MODERN ACADEMY

FOR ENGINEERING & TECHNOLOGY

Computer Department

Academic Year 2023/2024

December 2023

Toolbox Documentation

Course Title: Digital image processing

Course Code: CMPN332

Name	**Youssef Ahmed Saied**
Section	**2**
ID	**4200479**
Dr	**Sabry Mohamed Abdelmoaty**

Content:

First Installing the program:

1. These is the link of python to install first

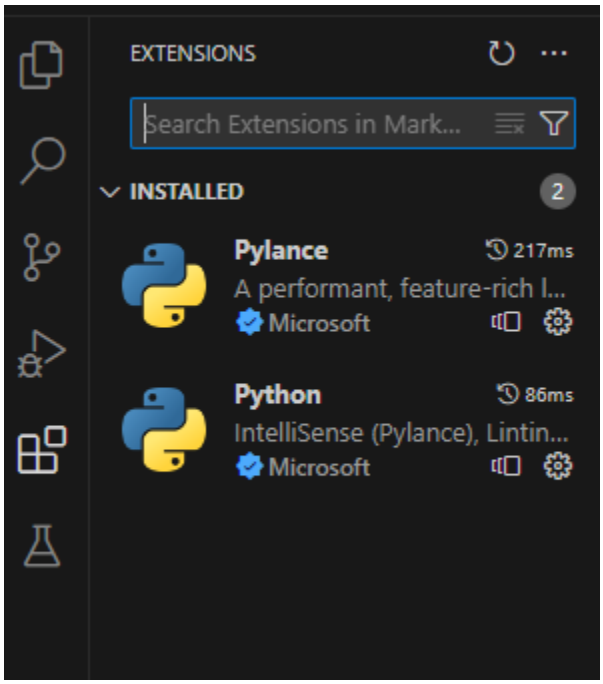
   https://www.python.org/ftp/python/3.12.1/python-3.12.1-amd64.exe

2. Then setup program by clicking next to begin setup process.

3. Download visual studio code
   https://code.visualstudio.com/download

   Then install it

4. Open visual studio code and download python extensions



5. Download The needed libirares

Second initializing the program:

```
#-----Includes------
import customtkinter as ctk
import tkinter as tk
from tkinter import filedialog ,simpledialog
import cv2
from tkinter import Canvas
from PIL import Image , ImageTk, ImageDraw
import numpy as np
import os
import subprocess
import os
from matplotlib import pyplot as plt
```

