master=master)  
  
 self.rotate\_value = 0  
 self.previous\_rotate\_value = 0  
  
 self.original\_image = self.master.processed\_image  
 self.rotated\_image = self.master.processed\_image  
  
 self.rotate\_label = Label(self, text="Rotate")  
 self.rotate\_scale = Scale(self, from\_=0, to\_=360, length=250, resolution=0.1, orient=HORIZONTAL)  
  
 self.rotate\_button = Button(master=self, text="Rotate")  
 self.preview\_button = Button(master=self, text="Preview")  
 self.cancel\_button = Button(master=self, text="Cancel")  
  
 self.rotate\_button.bind("<ButtonRelease>", self.rotate\_button\_released)  
 self.preview\_button.bind("<ButtonRelease>", self.show\_button\_released)  
 self.cancel\_button.bind("<ButtonRelease>", self.cancel\_button\_released)  
  
 self.rotate\_scale.set(0)  
  
 self.rotate\_label.pack()  
 self.rotate\_scale.pack()  
 self.rotate\_button.pack()  
 self.cancel\_button.pack(side=RIGHT)  
 self.preview\_button.pack()  
  
 def rotate\_button\_released(self, event):  
 self.master.processed\_image = self.rotated\_image  
 self.close()  
  
 def show\_button\_released(self, event):  
 scale = 1  
 rotate = self.rotate\_scale.get()  
 self.center = (self.original\_image.shape[1] / 2, self.original\_image.shape[0] / 2)  
 M = cv2.getRotationMatrix2D(self.center, rotate, scale)  
 self.rotated\_image = cv2.warpAffine(self.original\_image, M,  
 (self.original\_image.shape[1], self.original\_image.shape[0]))  
 self.show\_image(self.rotated\_image)  
  
 def cancel\_button\_released(self, event):  
 self.close()  
  
 def show\_image(self, img=None):  
 self.master.image\_viewer.show\_image(img=img)  
  
 def close(self):  
 self.show\_image()  
 self.destroy()  
  
  
class ResizeFrame(Toplevel):  
  
 def \_\_init\_\_(self, master=None):  
 Toplevel.\_\_init\_\_(self, master=master)  
  
 self.width\_value = 0  
 self.previous\_width\_value = 0  
 self.height\_value = 0  
 self.previous\_height\_value = 0  
  
 self.original\_image = self.master.processed\_image  
 self.resized\_image = self.master.processed\_image  
  
 self.width\_label = Label(self, text="Width")  
 self.width\_scale = Scale(self, from\_=1, to\_=100, length=250, resolution=0.1, orient=HORIZONTAL)  
 self.height\_label = Label(self, text="Height")  
 self.height\_scale = Scale(self, from\_=1, to\_=100, length=250, resolution=0.1, orient=HORIZONTAL)  
  
 self.resize\_button = Button(master=self, text="Resize")  
 self.preview\_button = Button(master=self, text="Preview")  
 self.cancel\_button = Button(master=self, text="Cancel")  
  
 self.resize\_button.bind("<ButtonRelease>", self.resize\_button\_released)  
 self.preview\_button.bind("<ButtonRelease>", self.show\_button\_released)  
 self.cancel\_button.bind("<ButtonRelease>", self.cancel\_button\_released)  
  
 self.width\_scale.set(100)  
 self.height\_scale.set(100)  
  
 self.width\_label.pack()  
 self.width\_scale.pack()  
 self.height\_label.pack()  
 self.height\_scale.pack()  
 self.resize\_button.pack()  
 self.cancel\_button.pack(side=RIGHT)  
 self.preview\_button.pack()  
  
 def resize\_button\_released(self, event):  
 self.master.processed\_image = self.resized\_image  
 self.close()  
  
 def show\_button\_released(self, event):  
 width = self.width\_scale.get()  
 height = self.height\_scale.get()  
 self.width = int(self.original\_image.shape[1] \* width / 100)  
 self.height = int(self.original\_image.shape[0] \* height / 100)  
 dim = (self.width, self.height)  
 self.resized\_image = cv2.resize(self.original\_image, dim)  
 self.show\_image(self.resized\_image)  
  
 def cancel\_button\_released(self, event):  
 self.close()  
  
 def show\_image(self, img=None):  
 self.master.image\_viewer.show\_image(img=img)  
  
 def close(self):  
 self.show\_image()  
 self.destroy()  
  
  
class TranslateFrame(Toplevel):  
  
 def \_\_init\_\_(self, master=None):  
 Toplevel.\_\_init\_\_(self, master=master)  
  
 self.tx\_value = 0  
 self.previous\_tx\_value = 0  
 self.ty\_value = 0  
 self.previous\_ty\_value = 0  
  
 self.original\_image = self.master.processed\_image  
 self.translated\_image = self.master.processed\_image  
  
 self.tx\_label = Label(self, text="Translate x")  
 self.tx\_scale = Scale(self, from\_=-(self.original\_image.shape[1] / 2), to\_=self.original\_image.shape[1] / 2,  
 length=250, resolution=0.1, orient=HORIZONTAL)  
 self.ty\_label = Label(self, text="Translate y")  
 self.ty\_scale = Scale(self, from\_=-(self.original\_image.shape[0] / 2), to\_=self.original\_image.shape[0] / 2,  
 length=250, resolution=0.1, orient=HORIZONTAL)  
  
 self.translate\_button = Button(master=self, text="Translate")  
 self.preview\_button = Button(master=self, text="Preview")  
 self.cancel\_button = Button(master=self, text="Cancel")  
  
 self.translate\_button.bind("<ButtonRelease>", self.translate\_button\_released)  
 self.preview\_button.bind("<ButtonRelease>", self.show\_button\_released)  
 self.cancel\_button.bind("<ButtonRelease>", self.cancel\_button\_released)  
  
 self.tx\_scale.set(0)  
 self.ty\_scale.set(0)  
  
 self.tx\_label.pack()  
 self.tx\_scale.pack()  
 self.ty\_label.pack()  
 self.ty\_scale.pack()  
 self.translate\_button.pack()  
 self.cancel\_button.pack(side=RIGHT)  
 self.preview\_button.pack()  
  
 def translate\_button\_released(self, event):  
 self.master.processed\_image = self.translated\_image  
 self.close()  
  
 def show\_button\_released(self, event):  
 tx = self.tx\_scale.get()  
 ty = self.ty\_scale.get()  
 translationMatrix = np.float32([[1.0, 0.0, tx], [0.0, 1.0, ty]])  
 self.translated\_image = cv2.warpAffine(self.original\_image, translationMatrix,  
 (self.original\_image.shape[1], self.original\_image.shape[0]))  
 self.show\_image(self.translated\_image)  
  
 def cancel\_button\_released(self, event):  
 self.close()  
  
 def show\_image(self, img=None):  
 self.master.image\_viewer.show\_image(img=img)  
  
 def close(self):  
 self.show\_image()  
 self.destroy()  
  
  
class ColorFrame(Toplevel):  
  
 def \_\_init\_\_(self, master=None):  
 Toplevel.\_\_init\_\_(self, master=master)  
  
 self.original\_image = self.master.processed\_image  
 self.filtered\_image = self.master.processed\_image  
  
 self.black\_white\_button = Button(master=self, text="Black White")  
 self.hsv\_button = Button(master=self, text="HSV")  
 self.cancel\_button = Button(master=self, text="Cancel")  
 self.apply\_button = Button(master=self, text="Apply")  
  
 self.black\_white\_button.bind("<ButtonRelease>", self.black\_white\_released)  
 self.hsv\_button.bind("<ButtonRelease>", self.hsv\_button\_released)  
 self.apply\_button.bind("<ButtonRelease>", self.apply\_button\_released)  
 self.cancel\_button.bind("<ButtonRelease>", self.cancel\_button\_released)  
  
 self.black\_white\_button.pack()  
 self.hsv\_button.pack()  
 self.cancel\_button.pack(side=RIGHT)  
 self.apply\_button.pack()  
  
 def black\_white\_released(self, event):  
 self.black\_white()  
 self.show\_image(self.filtered\_image)  
  
 def hsv\_button\_released(self, event):  
 self.hsv()  
 self.show\_image(self.filtered\_image)  
  
 def apply\_button\_released(self, event):  
 self.master.processed\_image = self.filtered\_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 black\_white(self):  
 self.filtered\_image = cv2.cvtColor(self.original\_image, cv2.COLOR\_BGR2GRAY)  
 self.filtered\_image = cv2.cvtColor(self.filtered\_image, cv2.COLOR\_GRAY2BGR)  
  