Third Source Code:

```
class Main(ctk.CTk):
```

```
    def __init__(self):
        super().__init__()


        #------Setup------
        self.title('ToolBox')
        self.geometry('1366x768')


        #------variable------
        self.image = None
        self.image2 = None
        self.original = None
        self.rotate_angle = ctk.IntVar(value=0)
        self.rotate_angle.trace('w' , self.rotate_image)
        self.scale_factor_x = 1.0
        self.scale_factor_y = 1.0
        self.scale_factors = ctk.StringVar(value="0.0:0.0")
        self.skew_factor_x = 1.0
        self.skew_factor_y = 1.0
        self.trans_factor_x = 0
        self.trans_factor_y = 0
        self.blur_angle = ctk.IntVar(value=0)
        self.blur_angle.trace('w' , self.blur_image)
        self.threshold = 0
        self.blend_factor_x = 0
        self.blend_factor_y = 0
        self.crop_start_factor_x  = 0
        self.crop_start_factor_y  = 0
        self.crop_width_factor_x  = 0
        self.crop_height_factor_y = 0

        #------Frames--------
        #Left Frame
        L_frame = ctk.CTkFrame(master = self , corner_radius = 0 , fg_color =
'#212121')
        L_frame.place(relx = 0 , rely = 0 , relwidth = 0.2 , relheight = 1)
        #Right Frame
        frame1 = ctk.CTkFrame(master = self , corner_radius = 0 , fg_color =
'#212121')
        frame1.place(relx = 0.2 , rely = 0 , relwidth = 0.2 , relheight = 1)
            #Frame1
        R_frame = ctk.CTkFrame(master = frame1 , fg_color = '#212121')
        R_frame.pack(fill = 'x' , expand = True, padx = 8 , pady = 8 )
            #Frame2
```

```
        frame2 = ctk.CTkFrame(master = L_frame , fg_color = '#212121')
        frame2.pack(fill = 'x' , padx = 8 , pady = 8 , expand = True)
            #Frame3
        frame3 = ctk.CTkFrame(master = L_frame , corner_radius = 0 , fg_color =
'transparent' )
        frame3.pack(fill = 'x' , expand = True)
            #Frame4
        frame4 = ctk.CTkFrame(master = self , corner_radius = 0 , fg_color =
'#212121' )
        frame4.place(relx = 0.4 , rely = 0.8 , relwidth = 0.2 , relheight = 0.2)


        #MID Frame
            #------Canvas------
        self.canvas = Canvas(master=self)
        self.canvas.place(relx = 0.4 , rely = 0 , relwidth = 0.6 , relheight =
0.8)




        #Bottum Frame
        B_frame = ctk.CTkFrame(master = self , corner_radius = 0 , fg_color =
'#212121')
        B_frame.place(relx = 0.6 , rely = 0.8 , relwidth = 0.4 , relheight = 0.2)



        #--------Main Buttons-------
        btn_frame = ctk.CTkFrame(master = B_frame , fg_color = '#111111')
        btn_frame.pack(fill = 'x' , expand = True )
            # Configure of row
        btn_frame.rowconfigure((0,1,2) , weight=1 , uniform='a')
            # Configure of column
        btn_frame.columnconfigure((0) , weight=1 , uniform='a')
            #Browse Button
        browse_button = ctk.CTkButton(master = btn_frame , text= 'Browse' ,
command=self.load_image, fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        browse_button.grid(row = 0 , column = 0  , padx = 8 , pady = 8, sticky =
'ew')
            #Reset Button
```

```
        reset_button = ctk.CTkButton(master = btn_frame , text= 'Reset' ,
command= self.reset_image, fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        reset_button.grid(row = 1 , column = 0 , padx = 8 , pady = 8, sticky =
'ew')
            #Export Button
        export_button = ctk.CTkButton(master = btn_frame , text= 'Export' ,
command= self.save_image, fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        export_button.grid(row = 2 , column = 0  , padx = 8 , pady = 8, sticky =
'ew')




        #---------Rotate Frame--------
        rotate_frame = ctk.CTkFrame(master = frame2 , fg_color = '#111111')
        rotate_frame.pack(fill = 'x' , expand = True , padx = 8 , pady = 8)
            # Configure of row
        rotate_frame.rowconfigure((0,1) , weight=1 , uniform='a')
            # Configure of column
        rotate_frame.columnconfigure((0,1) , weight=1 , uniform='a')
            # Rotate label
        ratate_label = ctk.CTkLabel(master=rotate_frame , text = 'Rotate
Image',text_color='white')
        ratate_label.grid(row = 0 , column = 0 , padx = 8 , pady = 8 , sticky =
'w')
            # Rotate label angle
        ratate_angle_label = ctk.CTkLabel(master=rotate_frame , textvariable =
self.rotate_angle)
        ratate_angle_label.grid(row = 0 , column = 1 , padx = 8 , pady = 8 ,
sticky = 'e')
            # Slider
        rotate_slider = ctk.CTkSlider(master=rotate_frame , from_=0 , to= 360 ,
variable=self.rotate_angle, fg_color =
'black',button_hover_color='#65e650',progress_color='#44D62C',button_color='#44D6
2C')
        rotate_slider.grid(row = 1 , column = 0 , columnspan = 2 , padx = 8 ,
pady  = 8 , sticky = 'ew')



        #Skew Image
        skew_frame = ctk.CTkFrame(master = frame2 , fg_color = '#111111')
        skew_frame.pack(fill = 'x' , expand = True , padx = 8 , pady = 8)
```

```
            # Configure of row
        skew_frame.rowconfigure((0,1) , weight=1 , uniform='a')
            # Configure of column
        skew_frame.columnconfigure((0,1) , weight=1 , uniform='a')
            # Skew label
        skew_label = ctk.CTkLabel(master=skew_frame , text = 'Skew
Image',text_color='white')
        skew_label.grid(row = 0 , column = 0 , padx = 8 , pady = 8 , sticky =
'w')
            # Skew label angle
        #---Buttons
        skew_button1 = ctk.CTkButton(master = skew_frame , text= 'Skew X' ,
command= lambda:self.skew_image('x'), fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        skew_button1.grid(row = 1 , column = 0 , padx = 8 , pady = 8)
        skew_button2 = ctk.CTkButton(master = skew_frame , text= 'Skew Y' ,
command= lambda:self.skew_image('y'), fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        skew_button2.grid(row = 1 , column = 1 , padx = 8 , pady = 8)
        #-----------------------------------------------------


        # Translation
        tran_frame = ctk.CTkFrame(master = frame2 , fg_color = '#111111')
        tran_frame.pack(fill = 'x' , expand = True , padx = 8 , pady = 8)
            # Configure of row
        tran_frame.rowconfigure((0,1) , weight=1 , uniform='a')
            # Configure of column
        tran_frame.columnconfigure((0,1) , weight=1 , uniform='a')
            # Translation label
        trans_label = ctk.CTkLabel(master=tran_frame , text =
'Translation',text_color='white')
        trans_label.grid(row = 0 , column = 0 , padx = 8 , pady = 8 , sticky =
'w')
            # Translation label angle
        #---Buttons Translation
        trans_button1 = ctk.CTkButton(master = tran_frame , text= 'Translate X' ,
command= lambda:self.trans_image('x'), fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        trans_button1.grid(row = 1 , column = 0 , padx = 8 , pady = 8)
        trans_button2 = ctk.CTkButton(master = tran_frame , text= 'Translate Y' ,
command= lambda:self.trans_image('y'), fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        trans_button2.grid(row = 1 , column = 1 , padx = 8 , pady = 8)
        #-----------------------------------------------------
```

```
        #---------Scale Frame--------
        scale_frame = ctk.CTkFrame(master = frame2 , fg_color = '#111111')
        scale_frame.pack(fill = 'x' , expand = True , padx = 8 , pady = 8)
            # Configure of row
        scale_frame.rowconfigure((0,1) , weight=1 , uniform='a')
            # Configure of column
        scale_frame.columnconfigure((0,1) , weight=1 , uniform='a')
            # Scale label
        scale_label = ctk.CTkLabel(master=scale_frame , text =
'Scale',text_color='white')
        scale_label.grid(row = 0 , column = 0 , padx = 8 , pady = 8 , sticky =
'w')
            # Scale Factors
        scale_fac_label = ctk.CTkLabel(master=scale_frame , textvariable =
self.scale_factors)
        scale_fac_label.grid(row = 0 , column = 1 , padx = 8 , pady = 8 , sticky
= 'e')
            #Scale X Button
        scale_x_button = ctk.CTkButton(master = scale_frame , text= 'Scale X' ,
command=lambda:self.scale_image("x"), fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        scale_x_button.grid(row = 1 , column = 0 , padx = 8 , pady = 8 , sticky =
'e')
            #Scale Y Button
        scale_y_button = ctk.CTkButton(master = scale_frame , text= 'Scale Y' ,
command=lambda:self.scale_image("y"), fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        scale_y_button.grid(row = 1 , column = 1 , padx = 8 , pady = 8 , sticky =
'w')


        #---------Flip Frame--------
        flip_frame = ctk.CTkFrame(master = frame2 , fg_color = '#111111')
        flip_frame.pack(fill = 'x' , expand = True , padx = 8 , pady = 8)
            # Configure of row
        flip_frame.rowconfigure((0,1) , weight=1 , uniform='a')
            # Configure of column
        flip_frame.columnconfigure((0,1,2) , weight=1 , uniform='a')
                    # Flip label
        scale_label = ctk.CTkLabel(master=flip_frame , text =
'Flip',text_color='white')
        scale_label.grid(row = 0 , column = 0 , padx = 8 , pady = 8 , sticky =
'w')
```

```
            #Flip X Button
        scale_x_button = ctk.CTkButton(master = flip_frame , text= 'X' ,
command=lambda:self.flip_image('x'), fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        scale_x_button.grid(row = 1 , column = 0 , padx = 8 , pady = 8 )
            #Flip Y Button
        scale_y_button = ctk.CTkButton(master = flip_frame , text= 'Y' ,
command=lambda:self.flip_image('y'), fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        scale_y_button.grid(row = 1 , column = 1 , padx = 8 , pady = 8 )
            #Flip Y Button
        scale_y_button = ctk.CTkButton(master = flip_frame , text= 'Both' ,
command=lambda:self.flip_image('b'), fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        scale_y_button.grid(row = 1 , column = 2 , padx = 8 , pady = 8 )



        #---------Filter Frame--------
        filter_frame = ctk.CTkFrame(master = frame2 , fg_color = '#111111')
        filter_frame.pack(fill = 'x' , expand = True , padx = 8 , pady = 8)
            # Configure of row
        filter_frame.rowconfigure((0,1) , weight=1 , uniform='a')
            # Configure of column
        filter_frame.columnconfigure((0) , weight=1 , uniform='a')
            # Filter label
        filter_label = ctk.CTkLabel(master=filter_frame , text =
'Filters',text_color='white')
        filter_label.grid(row = 0 , column = 0 , padx = 8 , pady = 8 , sticky =
'w')
            #Filter Drop Down Button
        filter_drop_down_button = ctk.CTkOptionMenu(filter_frame , values =
["Select", "Grayscale" , "Invert" , "Median Filter" ,"Averaging" ,"Laplacian" ,
"Edge Detection"] , command=self.filter_image, fg_color =
'#44D62C',button_color='#44D62C',button_hover_color='#65e650',dropdown_hover_colo
r='gray',text_color='black')
        filter_drop_down_button.grid(row = 1 , column = 0 , padx = 8 , pady = 8 ,
sticky = 'ew' )

        # ------Zoom----
        zoom_frame = ctk.CTkFrame(master = frame2 , fg_color = '#111111')
        zoom_frame.pack(fill = 'x' , expand = True , padx = 8 , pady = 8)
            # Configure of row
        zoom_frame.rowconfigure((0,1) , weight=1 , uniform='a')
            # Configure of column
```

```
        zoom_frame.columnconfigure((0) , weight=1 , uniform='a')
            # zoom label