 def hsv(self):  
 self.filtered\_image = cv2.cvtColor(self.original\_image, cv2.COLOR\_RGB2HSV)  
  
 def close(self):  
 self.destroy()  
  
  
class FlipFrame(Toplevel):  
  
 def \_\_init\_\_(self, master=None):  
 Toplevel.\_\_init\_\_(self, master=master)  
  
 self.original\_image = self.master.processed\_image  
 self.flipped\_image = self.master.processed\_image  
  
 self.flipx\_button = Button(master=self, text="FlipX")  
 self.flipy\_button = Button(master=self, text="FlipY")  
 self.cancel\_button = Button(master=self, text="Cancel")  
 self.apply\_button = Button(master=self, text="Apply")  
  
 self.flipx\_button.bind("<ButtonRelease>", self.flipx\_button\_released)  
 self.flipy\_button.bind("<ButtonRelease>", self.flipy\_button\_released)  
 self.apply\_button.bind("<ButtonRelease>", self.apply\_button\_released)  
 self.cancel\_button.bind("<ButtonRelease>", self.cancel\_button\_released)  
  
 self.flipx\_button.pack()  
 self.flipy\_button.pack()  
 self.cancel\_button.pack(side=RIGHT)  
 self.apply\_button.pack()  
  
 def flipx\_button\_released(self, event):  
 self.flipx()  
 self.show\_image(self.flipped\_image)  
  
 def flipy\_button\_released(self, event):  
 self.flipy()  
 self.show\_image(self.flipped\_image)  
  
 def apply\_button\_released(self, event):  
 self.master.processed\_image = self.flipped\_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 flipx(self):  
 self.flipVertical = cv2.flip(self.master.processed\_image, 0)  
 self.flipped\_image = self.flipVertical  
  
 def flipy(self):  
 self.flipHorizontal = cv2.flip(self.master.processed\_image, 1)  
 self.flipped\_image = self.flipHorizontal  
  
 def close(self):  
 self.destroy()  
  
  
class AdjustFrame(Toplevel):  
  
 def \_\_init\_\_(self, master=None):  
 Toplevel.\_\_init\_\_(self, master=master)  
  
 self.brightness\_value = 0  
 self.previous\_brightness\_value = 0  
  
 self.original\_image = self.master.processed\_image  
 self.processing\_image = self.master.processed\_image  
  
 self.brightness\_label = Label(self, text="Brightness")  
 self.brightness\_scale = Scale(self, from\_=0, to\_=2, length=250, resolution=0.1,  
 orient=HORIZONTAL)  
 self.r\_label = Label(self, text="R")  
 self.r\_scale = Scale(self, from\_=-100, to\_=100, length=250, resolution=1,  
 orient=HORIZONTAL)  
 self.g\_label = Label(self, text="G")  
 self.g\_scale = Scale(self, from\_=-100, to\_=100, length=250, resolution=1,  
 orient=HORIZONTAL)  
 self.b\_label = Label(self, text="B")  
 self.b\_scale = Scale(self, from\_=-100, to\_=100, length=250, resolution=1,  
 orient=HORIZONTAL)  
 self.b\_label = Label(self, text="B")  
 self.b\_scale = Scale(self, from\_=-100, to\_=100, length=250, resolution=1,  
 orient=HORIZONTAL)  
 self.apply\_button = Button(self, text="Apply")  
 self.preview\_button = Button(self, text="Preview")  
 self.cancel\_button = Button(self, text="Cancel")  
  
 self.brightness\_scale.set(1)  
  
 self.apply\_button.bind("<ButtonRelease>", self.apply\_button\_released)  
 self.preview\_button.bind("<ButtonRelease>", self.show\_button\_release)  
 self.cancel\_button.bind("<ButtonRelease>", self.cancel\_button\_released)  
  
 self.brightness\_label.pack()  
 self.brightness\_scale.pack()  
 self.r\_label.pack()  
 self.r\_scale.pack()  
 self.g\_label.pack()  
 self.g\_scale.pack()  
 self.b\_label.pack()  
 self.b\_scale.pack()  
 self.cancel\_button.pack(side=RIGHT)  
 self.preview\_button.pack(side=RIGHT)  
 self.apply\_button.pack()  
  
 def apply\_button\_released(self, event):  
 self.master.processed\_image = self.processing\_image  
 self.close()  
  
 def show\_button\_release(self, event):  
 self.processing\_image = cv2.convertScaleAbs(self.original\_image, alpha=self.brightness\_scale.get())  
 b, g, r = cv2.split(self.processing\_image)  
  
 for b\_value in b:  
 cv2.add(b\_value, self.b\_scale.get(), b\_value)  
 for g\_value in g:  
 cv2.add(g\_value, self.g\_scale.get(), g\_value)  
 for r\_value in r:  
 cv2.add(r\_value, self.r\_scale.get(), r\_value)  
  
 self.processing\_image = cv2.merge((b, g, r))  
 self.show\_image(self.processing\_image)  
  
 def cancel\_button\_released(self, event):  
 self.close()  
  
 def show\_image(self, img=None):  
 self.master.image\_viewer.show\_image(img=img)  
  
 def close(self):  
 self.show\_image()  
 self.destroy()  
  
  
class FilterFrame(Toplevel):  
  
 def \_\_init\_\_(self, master=None):  
 Toplevel.\_\_init\_\_(self, master=master)  
  
 self.original\_image = self.master.processed\_image  
 self.filtered\_image = None  
  
 self.negative\_button = Button(master=self, text="Negative")  
 self.sepia\_button = Button(master=self, text="Sepia")  
 self.emboss\_button = Button(master=self, text="Emboss")  
 self.gaussian\_blur\_button = Button(master=self, text="Gaussian Blur")  
 self.median\_blur\_button = Button(master=self, text="Median Blur")  
 self.bilateral\_button = Button(master=self, text="Bilateral Blur")  
 self.average\_button = Button(master=self, text="Average Blur")  
 self.boxFilter\_button = Button(master=self, text="Box Filter Blur")  
 self.sharpen1\_button = Button(master=self, text="Sharpen 1")  
 self.sharpen2\_button = Button(master=self, text="Sharpen 2")  
 self.cancel\_button = Button(master=self, text="Cancel")  
 self.apply\_button = Button(master=self, text="Apply")  
  
 self.negative\_button.bind("<ButtonRelease>", self.negative\_button\_released)  
 self.sepia\_button.bind("<ButtonRelease>", self.sepia\_button\_released)  
 self.emboss\_button.bind("<ButtonRelease>", self.emboss\_button\_released)  
 self.gaussian\_blur\_button.bind("<ButtonRelease>", self.gaussian\_blur\_button\_released)  
 self.median\_blur\_button.bind("<ButtonRelease>", self.median\_blur\_button\_released)  
 self.bilateral\_button.bind("<ButtonRelease>", self.bilateral\_button\_released)  
 self.average\_button.bind("<ButtonRelease>", self.average\_button\_released)  
 self.boxFilter\_button.bind("<ButtonRelease>", self.boxFilter\_button\_released)  
 self.sharpen1\_button.bind("<ButtonRelease>", self.sharpen1\_button\_released)  
 self.sharpen2\_button.bind("<ButtonRelease>", self.sharpen2\_button\_released)  
 self.apply\_button.bind("<ButtonRelease>", self.apply\_button\_released)  
 self.cancel\_button.bind("<ButtonRelease>", self.cancel\_button\_released)  
  
 self.negative\_button.pack()  
 self.sepia\_button.pack()  
 self.emboss\_button.pack()  
 self.gaussian\_blur\_button.pack()  
 self.median\_blur\_button.pack()  
 self.bilateral\_button.pack()  
 self.average\_button.pack()  
 self.boxFilter\_button.pack()  
 self.sharpen1\_button.pack()  
 self.sharpen2\_button.pack()  
 self.cancel\_button.pack(side=RIGHT)  
 self.apply\_button.pack()  
  
 def sharpen1\_button\_released(self, event):  
 self.sharpen1\_blur()  
 self.show\_image()  
  
 def sharpen2\_button\_released(self, event):  
 self.sharpen2\_blur()  
 self.show\_image()  
  
 def bilateral\_button\_released(self, event):  
 self.bilateral\_blur()  
 self.show\_image()  
  
 def average\_button\_released(self, event):  
 self.average\_blur()  
 self.show\_image()  
  
 def boxFilter\_button\_released(self, event):  
 self.boxFilter\_blur()  
 self.show\_image()  
  