        zoom_label = ctk.CTkLabel(master=zoom_frame , text =
'Croping',text_color='white')
        zoom_label.grid(row = 0 , column = 0 , padx = 8 , pady = 8 , sticky =
'w')
            # Zoom Button
        zoom_button = ctk.CTkButton(master = zoom_frame , text= 'Crop' ,
command=self.zoom_image, fg_color =
'#44D62C',hover_color='#65e650',text_color='Black',font=('Bold',15))
        zoom_button.grid(row = 1, column = 0 , padx = 8 , pady = 8 , sticky =
'ew')



        #---------Bit Plan Frame--------
        bit_frame = ctk.CTkFrame(master = R_frame , fg_color = '#111111')
        bit_frame.pack(fill = 'x' , expand = True , padx = 8 , pady = 8)
            # Configure of row
        bit_frame.rowconfigure((0,1) , weight=1 , uniform='a')
            # Configure of column
        bit_frame.columnconfigure((0) , weight=1 , uniform='a')
            # Bit Plan label
        bit_label = ctk.CTkLabel(master=bit_frame , text = 'Bit
plan',text_color='white')
        bit_label.grid(row = 0 , column = 0 , padx = 8 , pady = 8 , sticky = 'w')
            #Bit Plan Drop Down Button
        bit_drop_down_button = ctk.CTkOptionMenu(bit_frame , values = ["Select"
,"128" , "64" , "32" , "16" , "8", "4" , "2" , "1"] , command=self.bit_image,
fg_color =
'#44D62C',button_color='#44D62C',button_hover_color='#65e650',dropdown_hover_colo
r='gray',text_color='black')
        bit_drop_down_button.grid(row = 1 , column = 0 , padx = 8 , pady = 8 ,
sticky = 'ew' )



        #---------Power Frame--------
        pow_frame = ctk.CTkFrame(master = R_frame , fg_color = '#111111')
        pow_frame.pack(fill = 'x' , expand = True , padx = 8 , pady = 8)
            # Configure of row
        pow_frame.rowconfigure((0,1) , weight=1 , uniform='a')
            # Configure of column
        pow_frame.columnconfigure((0) , weight=1 , uniform='a')
            # Power label
```

```
        bit_label = ctk.CTkLabel(master=pow_frame , text = 'Power
Transformation',text_color='white')
        bit_label.grid(row = 0 , column = 0 , padx = 8 , pady = 8 , sticky = 'w')
            #Power Drop Down Button
        pow_drop_down_button = ctk.CTkOptionMenu(pow_frame , values = ["Select" ,
"0.1" , "0.4" , "0.8" , "1.2" , "1.6", "2" ] , command=self.power_image, fg_color
=
'#44D62C',button_color='#44D62C',button_hover_color='#65e650',dropdown_hover_colo
r='gray',text_color='black')
        pow_drop_down_button.grid(row = 1 , column = 0 , padx = 8 , pady = 8 ,
sticky = 'ew' )


        #---------Sharping Frame--------
        sharp_frame = ctk.CTkFrame(master = R_frame , fg_color = '#111111')
        sharp_frame.pack(fill = 'x' , expand = True , padx = 8 , pady = 8)
            # Configure of row
        sharp_frame.rowconfigure((0,1) , weight=1 , uniform='a')
            # Configure of column
        sharp_frame.columnconfigure((0,1,2) , weight=1 , uniform='a')
                    # Flip label
        sharp_label = ctk.CTkLabel(master=sharp_frame , text =
'Sharping',text_color='white')
        sharp_label.grid(row = 0 , column = 0 , padx = 8 , pady = 8 , sticky =
'w')
            #sharp X Button
        scale_x_button = ctk.CTkButton(master = sharp_frame , text= 'Middle' ,
command=lambda:self.sharp_image('l'), fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        scale_x_button.grid(row = 1 , column = 1 , padx = 8 , pady = 8 )
            #sharp Y Button
        scale_y_button = ctk.CTkButton(master = sharp_frame , text= 'High' ,
command=lambda:self.sharp_image('m'), fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        scale_y_button.grid(row = 1 , column = 2 , padx = 8 , pady = 8 )
            #sharp Y Button
        scale_y_button = ctk.CTkButton(master = sharp_frame , text= 'Low' ,
command=lambda:self.sharp_image('h'), fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        scale_y_button.grid(row = 1 , column = 0 , padx = 8 , pady = 8 )


       #---------Blur Frame--------
        blur_frame = ctk.CTkFrame(master = R_frame , fg_color = '#111111')
        blur_frame.pack(fill = 'x' , expand = True , padx = 8 , pady = 8)
```

```
            # Configure of row
        blur_frame.rowconfigure((0,1) , weight=1 , uniform='a')
            # Configure of column
        blur_frame.columnconfigure((0,1) , weight=1 , uniform='a')
            # Blur label
        blur_label = ctk.CTkLabel(master=blur_frame , text = 'Gaussian
Blur',text_color='white')
        blur_label.grid(row = 0 , column = 0 , padx = 8 , pady = 8 , sticky =
'w')
            # Blur label angle
        blur_angle_label = ctk.CTkLabel(master=blur_frame , textvariable =
self.blur_angle)
        blur_angle_label.grid(row = 0 , column = 1 , padx = 8 , pady = 8 , sticky
= 'e')
            # Slider
        blur_slider = ctk.CTkSlider(master=blur_frame , from_=0 , to= 50 ,
variable=self.blur_angle,fg_color =
'black',button_hover_color='#65e650',progress_color='#44D62C',button_color='#44D6
2C')
        blur_slider.grid(row = 1 , column = 0 , columnspan = 2 , padx = 8 ,
pady  = 8 , sticky = 'ew')




        #---------threshold Frame--------
        threshold_frame = ctk.CTkFrame(master = R_frame , fg_color = '#111111')
        threshold_frame.pack(fill = 'x' , expand = True , padx = 8 , pady = 8)
            # Configure of row
        threshold_frame.rowconfigure((0,1) , weight=1 , uniform='a')
            # Configure of column
        threshold_frame.columnconfigure((0) , weight=1 , uniform='a')
            # threshold label
        threshold_label = ctk.CTkLabel(master=threshold_frame , text =
'Threshold',text_color='white')
        threshold_label.grid(row = 0 , column = 0 , padx = 8 , pady = 8 , sticky
= 'w')
            # Button
        threshold_button = ctk.CTkButton(master = threshold_frame , text= 'Range'
, command=self.threshold_image, fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        threshold_button.grid(row = 1 , column = 0 , padx = 8 , pady = 8 , sticky
= 'ew' )
```

```
        #---------Sobel Frame--------
        sobel_frame = ctk.CTkFrame(master = R_frame , fg_color = '#111111')
        sobel_frame.pack(fill = 'x' , expand = True , padx = 8 , pady = 8)
            # Configure of row
        sobel_frame.rowconfigure((0,1) , weight=1 , uniform='a')
            # Configure of column
        sobel_frame.columnconfigure((0,1,2) , weight=1 , uniform='a')
                    # Flip label
        sobel_label = ctk.CTkLabel(master=sobel_frame , text = 'Sobel
Filter',text_color='white')
        sobel_label.grid(row = 0 , column = 0 , padx = 8 , pady = 8 , sticky =
'w')
            #Flip X Button
        sobel_x_button = ctk.CTkButton(master = sobel_frame , text= 'X' ,
command=lambda:self.sobel_image('x'), fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        sobel_x_button.grid(row = 1 , column = 0 , padx = 8 , pady = 8 )
            #Flip Y Button
        sobel_y_button = ctk.CTkButton(master = sobel_frame , text= 'Y' ,
command=lambda:self.sobel_image('y'), fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        sobel_y_button.grid(row = 1 , column = 1 , padx = 8 , pady = 8 )
            #Flip Y Button
        sobel_b_button = ctk.CTkButton(master = sobel_frame , text= 'Both' ,
command=lambda:self.sobel_image('b'), fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        sobel_b_button.grid(row = 1 , column = 2 , padx = 8 , pady = 8 )


        # ----blend----
        blend_frame = ctk.CTkFrame(master = R_frame , fg_color = '#111111')
        blend_frame.pack(fill = 'x' , expand = True , padx = 8 , pady = 8)
            # Configure of row
        blend_frame.rowconfigure((0,1) , weight=1 , uniform='a')
            # Configure of column
        blend_frame.columnconfigure((0) , weight=1 , uniform='a')
            # blend label
        blend_label = ctk.CTkLabel(master=blend_frame , text =
'Blending',text_color='white')
        blend_label.grid(row = 0 , column = 0 , padx = 8 , pady = 8 , sticky =
'w')
            # blend Button
```

```
        blend_button = ctk.CTkButton(master = blend_frame , text= 'blend' ,
command=self.blend_image, fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        blend_button.grid(row = 1, column = 0 , padx = 8 , pady = 8 , sticky =
'ew')


        # ----OCR----
        ocr_frame = ctk.CTkFrame(master = frame4 , fg_color = '#111111')
        ocr_frame.pack(fill = 'x' , expand = True , padx = 8 , pady = 8)
            # Configure of row
        ocr_frame.rowconfigure((0,1) , weight=1 , uniform='a')
            # Configure of column
        ocr_frame.columnconfigure((0) , weight=1 , uniform='a')
            # ocr label
        ocr_label = ctk.CTkLabel(master=ocr_frame , text =
'OCR',text_color='white')
        ocr_label.grid(row = 0 , column = 0 , padx = 8 , pady = 8 , sticky = 'w')
            # ocr Button
        ocr_button = ctk.CTkButton(master = ocr_frame , text= 'OCR'
,command=self.ocr , fg_color =
'#44D62C',hover_color='#65e650',text_color='Black')
        ocr_button.grid(row = 1, column = 0 , padx = 8 , pady = 8 , sticky =
'ew')




    #Load image function ( Browse )
    def load_image(self):
        file_path = filedialog.askopenfilename(filetypes=[("Select An Image" ,
"*.png;*.jpg;*.jpeg")])
        if file_path :
            self.original = cv2.imread(file_path)
            self.image = self.original
            self.display_image()

    def load_image2(self):
        file_path = filedialog.askopenfilename(filetypes=[("Select An Image" ,
"*.png;*.jpg;*.jpeg")])
        if file_path :
            self.image2 = cv2.imread(file_path)
```

```
    # Display Image function
    def display_image(self):
        if self.image is not None :
            rgb_image = cv2.cvtColor(self.image , cv2.COLOR_BGR2RGB)
            tk_image = ImageTk.PhotoImage(Image.fromarray(rgb_image))
            self.canvas.create_image(0 , 0 , anchor = 'nw' , image = tk_image)
            self.canvas.tk_image = tk_image


    #Rotate image
    def rotate_image(self , *arg):
        if self.image is not None :
            center = (self.image.shape[1] // 2,self.image.shape[0] // 2) # to get
center of image
            rotation_matrix = cv2.getRotationMatrix2D(center ,
self.rotate_angle.get(), 1.0)
            self.image = cv2.warpAffine(self.original , rotation_matrix ,
(self.image.shape[1] , self.image.shape[0] ))
            self.display_image()


    #Scale image
    def scale_image(self , scale):
        if self.image is not None :
            if scale == 'x':
                self.scale_factor_x = simpledialog.askfloat("Scale X" , "Enter
scale for X direction:" , initialvalue = self.scale_factor_x)
            elif scale == 'y':
                self.scale_factor_y = simpledialog.askfloat("Scale Y" , "Enter
scale for Y direction:" , initialvalue = self.scale_factor_y)
            self.scale_factors.set(f"{self.scale_factor_x} :
{self.scale_factor_y}")
            self.image = cv2.resize(self.original , (0,0) ,
fx=self.scale_factor_x , fy = self.scale_factor_y)
            self.display_image()


    #----------Skewing---------------
    def skew_image(self , skew):
        if self.image is not None :
            height, width = self.image.shape[:2]

            if skew =='x':
```

```
                self.skew_factor_x = simpledialog.askfloat("Skew X" , "Enter skew
for X direction:" , initialvalue = self.skew_factor_x)
                pts1 = np.float32([[0, 0], [width - 1, 0], [0, height - 1],
[width - 1, height - 1]])
                pts2 = np.float32([[-self.skew_factor_x, 0], [width - 1 -