 def sepia\_button\_released(self, event):  
 self.sepia()  
 self.show\_image()  
  
 def emboss\_button\_released(self, event):  
 self.emboss()  
 self.show\_image()  
  
 def negative\_button\_released(self, event):  
 self.negative()  
 self.show\_image()  
  
 def gaussian\_blur\_button\_released(self, event):  
 self.gaussian\_blur()  
 self.show\_image()  
  
 def median\_blur\_button\_released(self, event):  
 self.median\_blur()  
 self.show\_image()  
  
 def apply\_button\_released(self, event):  
 self.master.processed\_image = self.filtered\_image  
 self.show\_image()  
 self.close()  
  
 def cancel\_button\_released(self, event):  
 self.master.image\_viewer.show\_image()  
 self.close()  
  
 def show\_image(self):  
 self.master.image\_viewer.show\_image(img=self.filtered\_image)  
  
 def negative(self):  
 self.filtered\_image = cv2.bitwise\_not(self.original\_image)  
  
 def sepia(self):  
 kernel = np.array([[0.272, 0.534, 0.131],  
 [0.349, 0.686, 0.168],  
 [0.393, 0.769, 0.189]])  
  
 self.filtered\_image = cv2.filter2D(self.original\_image, -1, kernel)  
  
 def emboss(self):  
 kernel = np.array([[0, -1, -1],  
 [1, 0, -1],  
 [1, 1, 0]])  
  
 self.filtered\_image = cv2.filter2D(self.original\_image, -1, kernel)  
  
 def gaussian\_blur(self):  
 self.filtered\_image = cv2.GaussianBlur(self.original\_image, (5, 5), 3)  
  
 def median\_blur(self):  
 self.filtered\_image = cv2.medianBlur(self.original\_image, 5)  
  
 def bilateral\_blur(self):  
 self.filtered\_image = cv2.bilateralFilter(self.original\_image, 9, 75, 75)  
  
 def average\_blur(self):  
 self.filtered\_image = cv2.blur(self.original\_image, (5, 5))  
  
 def boxFilter\_blur(self):  
 self.filtered\_image = cv2.boxFilter(self.original\_image, 0, (7, 7))  
  
 def sharpen1\_blur(self):  
 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]])  
 self.filtered\_image = cv2.filter2D(self.original\_image, -1, kernel\_sharpening)  
  
 def sharpen2\_blur(self):  
 kernel\_sharpening = np.array([[-1, -1, -1], [-1, 10, -1], [-1, -1, -1]])  
 self.filtered\_image = cv2.filter2D(self.original\_image, -1, kernel\_sharpening)  
  
 def close(self):  
 self.destroy()  
  
  
class MergeSplitFrame(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 Horizontal")  
 self.mergev\_button = Button(master=self, text="Merge Vertical")  
 self.splith\_button = Button(master=self, text="Split Horizontal")  
 self.splitv\_button = Button(master=self, text="Split Vertical")  
 self.splitImage\_button = Button(master=self, text="Split")  
 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.splith\_button.bind("<ButtonRelease>", self.splith\_button\_released)  
 self.splitv\_button.bind("<ButtonRelease>", self.splitv\_button\_released)  
 self.splitImage\_button.bind("<ButtonRelease>", self.splitImage\_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.splith\_button.pack()  
 self.splitv\_button.pack()  
 self.splitImage\_button.pack()  
 self.cancel\_button.pack(side=RIGHT)  
 self.apply\_button.pack()  
  
 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 splith\_button\_released(self, event):  
 self.splith()  
 self.show\_image(self.edited\_image)  
  
 def splitv\_button\_released(self, event):  
 self.splitv()  
 self.show\_image(self.edited\_image)  
  
 def splitImage\_button\_released(self, event):  
 self.splitImage()  
 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 splith(self):  
 width = self.original\_image.shape[1]  
 height = self.original\_image.shape[0]  
  
 x = slice(0, width, 1)  
  
 y1 = slice(0, int(height / 2), 1)  
 y2 = slice(int(height / 2), height, 1)  
  
 # ..........................................................  
 cv2.imshow("split horizontal 1", self.original\_image[y1, x])  
 cv2.moveWindow("split horizontal 1", 0, 0)  
  
 original\_file\_type = self.master.filename.split('.')[-1]  
 filename = filedialog.asksaveasfilename()  
 filename = filename + "." + original\_file\_type  
 save\_image = self.original\_image[y1, x]  
 cv2.imwrite(filename, save\_image)  
 self.master.filename = filename  
  
 # ..........................................................  
 cv2.imshow("split horizontal 2", self.original\_image[y2, x])  
 cv2.moveWindow("split horizontal 2", 0, int(height / 2), )  
  
 original\_file\_type = self.master.filename.split('.')[-1]  
 filename = filedialog.asksaveasfilename()  
 filename = filename + "." + original\_file\_type  
  
 save\_image = self.original\_image[y2, x]  
 cv2.imwrite(filename, save\_image)  
  
 self.master.filename = filename  
  
 def splitv(self):  
 width = self.original\_image.shape[1]  
 height = self.original\_image.shape[0]  
  
 y = slice(0, height, 1)  
  
 x1 = slice(0, int(width / 2), 1)  
 x2 = slice(int(width / 2), width, 1)  
  
 # ........................................................  
 cv2.imshow("split vertical 1", self.original\_image[y, x1])  
 cv2.moveWindow("split vertical 1", 0, 0)  
  
 original\_file\_type = self.master.filename.split('.')[-1]  
 filename = filedialog.asksaveasfilename()  
 filename = filename + "." + original\_file\_type  
 save\_image = self.original\_image[y, x1]  
 cv2.imwrite(filename, save\_image)  
 self.master.filename = filename  
  
 # ........................................................  
 cv2.imshow("split vertical 2", self.original\_image[y, x2])  
 cv2.moveWindow("split vertical 2", int(width / 2), 0)  
  
 original\_file\_type = self.master.filename.split('.')[-1]  
 filename = filedialog.asksaveasfilename()  
 filename = filename + "." + original\_file\_type  
 save\_image = self.original\_image[y, x2]  
 cv2.imwrite(filename, save\_image)  
 self.master.filename = filename  
  
 def splitImage(self):  
 width = self.original\_image.shape[1]  
 height = self.original\_image.shape[0]  
  
 x = slice(0, width, 1)  
 y = slice(0, height, 1)  
  
 x1 = slice(0, int(width / 2), 1)  
 y1 = slice(0, int(height / 2), 1)  
 x2 = slice(int(width / 2), width, 1)  
 y2 = slice(int(height / 2), height, 1)  
  
 # .........................................................  
 cv2.imshow("Split lower left", self.original\_image[y2, x1])  
 cv2.moveWindow("Split lower left", 0, int(height / 2))  
  
 original\_file\_type = self.master.filename.split('.')[-1]  
 filename = filedialog.asksaveasfilename()  
 filename = filename + "." + original\_file\_type  
 save\_image = self.original\_image[y2, x1]  
 cv2.imwrite(filename, save\_image)  
 self.master.filename = filename  
  
 # .........................................................  
 cv2.imshow("split lower right", self.original\_image[y2, x2])  
 cv2.moveWindow("split lower right", int(width / 2), int(height / 2))  
  
 original\_file\_type = self.master.filename.split('.')[-1]  
 filename = filedialog.asksaveasfilename()  
 filename = filename + "." + original\_file\_type  
 save\_image = self.original\_image[y2, x2]  
 cv2.imwrite(filename, save\_image)  
 self.master.filename = filename  
  
 # .........................................................  
 cv2.imshow("split upper left", self.original\_image[y1, x1])  
 cv2.moveWindow("split upper left", 0, 0)  
  
 original\_file\_type = self.master.filename.split('.')[-1]  
 filename = filedialog.asksaveasfilename()  
 filename = filename + "." + original\_file\_type  
 save\_image = self.original\_image[y1, x1]  
 cv2.imwrite(filename, save\_image)  
 self.master.filename = filename  
  