self.skew_factor_x, 0], [self.skew_factor_x, height - 1], [width - 1, height -
1]])
                matrix = cv2.getPerspectiveTransform(pts1, pts2)
                self.image = cv2.warpPerspective(self.image, matrix, (width,
height))
                #----------------------

            elif skew =='y':
                self.skew_factor_y = simpledialog.askfloat("Skew Y" , "Enter skew
for Y direction:" , initialvalue = self.skew_factor_y)
                pts1 = np.float32([[0, 0], [width - 1, 0], [0, height - 1],
[width - 1, height - 1]])
                pts2 = np.float32([[0, -self.skew_factor_y], [width - 1,
self.skew_factor_y], [0, height - 1 -self.skew_factor_y], [width - 1 , height -1
+ self.skew_factor_y]])
                matrix = cv2.getPerspectiveTransform(pts1, pts2)
                self.image = cv2.warpPerspective(self.image, matrix, (width,
height))
            self.display_image()
    #-------------------------------------------------------


    #----------Skewing---------------
    def trans_image(self , trans):
        if self.image is not None :
            height, width = self.image.shape[:2]
            if trans =='x':
                self.trans_factor_x = simpledialog.askfloat("Translate X" ,
"Enter Translate value for X direction:" , initialvalue = self.trans_factor_x)
                translation_matrix = np.float32([[1, 0, self.trans_factor_x], [0,
1, 0]])
                self.image = cv2.warpAffine(self.image, translation_matrix,
(width, height))
                #----------------------
            elif trans =='y':
                self.trans_factor_y = simpledialog.askfloat("Translate Y" ,
"Enter Translate value for Y direction:" , initialvalue = self.trans_factor_y)
                translation_matrix = np.float32([[1, 0, 0], [0, 1,
self.trans_factor_x]])
```

```
                self.image = cv2.warpAffine(self.image, translation_matrix,
(width, height))
            self.display_image()
    #-------------------------------------------------------


    #flip Image
    def flip_image(self , flip):
        if self.image is not None :
            if flip == 'x':
                self.image = cv2.flip(self.image , 1 )
            elif flip == 'y':
                self.image = cv2.flip(self.image , 0 )
            elif flip == 'b':
                self.image = cv2.flip(self.image , -1 )
            self.display_image()


    # Split Colors
    def split_color(self , color):
        if self.image is not None :
            r , g , b = cv2.split(self.image)
            blank = np.zeros(self.image.shape[:2] , dtype = "uint8")
            if color == 'Red':
                self.image = cv2.merge([blank,blank,r])
            elif color == 'Green':
                self.image = cv2.merge([blank,blank,g])
            elif color == 'Blue':
                self.image = cv2.merge([blank,blank,b])
            self.display_image()


    # Filter Image
    def filter_image(self , event):
        if self.image is not None :
            if event == "None" :
                self.image = self.original
            elif event == "Grayscale" :
                self.image = cv2.cvtColor(self.image,cv2.COLOR_BGR2GRAY)
            elif event == "Invert" :
                self.image = 255 - self.image
            elif event == "Median Filter" :
                self.image = cv2.medianBlur(self.image , 5)
            elif event == "Averaging" :
                self.image = cv2.cvtColor(self.image,cv2.COLOR_BGR2GRAY)
```

```
                kernel_size = (3, 3)
                kernel = (1.0 / (kernel_size[0] * kernel_size[1])) *
np.ones(kernel_size, dtype=np.float32)
                self.image = cv2.filter2D(self.image, -1, kernel)
            elif event == "Laplacian" :
                self.image = cv2.cvtColor(self.image,cv2.COLOR_BGR2GRAY)
                laplacian_kernel = np.array([[0, -1, 0],[-1, 4, -1],[0, -1, 0]],
dtype=np.float32)
                self.image = cv2.filter2D(self.image, -1, laplacian_kernel)
            elif event == "Edge Detection":
                self.image = cv2.cvtColor(self.image,cv2.COLOR_BGR2GRAY)
                kernel_L = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]],
dtype=np.float32)
                self.image = cv2.filter2D(self.image, cv2.CV_8UC1, kernel_L)
            self.display_image()


    # Zooming
    def zoom_image(self):
        if self.image is not None :
            self.crop_start_factor_x = simpledialog.askinteger("X point" , "Enter
start point for X direction:" , initialvalue = self.crop_start_factor_x)
            self.crop_start_factor_y = simpledialog.askinteger("Y point" , "Enter
start point for Y direction:" , initialvalue = self.crop_start_factor_y)
            self.crop_width_factor_x = simpledialog.askinteger("Width" , "Enter
width of crop image:" , initialvalue = self.crop_width_factor_x)
            self.crop_height_factor_y = simpledialog.askinteger("Height" , "Enter
height of crop image:" , initialvalue = self.crop_height_factor_y)
            x, y, width, height = self.crop_start_factor_x,
self.crop_start_factor_y, self.crop_width_factor_x, self.crop_height_factor_y
            self.image = self.image[y:y+height, x:x+width]
            self.display_image()

    # Bit Plan
    def bit_image(self , event):
        if self.image is not None :
            lst = []
            for row in range(self.image.shape[0]):
                for col in range(self.image.shape[1]):
                    lst.append(np.binary_repr(self.image[row][col] ,width=8))
            if event == '128':
                self.image = (np.array([int(i[0]) for i in lst],dtype = np.uint8)
* 128).reshape(self.image.shape[0],self.image.shape[1])
            elif event == '64':
```

```
                self.image = (np.array([int(i[1]) for i in lst],dtype = np.uint8)
* 64).reshape(self.image.shape[0],self.image.shape[1])
            elif event == '32':
                self.image = (np.array([int(i[2]) for i in lst],dtype = np.uint8)
* 32).reshape(self.image.shape[0],self.image.shape[1])
            elif event == '16':
                self.image = (np.array([int(i[3]) for i in lst],dtype = np.uint8)
* 16).reshape(self.image.shape[0],self.image.shape[1])
            elif event == '8':
                self.image = (np.array([int(i[4]) for i in lst],dtype = np.uint8)
* 8).reshape(self.image.shape[0],self.image.shape[1])
            elif event == '4':
                self.image = (np.array([int(i[5]) for i in lst],dtype = np.uint8)
* 4).reshape(self.image.shape[0],self.image.shape[1])
            elif event == '2':
                self.image = (np.array([int(i[6]) for i in lst],dtype = np.uint8)
* 2).reshape(self.image.shape[0],self.image.shape[1])
            elif event == '1':
                self.image = (np.array([int(i[7]) for i in lst],dtype = np.uint8)
* 1).reshape(self.image.shape[0],self.image.shape[1])
            self.display_image()



    # Power Transformation
    def power_image(self , event):
        if self.image is not None :
            if event == '0.1':
                arr = np.array(255*(self.image/255)**0.1,dtype='uint8')
            elif event == '0.4':
                arr = np.array(255*(self.image/255)**0.4,dtype='uint8')
            elif event == '0.8':
                arr = np.array(255*(self.image/255)**0.8,dtype='uint8')
            elif event == '1.2':
                arr = np.array(255*(self.image/255)**1.2,dtype='uint8')
            elif event == '1.6':
                arr = np.array(255*(self.image/255)**1.6,dtype='uint8')
            elif event == '2':
                arr = np.array(255*(self.image/255)**2,dtype='uint8')
            cv2.normalize(arr,arr,0,255,cv2.NORM_MINMAX)
            cv2.convertScaleAbs(arr,arr)
            self.image = arr
            self.display_image()
```

```
    #Sharp Image
    def sharp_image(self , sharp):
        if self.image is not None :
            if sharp == 'l':
                kernal_shearing = np.array([[-1, -1, -1],[-1, 9, -1],[-1, -1, -
1]])
                self.image = cv2.filter2D(self.image, -1, kernal_shearing)
            elif sharp == 'm':
                kernal_shearing = np.array([[1, 1, 1],[1, -7, 1],[1, 1, 1]])
                self.image = cv2.filter2D(self.image, -1, kernal_shearing)
            elif sharp == 'h':
                kernal_shearing = np.array([[-1,-1,-1,-1,-1],
                            [-1,2,2,2,-1],
                            [-1,2,8,2,-1],
                            [-1,2,2,2,-1],
                            [-1,-1,-1,-1,-1]]) / 8.0
                self.image = cv2.filter2D(self.image, -1, kernal_shearing)
            self.display_image()


    # Blur
    def blur_image(self , *arg):
        if self.image is not None :
            var = self.blur_angle.get()
            self.image = cv2.GaussianBlur(self.image , (var,var) , 0)
            self.display_image()

   # Threshold
    def threshold_image(self):
        if self.image is not None :
            self.threshold = simpledialog.askinteger("Threshould" , "---" ,
initialvalue = self.threshold)
            self.image = cv2.cvtColor(self.image , cv2.COLOR_BGR2GRAY)
            ret , self.image = cv2.threshold(self.image ,
self.threshold,self.threshold, 0)
            self.display_image()


    # Sobel
    def sobel_image(self,sobel):
        if self.image is not None :
            self.image = cv2.cvtColor(self.image, cv2.COLOR_BGR2GRAY)
            if sobel == 'x':
                self.image = cv2.Sobel(self.image , cv2.CV_8U , 1 , 0 , 3 )
            elif sobel == 'y':
```

```
                self.image = cv2.Sobel(self.image , cv2.CV_8U , 0 , 1 , 3 )
            elif sobel == 'b':
                self.image = cv2.Sobel(self.image , cv2.CV_8U , 1 , 1 , 3 )
            self.display_image()



    #Reset Image
    def reset_image(self):
        if self.image is not None :
            self.image = self.original
            self.scale_factor_x = 1.0
            self.scale_factor_y = 1.0
            self.scale_factors.set(f"{self.scale_factor_x} :
{self.scale_factor_y}")
            self.display_image()


    #blend
    def blend_image(self):
        if self.image is not None :
            height, width = self.image.shape[:2]
            self.load_image2()
            self.blend_factor_x = simpledialog.askfloat("Ratio" , "Enter ratio of
first image:" , initialvalue = self.blend_factor_x)
            self.blend_factor_y = simpledialog.askfloat("Ratio" , "Enter ratio of
second image:" , initialvalue = self.blend_factor_y)
            self.image2 = cv2.resize(self.image2,(width,height))
            for row in range(height):
                for col in range(width):
                    self.image[row,col] = self.image[row,col] *
self.blend_factor_x + self.image2[row,col] * self.blend_factor_y
            self.display_image()


    def ocr(self):
        try:
            subprocess.run(["python", "Main.py"], check=True)
            print("Main.py executed successfully")
        except subprocess.CalledProcessError as e:
            print("Error executing Main.py:", e)


    #Save image function ( Export )
    def save_image(self):
        if self.image is not None :
```

```
            file_path = filedialog.asksaveasfilename(title="Save Image"
,defaultextension=".png" ,filetypes=[("PNG files" , "*.png")])
        if file_path:
            cv2.imwrite(file_path , self.image)









if __name__=='__main__':
    app = Main()
    app.mainloop()
```