 # .........................................................  
 cv2.imshow("split upper right", self.original\_image[y1, x2])  
 cv2.moveWindow("split upper right", int(width / 2), 0)  
  
 original\_file\_type = self.master.filename.split('.')[-1]  
 filename = filedialog.asksaveasfilename()  
 filename = filename + "." + original\_file\_type  
 save\_image = self.original\_image[y1, x2]  
 cv2.imwrite(filename, save\_image)  
 self.master.filename = filename  
  
 def close(self):  
 self.destroy()  
  
  
class SegmentFrame(Toplevel):  
  
 def \_\_init\_\_(self, master=None):  
 Toplevel.\_\_init\_\_(self, master=master)  
  
 self.original\_image = self.master.processed\_image  
 self.filtered\_image = None  
  
 self.canny\_button = Button(master=self, text="Canny")  
 self.laplacian\_button = Button(master=self, text="Laplacian")  
 self.sobel\_button = Button(master=self, text="Sobel")  
 self.prewitt\_button = Button(master=self, text="Prewitt")  
 self.threshold\_button = Button(master=self, text="Thresholding")  
 self.cluster\_button = Button(master=self, text="Clustering")  
 self.cancel\_button = Button(master=self, text="Cancel")  
 self.apply\_button = Button(master=self, text="Apply")  
  
 self.canny\_button.bind("<ButtonRelease>", self.canny\_button\_released)  
 self.laplacian\_button.bind("<ButtonRelease>", self.laplacian\_button\_released)  
 self.sobel\_button.bind("<ButtonRelease>", self.sobel\_button\_released)  
 self.prewitt\_button.bind("<ButtonRelease>", self.prewitt\_button\_released)  
 self.threshold\_button.bind("<ButtonRelease>", self.threshold\_button\_released)  
 self.cluster\_button.bind("<ButtonRelease>", self.cluster\_button\_released)  
 self.apply\_button.bind("<ButtonRelease>", self.apply\_button\_released)  
 self.cancel\_button.bind("<ButtonRelease>", self.cancel\_button\_released)  
  
 self.canny\_button.pack()  
 self.laplacian\_button.pack()  
 self.sobel\_button.pack()  
 self.prewitt\_button.pack()  
 self.threshold\_button.pack()  
 self.cluster\_button.pack()  
 self.cancel\_button.pack(side=RIGHT)  
 self.apply\_button.pack()  
  
 def canny\_button\_released(self, event):  
 self.canny()  
 self.show\_image()  
  
 def laplacian\_button\_released(self, event):  
 self.laplacian()  
 self.show\_image()  
  
 def sobel\_button\_released(self, event):  
 self.sobel()  
 self.show\_image()  
  
 def prewitt\_button\_released(self, event):  
 self.prewitt()  
 self.show\_image()  
  
 def threshold\_button\_released(self, event):  
 self.threshold()  
 self.show\_image()  
  
 def cluster\_button\_released(self, event):  
 self.cluster()  
 self.show\_image()  
  
 def apply\_button\_released(self, event):  
 self.master.processed\_image = self.filtered\_image  
 self.show\_image()  
 self.close()  
  
 def cancel\_button\_released(self, event):  
 self.master.image\_viewer.show\_image()  
 self.close()  
  
 def show\_image(self):  
 self.master.image\_viewer.show\_image(img=self.filtered\_image)  
  
 def canny(self):  
 self.gray\_img = cv2.cvtColor(self.original\_image, cv2.COLOR\_BGRA2GRAY)  
 self.filtered\_image = cv2.Canny(self.gray\_img, 30, 200)  
  
 def laplacian(self):  
 self.gray\_img = cv2.cvtColor(self.original\_image, cv2.COLOR\_BGRA2GRAY)  
 lap = cv2.Laplacian(self.gray\_img, cv2.CV\_64F)  
 self.filtered\_image = np.uint8(np.absolute(lap))  
  
 def sobel(self):  
 self.gray\_img = cv2.cvtColor(self.original\_image, cv2.COLOR\_BGRA2GRAY)  
 sobelX = cv2.Sobel(self.gray\_img, cv2.CV\_64F, 1, 0)  
 sobelY = cv2.Sobel(self.gray\_img, cv2.CV\_64F, 0, 1)  
  
 sobelX = np.uint8(np.absolute(sobelX))  
 sobelY = np.uint8(np.absolute(sobelY))  
  
 self.filtered\_image = cv2.bitwise\_or(sobelX, sobelY)  
  
 def prewitt(self):  
 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(self.original\_image, -1, kernelx)  
 img\_prewitty = cv2.filter2D(self.original\_image, -1, kernely)  
 self.filtered\_image = cv2.bitwise\_or(img\_prewittx, img\_prewitty)  
  
 def threshold(self):  
 self.gray\_img = cv2.cvtColor(self.original\_image, cv2.COLOR\_BGRA2GRAY)  
 retval, threshold = cv2.threshold(self.gray\_img, 62, 255, cv2.THRESH\_BINARY)  
 self.filtered\_image = threshold  
  
 def cluster(self):  
 img = self.original\_image  
 Z = img.reshape((-1, 3))  
  