OCR

```
#-----Includes------
import customtkinter as ctk
import tkinter as tk
from tkinter import filedialog ,simpledialog
import cv2
from tkinter import Canvas
from PIL import Image , ImageTk, ImageDraw
import numpy as np
import os


import DetectChars
import DetectPlates
import PossiblePlate


# module level variables
###########################################################################
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)
```

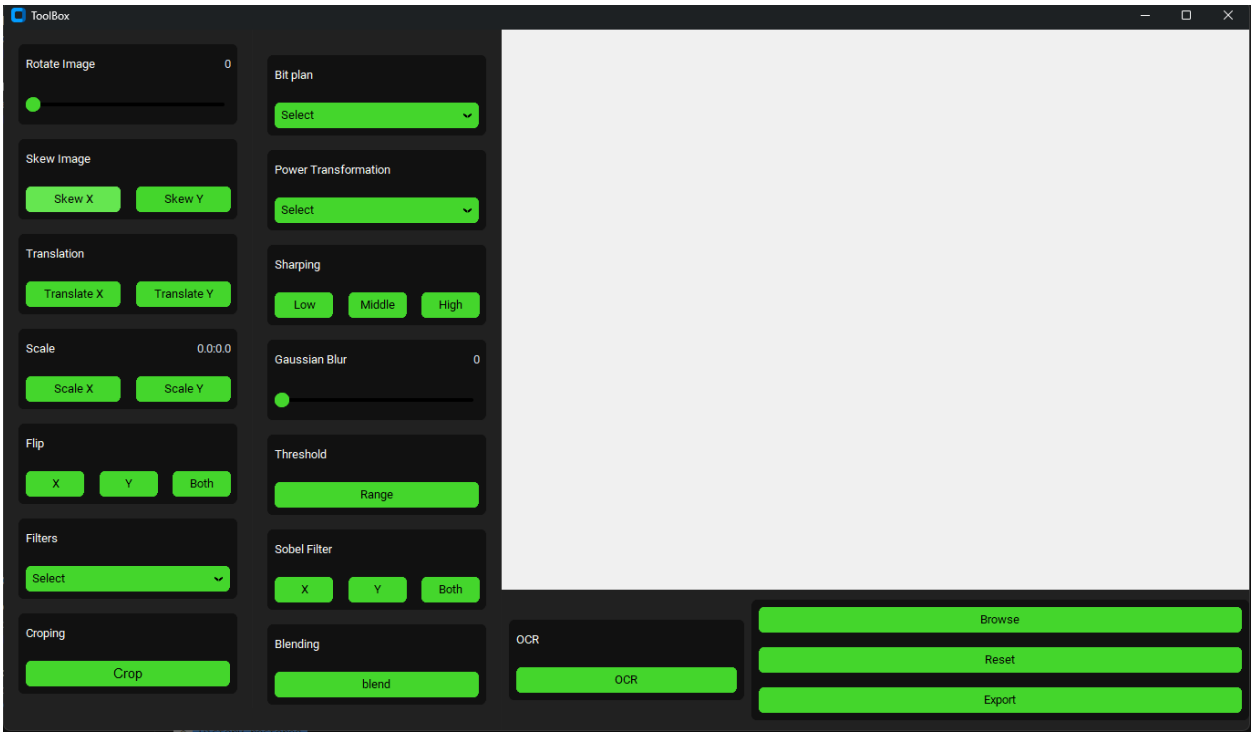
```
showSteps = False
classifications_path = "C:/path/to/classifications.txt"

# ... (rest of your code)


###################################################################################
###################
def main():



    blnKNNTrainingSuccessful = DetectChars.loadKNNDataAndTrainKNN()         #
attempt KNN training

    if blnKNNTrainingSuccessful == False:                               # if KNN
training was not successful
        print("\nerror: KNN traning was not successful\n")  # show error message
        return                                                          # and
exit program
    # end if
    imgOriginalScene  = cv2.imread(filedialog.askopenfilename(filetypes=[("Select
An Image" , "*.png;*.jpg;*.jpeg")]))          # open image

    if imgOriginalScene is None:                            # if image was not
read successfully
        print("\nerror: image not read from file \n\n")  # print error message to
std out
        os.system("pause")                                  # pause so user can
see error message
        return                                              # and exit program
    # end if

    listOfPossiblePlates =
DetectPlates.detectPlatesInScene(imgOriginalScene)           # detect plates

    listOfPossiblePlates =
DetectChars.detectCharsInPlates(listOfPossiblePlates)        # detect chars in
plates

    cv2.imshow("imgOriginalScene", imgOriginalScene)            # show scene
image

    if len(listOfPossiblePlates) == 0:                          # if no plates
were found
```

```
        print("\nno license plates were detected\n")  # inform user no plates
were found
    else:                                                   # else
                # if we get in here list of possible plates has at leat one plate

                # sort the list of possible plates in DESCENDING order (most
number of chars to least number of chars)
        listOfPossiblePlates.sort(key = lambda possiblePlate:
len(possiblePlate.strChars), reverse = True)

                # suppose the plate with the most recognized chars (the first
plate in sorted by string length descending order) is the actual plate
        licPlate = listOfPossiblePlates[0]

        cv2.imshow("imgPlate", licPlate.imgPlate)           # show crop of plate
and threshold of plate
        cv2.imshow("imgThresh", licPlate.imgThresh)

        if len(licPlate.strChars) == 0:                     # if no chars were
found in the plate
            print("\nno characters were detected\n\n")  # show message
            return                                          # and exit program
        # end if

        drawRedRectangleAroundPlate(imgOriginalScene, licPlate)             #
draw red rectangle around plate

        print("\nlicense plate read from image = " + licPlate.strChars + "\n")  #
write license plate text to std out
        print("----------------------------------------")

        writeLicensePlateCharsOnImage(imgOriginalScene, licPlate)           #
write license plate text on the image

        cv2.imshow("imgOriginalScene", imgOriginalScene)                # re-show
scene image

        cv2.imwrite("imgOriginalScene.png", imgOriginalScene)           # write
image out to file

    # end if else

    cv2.waitKey(0)                  # hold windows open until user presses a key

    return
```

```
# end main

###################################################################################################
def drawRedRectangleAroundPlate(imgOriginalScene, licPlate):

    p2fRectPoints = cv2.boxPoints(licPlate.rrLocationOfPlateInScene)            # get 4 vertices of rotated rect
    p2fRectPoints = np.int0(p2fRectPoints)
    cv2.line(imgOriginalScene, tuple(p2fRectPoints[0]), tuple(p2fRectPoints[1]), SCALAR_RED, 2)         # draw 4 red lines
    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)
# end function

###################################################################################################
def writeLicensePlateCharsOnImage(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                    # choose a plain jane font
    fltFontScale = float(plateHeight) / 30.0                  # base font scale on height of plate area
    intFontThickness = int(round(fltFontScale * 1.5))         # base font thickness on font scale

    textSize, baseline = cv2.getTextSize(licPlate.strChars, intFontFace, fltFontScale, intFontThickness)        # call getTextSize

            # unpack roatated rect into center point, width and height, and angle
```

```
    ( (intPlateCenterX, intPlateCenterY), (intPlateWidth, intPlateHeight),
fltCorrectionAngleInDeg ) = licPlate.rrLocationOfPlateInScene

    intPlateCenterX = int(intPlateCenterX)              # make sure center is an
integer
    intPlateCenterY = int(intPlateCenterY)

    ptCenterOfTextAreaX = int(intPlateCenterX)         # the horizontal location
of the text area is the same as the plate

    if intPlateCenterY < (sceneHeight *
0.75):                                                  # if the license plate is
in the upper 3/4 of the image
        ptCenterOfTextAreaY = int(round(intPlateCenterY)) + int(round(plateHeight
* 1.6))      # write the chars in below the plate
    else:
            # else if the license plate is in the lower 1/4 of the image
        ptCenterOfTextAreaY = int(round(intPlateCenterY)) - int(round(plateHeight
* 1.6))      # write the chars in above the plate
    # end if

    textSizeWidth, textSizeHeight = textSize                # unpack text size
width and height

    ptLowerLeftTextOriginX = int(ptCenterOfTextAreaX - (textSizeWidth /
2))           # calculate the lower left origin of the text area
    ptLowerLeftTextOriginY = int(ptCenterOfTextAreaY + (textSizeHeight /
2))          # based on the text area center, width, and height

            # write the text on the image
    cv2.putText(imgOriginalScene, licPlate.strChars, (ptLowerLeftTextOriginX,
ptLowerLeftTextOriginY), intFontFace, fltFontScale, SCALAR_YELLOW,
intFontThickness)
    return imgOriginalScene
# end function

#################################################################################
##################
if __name__ == "__main__":
    main()
```

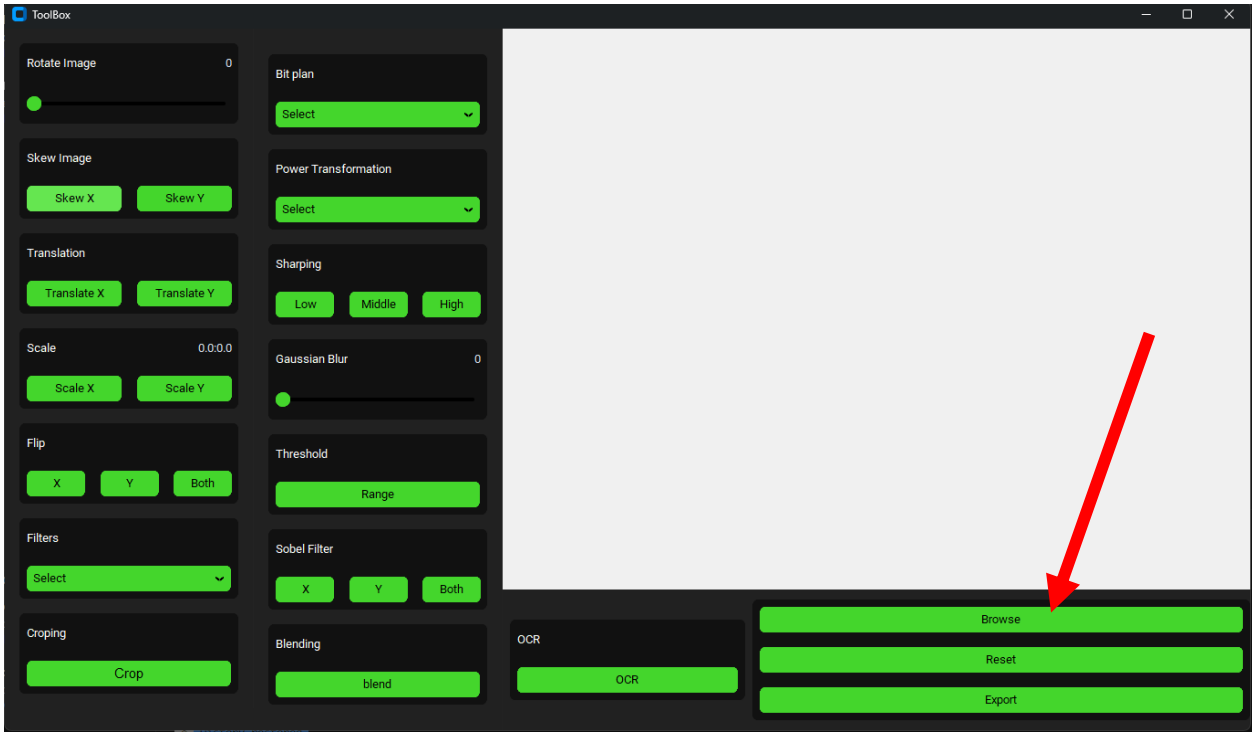
Fourth program final view:



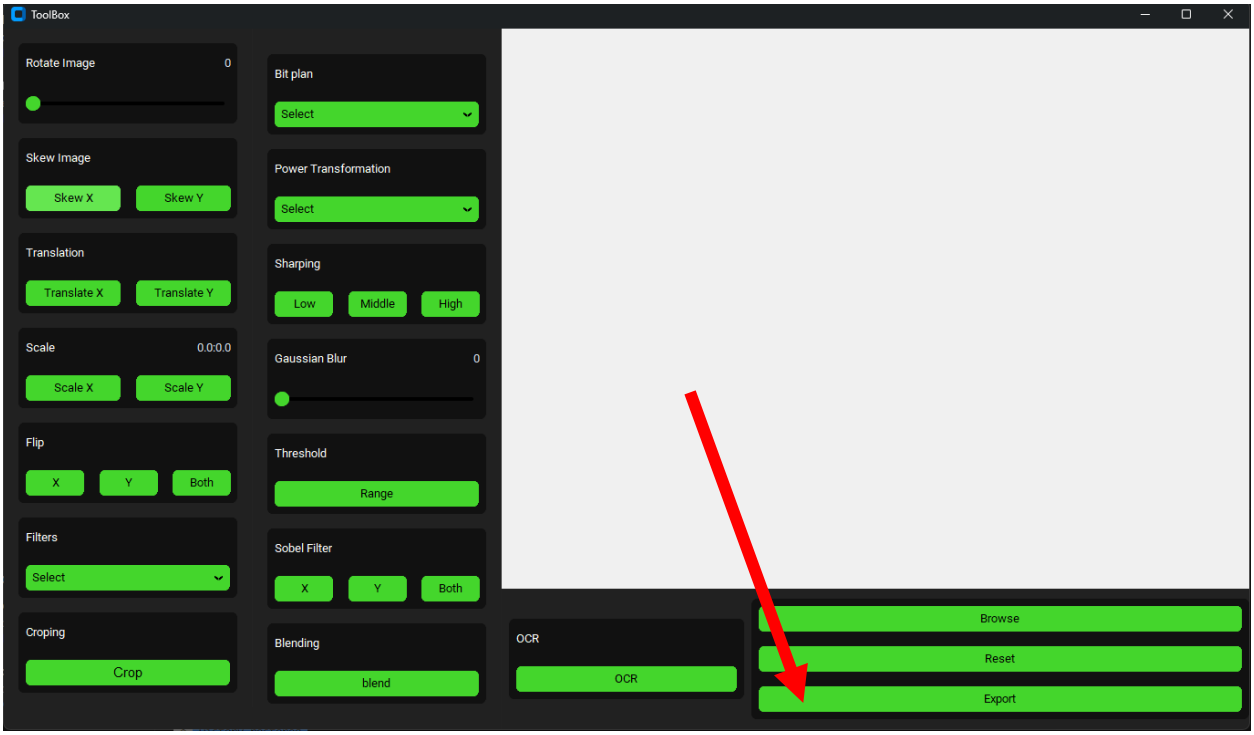
Fifth Every button and component in the program:

Main part

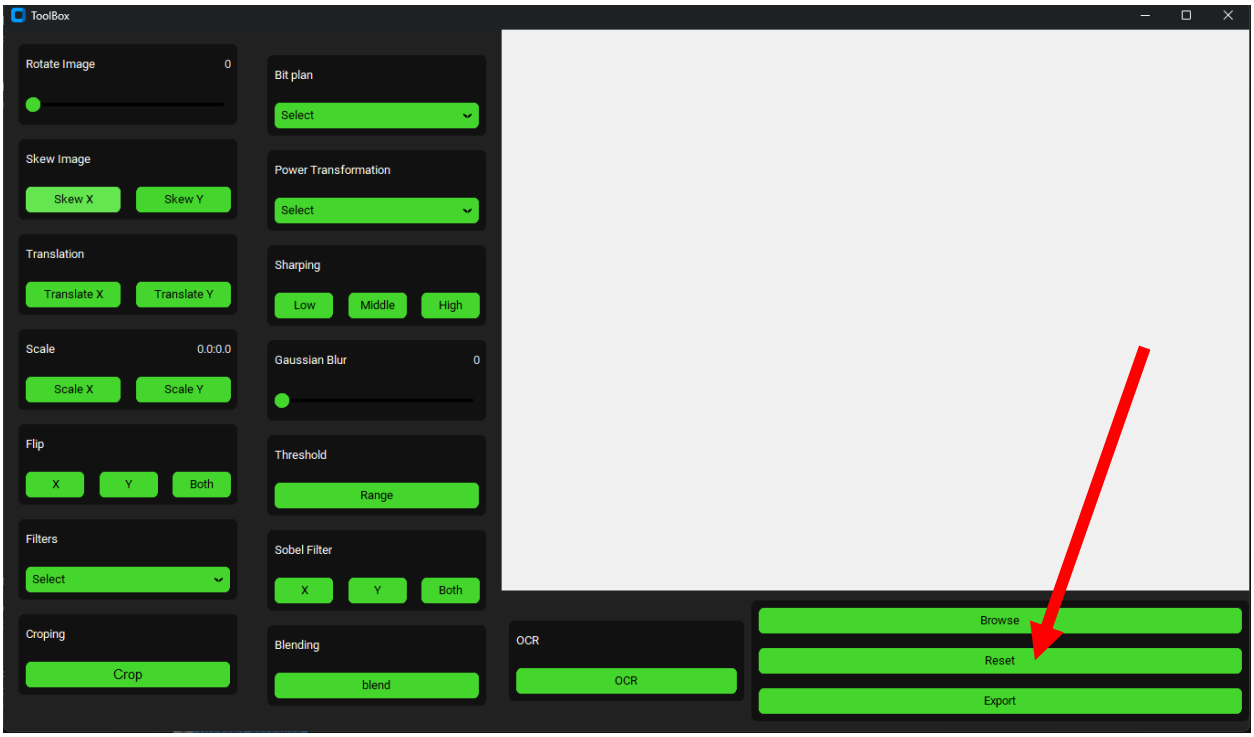
1. Browse → Used to upload image to toolbox


ToolBox
Rotate Image
0
Skew Image
Skew X
Skew Y
Translation
Translate X
Translate Y
Scale
0.0:0.0
Scale X
Scale Y
Flip
X
Y
Both
Filters
Select
Croping
Crop
Bit plan
Select
Power Transformation
Select
Sharping
Low
Middle
High
Gaussian Blur
0
Threshold
Range
Sobel Filter
X
Y
Both
Blending
blend
OCR
OCR
Browse
Reset
Export

2. Save image

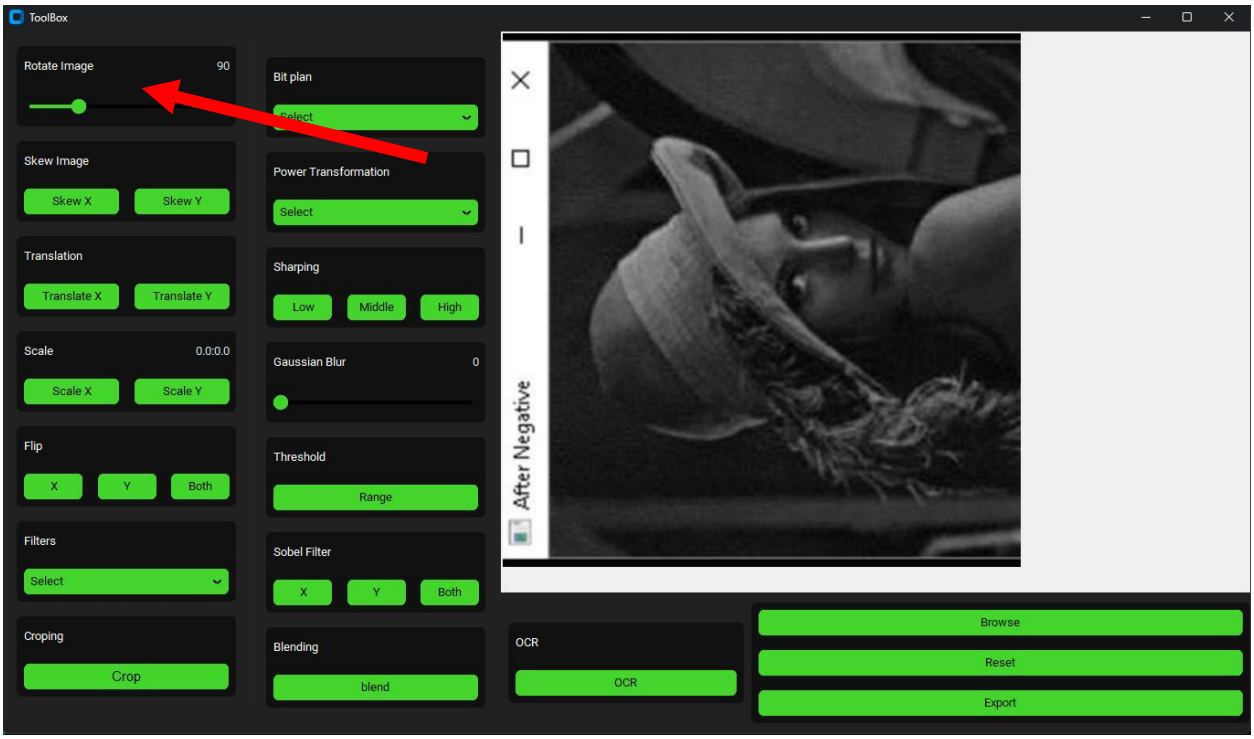

ToolBox
Rotate Image
0
Skew Image
Skew X
Skew Y
Translation
Translate X
Translate Y
Scale
0.0:0.0
Scale X
Scale Y
Flip
X
Y
Both
Filters
Select
Croping
Crop
Bit plan
Select
Power Transformation
Select
Sharping
Low
Middle
High
Gaussian Blur
0
Threshold
Range
Sobel Filter
X
Y
Both
Blending
blend
OCR
OCR
Browse
Reset
Export

3. Reset Image to the first upload view

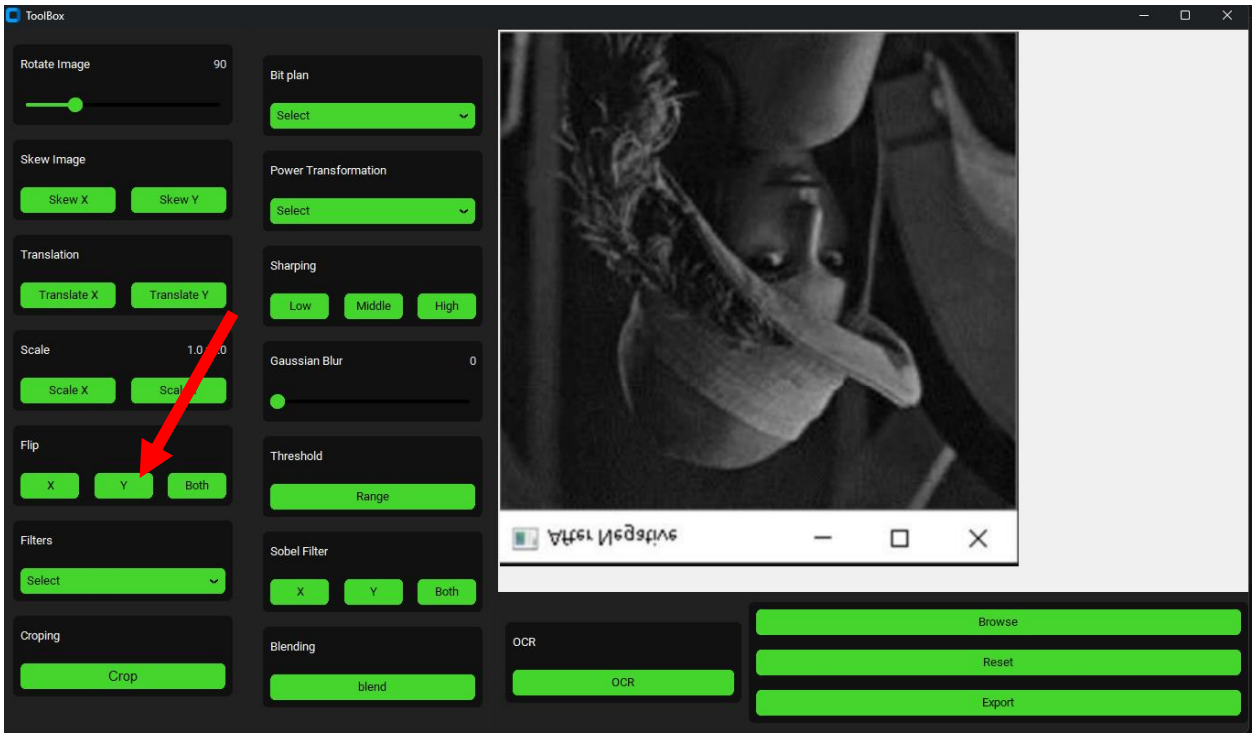


Edit part

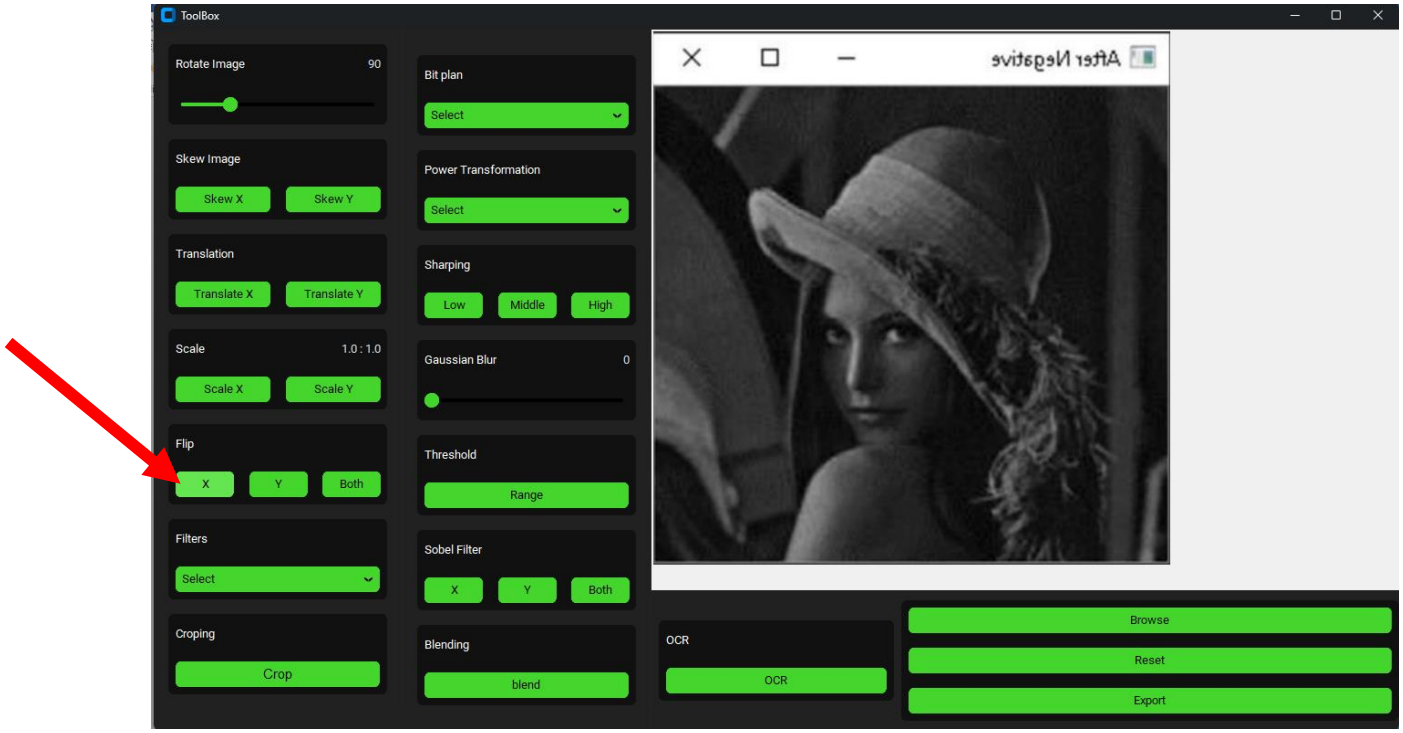
1. Rotate the Image



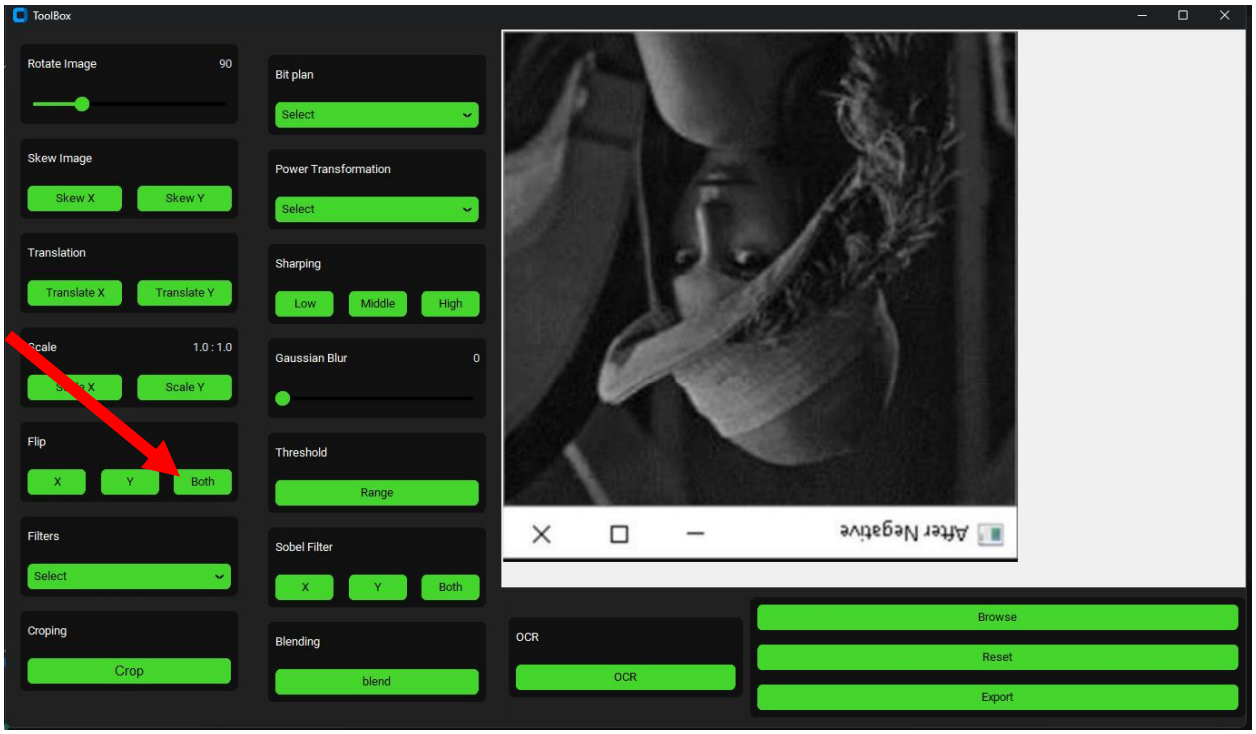
2. Flip Vertical



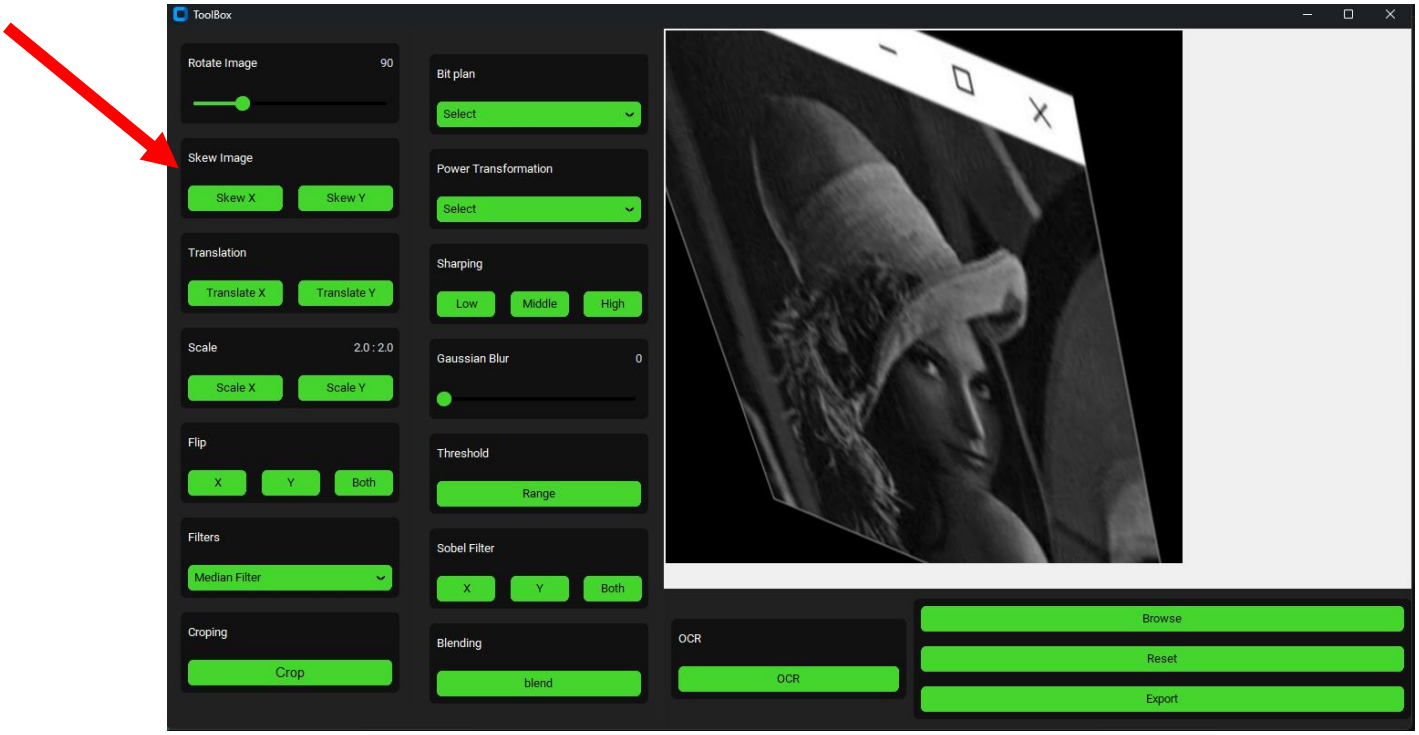
3. Flip horizontal



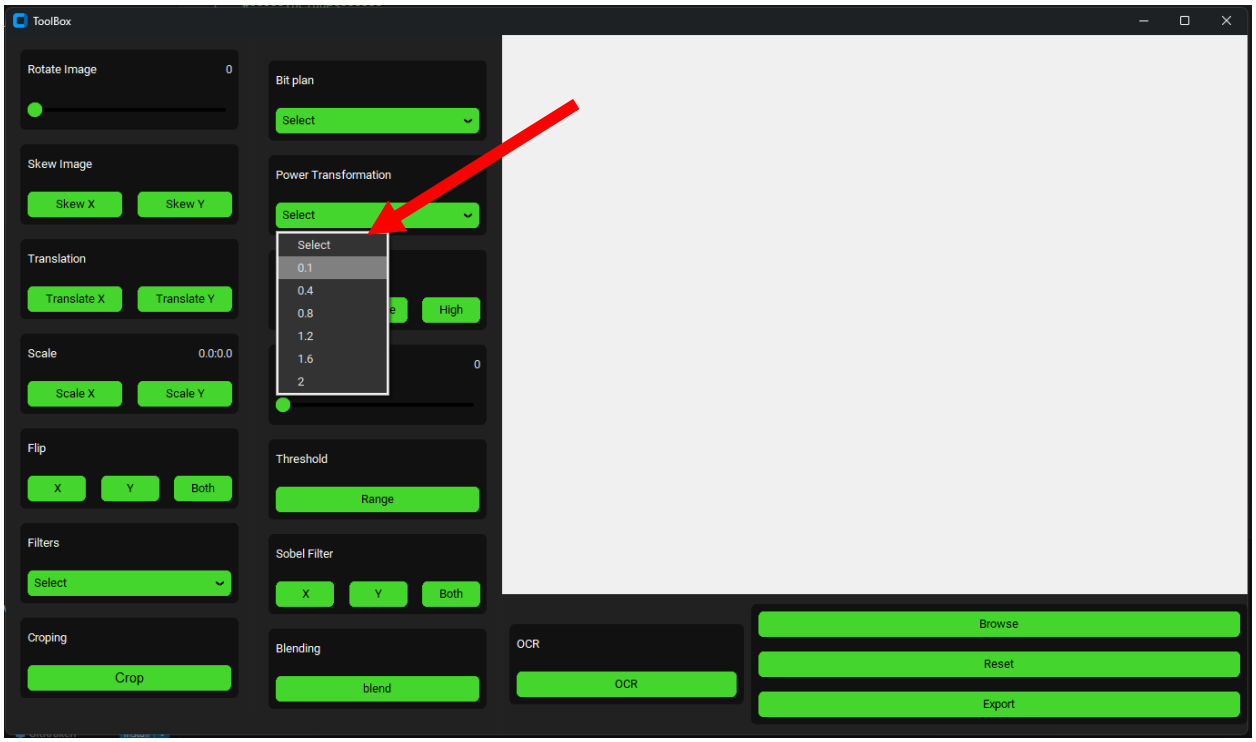
4. Flip Both



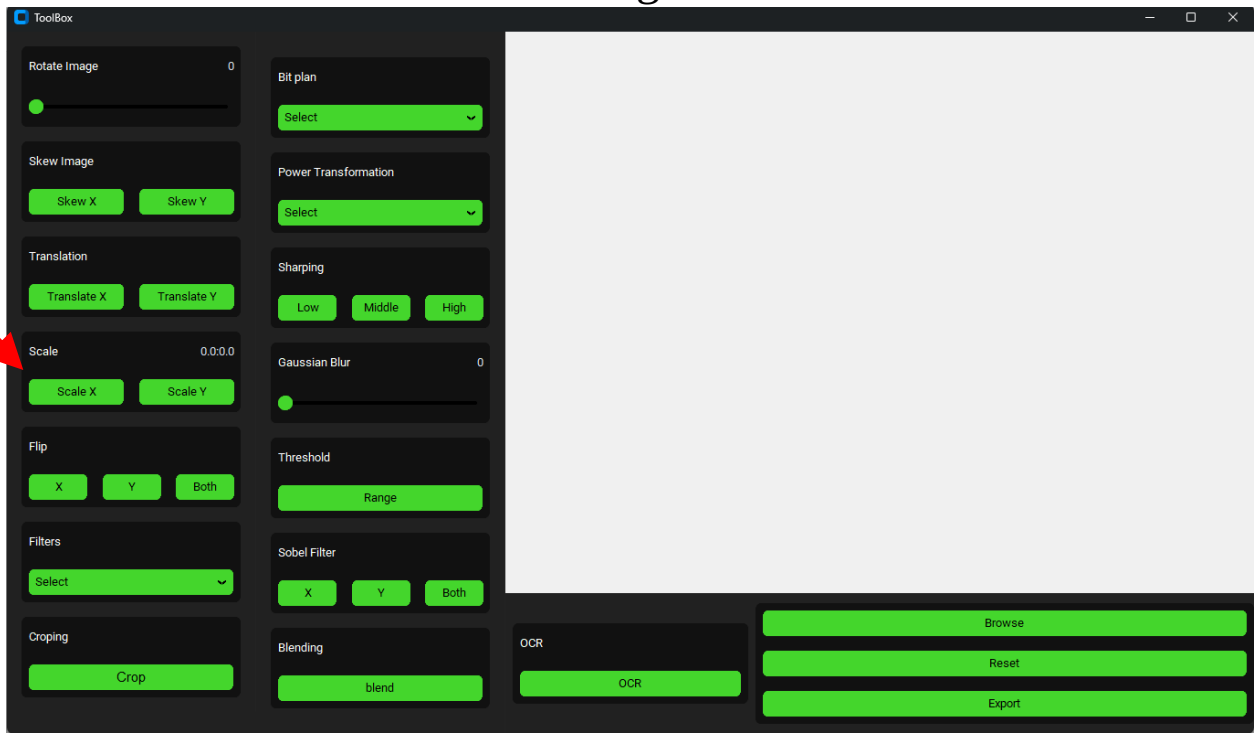
5. Skewing the image



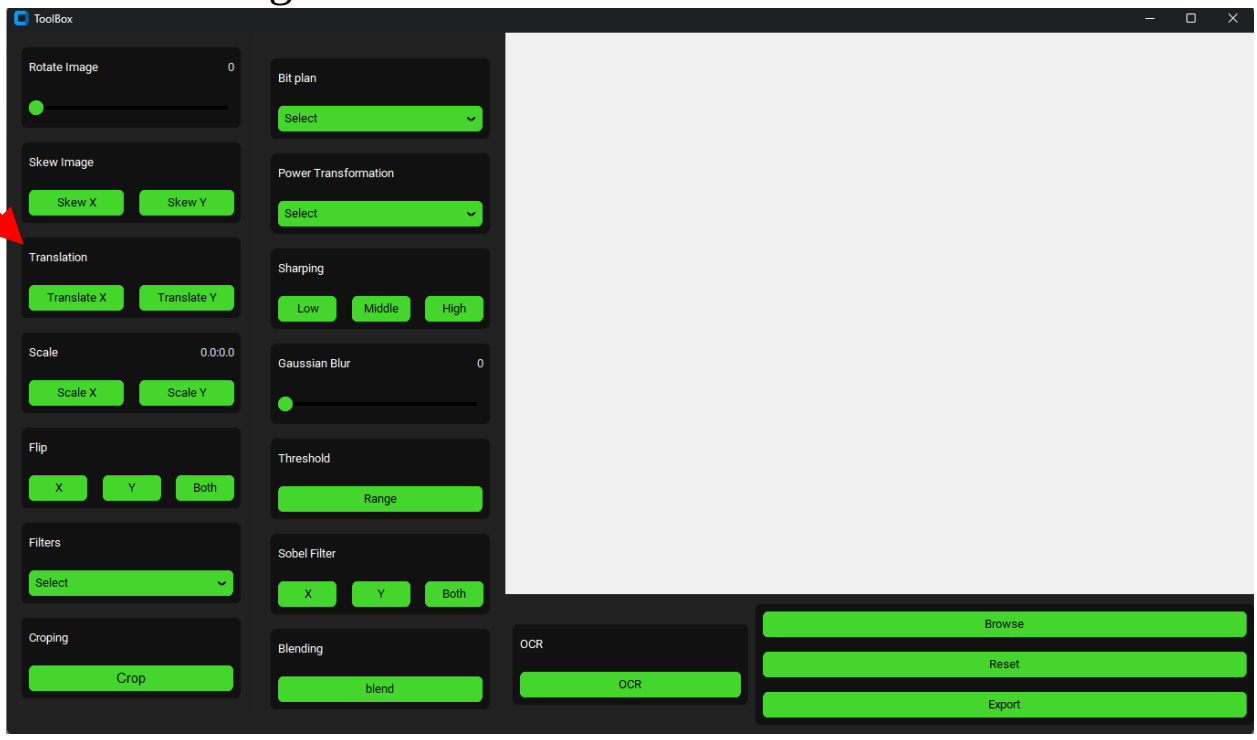
6. Power transformation filter by values (0.1,0.4,0.8,1.2,1.6,2)