 # convert to np.float32  
 Z = np.float32(Z)  
  
 # define criteria, number of clusters(K) and apply kmeans()  
 criteria = (cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER, 10, 1.0)  
 K = 8  
 ret, label, center = cv2.kmeans(Z, K, None, criteria, 10, cv2.KMEANS\_RANDOM\_CENTERS)  
  
 # Now convert back into uint8, and make original image  
 center = np.uint8(center)  
 res = center[label.flatten()]  
 res2 = res.reshape((img.shape))  
  
 self.filtered\_image = res2  
  
 def close(self):  
 self.destroy()  
  
  
class ImageViewer(Frame):  
 def \_\_init\_\_(self, master=None):  
 Frame.\_\_init\_\_(self, master=master, bg='#6ac7e6', width=600, height=400)  
  
 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='#6ac7e6', width=600, height=400)  
 # self.canvas.place(relx=0.5, rely=0.5, anchor=CENTER)  
 self.canvas = Canvas(self, bg='#6ac7e6', width=600, height=400)  
 self.canvas.place(relx=0.5, rely=0.5, anchor=CENTER)  
 self.canvas2 = Canvas(self, bg='#6ac7e6', width=150, height=150)  
 self.canvas2.place(relx=0.87, rely=0.83, anchor=CENTER)  
 self.canvas3 = Canvas(self, bg='#6ac7e6', width=150, height=150)  
 self.canvas3.place(relx=0.13, rely=0.83, 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 > self.winfo\_height() or width > self.winfo\_width():  
 if ratio < 1:  
 new\_width = self.winfo\_width()  
 new\_height = int(new\_width \* ratio)  
 else:  
 new\_height = self.winfo\_height()  
 new\_width = int(new\_height \* (width / height))  
  
 if height2 > 150 or width2 > 150:  
 if ratio2 < 1:  
 new\_width2 = 150  
 new\_height2 = int(new\_width2 \* ratio2)  
 else:  
 new\_height2 = 150  
 new\_width2 = int(new\_height2 \* (width2 / height2))  
  
 if height3 > 150 or width3 > 150:  
 if ratio3 < 1:  
 new\_width3 = 150  
 new\_height3 = int(new\_width3 \* ratio3)  
 else:  
 new\_height3 = 150  
 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\_draw(self):  
 self.canvas.bind("<ButtonPress>", self.start\_draw)  
 self.canvas.bind("<B1-Motion>", self.draw)  
  
 self.master.is\_draw\_state = True  
  
 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\_draw(self):  
 self.canvas.unbind("<ButtonPress>")  
 self.canvas.unbind("<B1-Motion>")  
  
 self.master.is\_draw\_state = False  
  
 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\_draw(self, event):  
 self.x = event.x  
 self.y = event.y  
  
 def draw(self, event):  
 self.draw\_ids.append(self.canvas.create\_line(self.x, self.y, event.x, event.y, width=2,  
 fill="black", capstyle=ROUND, smooth=True))  
  
 cv2.line(self.master.processed\_image, (int(self.x \* self.ratio), int(self.y \* self.ratio)),  
 (int(event.x \* self.ratio), int(event.y \* self.ratio)),  
 (0, 0, 255), thickness=int(self.ratio \* 2),  
 lineType=8)  
  
 self.x = event.x  
 self.y = event.y  
  
 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\_hist(self):  
 plt.figure(num='Image Histogram')  
 hist = cv2.calcHist(self.master.processed\_image, [0], None, [256], [0, 256])  
 plt.plot(hist)  
 plt.hist(self.master.processed\_image.flatten(), 256, [0, 256], )  
 # plt.title('Image Histogram')  
 plt.show()  
  
 def deactivate\_hist(self):  
 pass  
  
 def clear\_canvas(self):  
 self.canvas.delete("all")  
  
 def clear\_draw(self):  
 self.canvas.delete(self.draw\_ids)  
  
  
root = Main()  
root.mainloop()