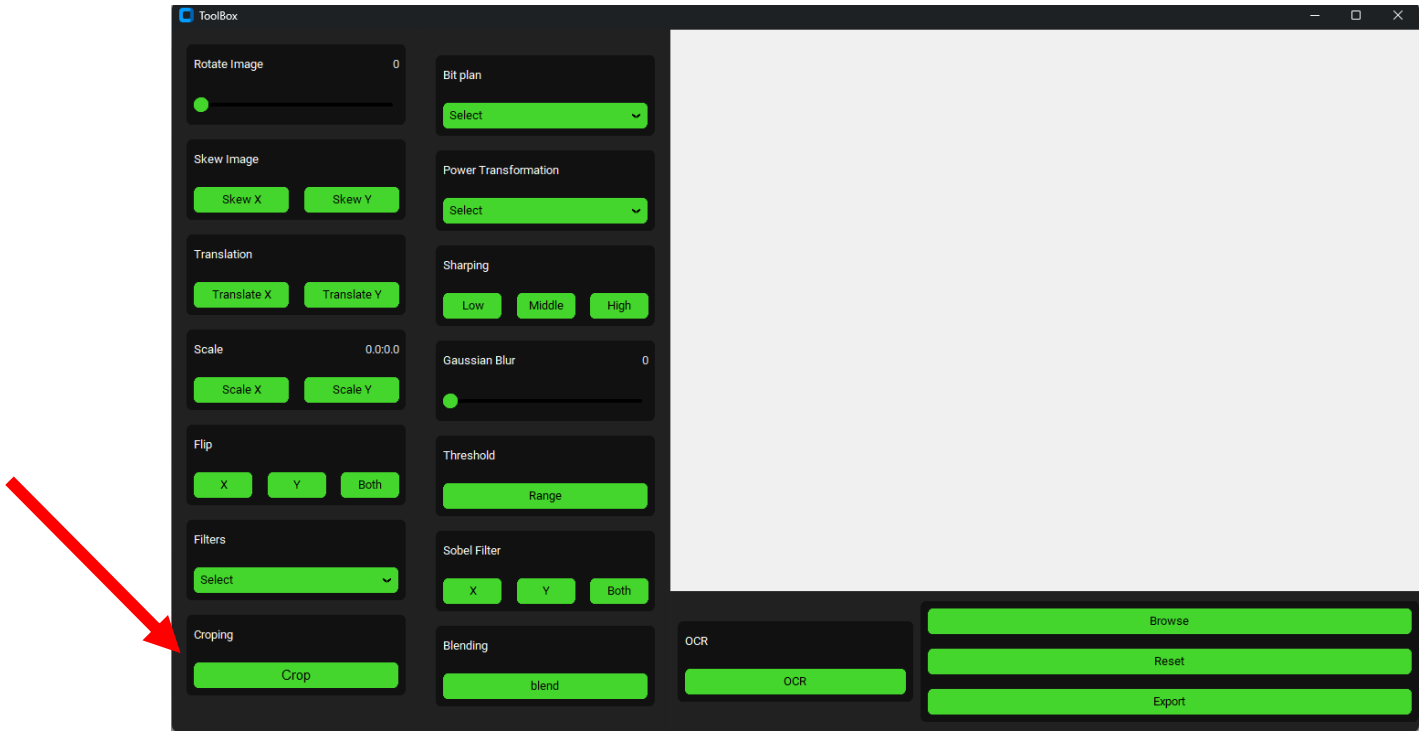
7. Increase or Decrease Size of image



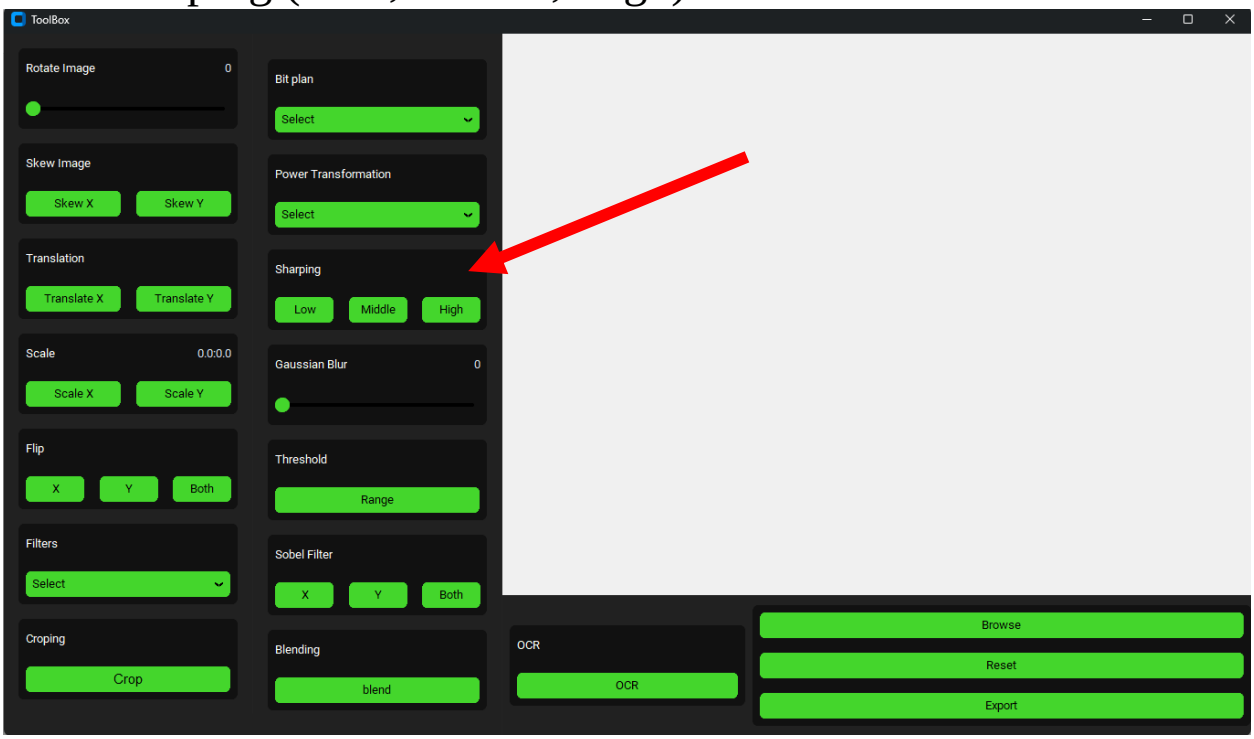
8. Translate image



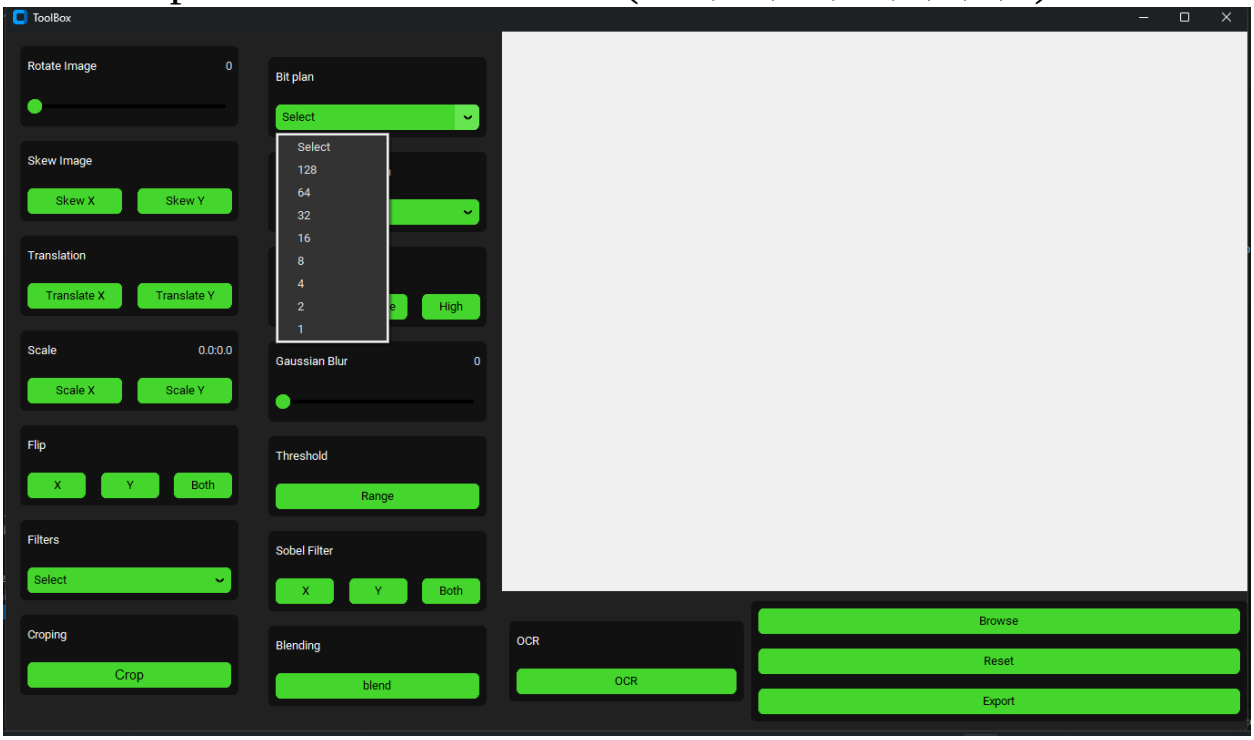
9. Cropping image



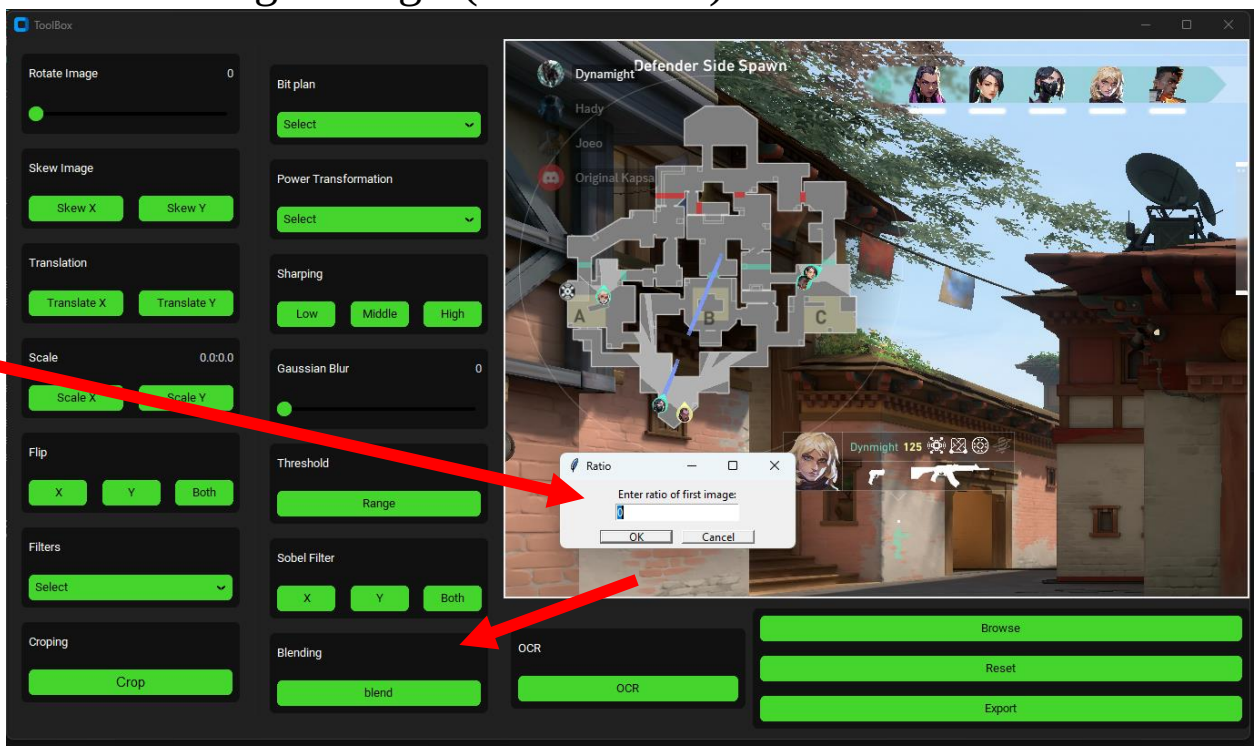
10. Sharping (Low, Middle, High)



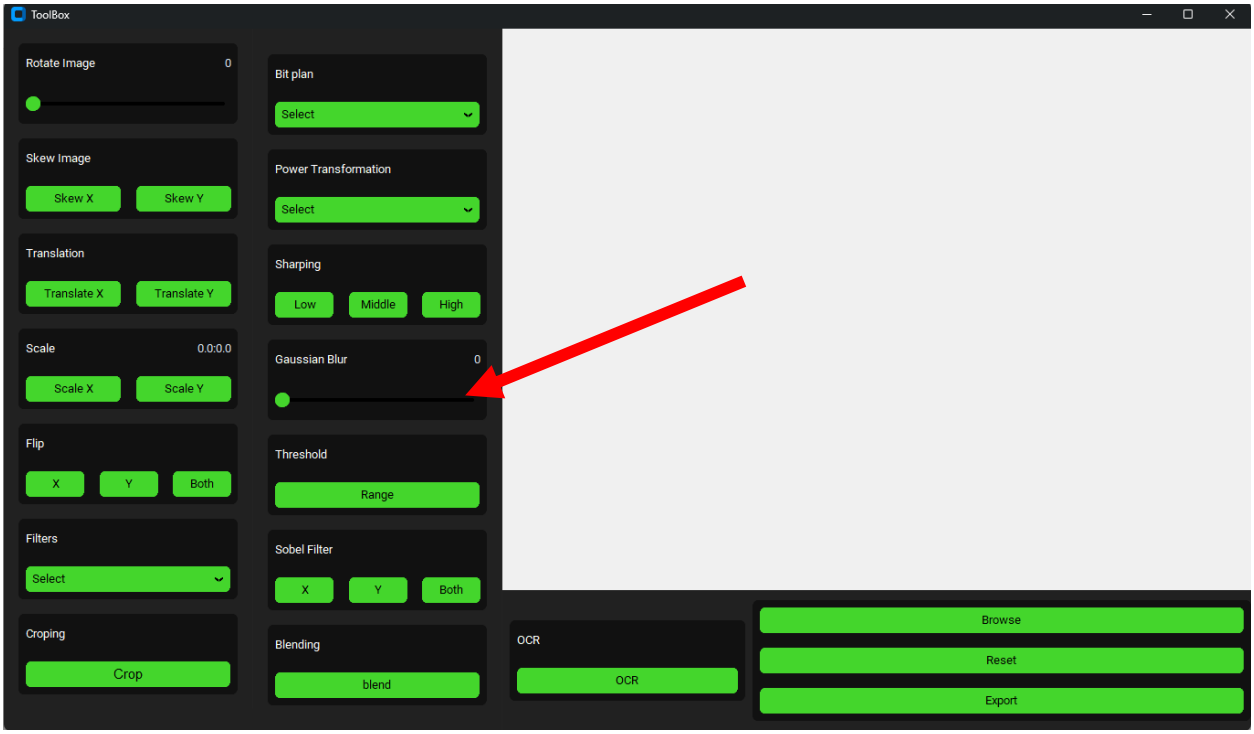
11. Bit plane filter with values (128,64,32,16,8,4,2,1)



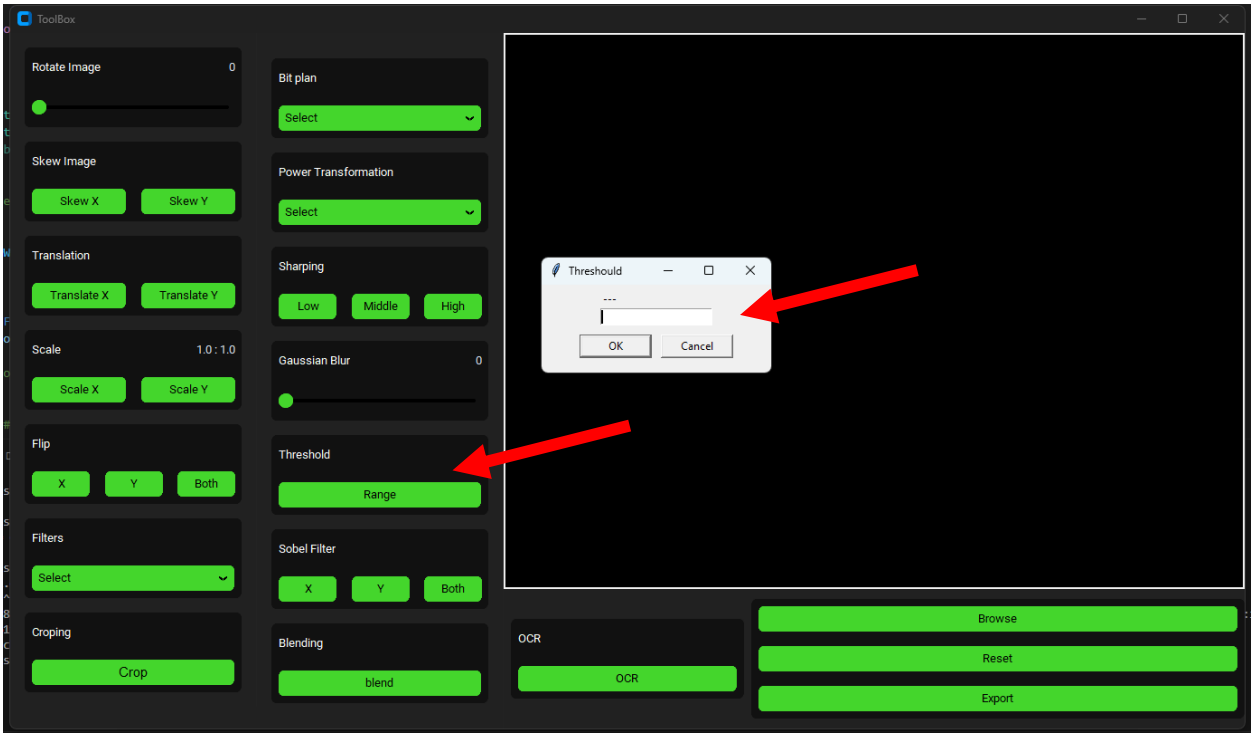
12. Blending 2 images(With Ratios)



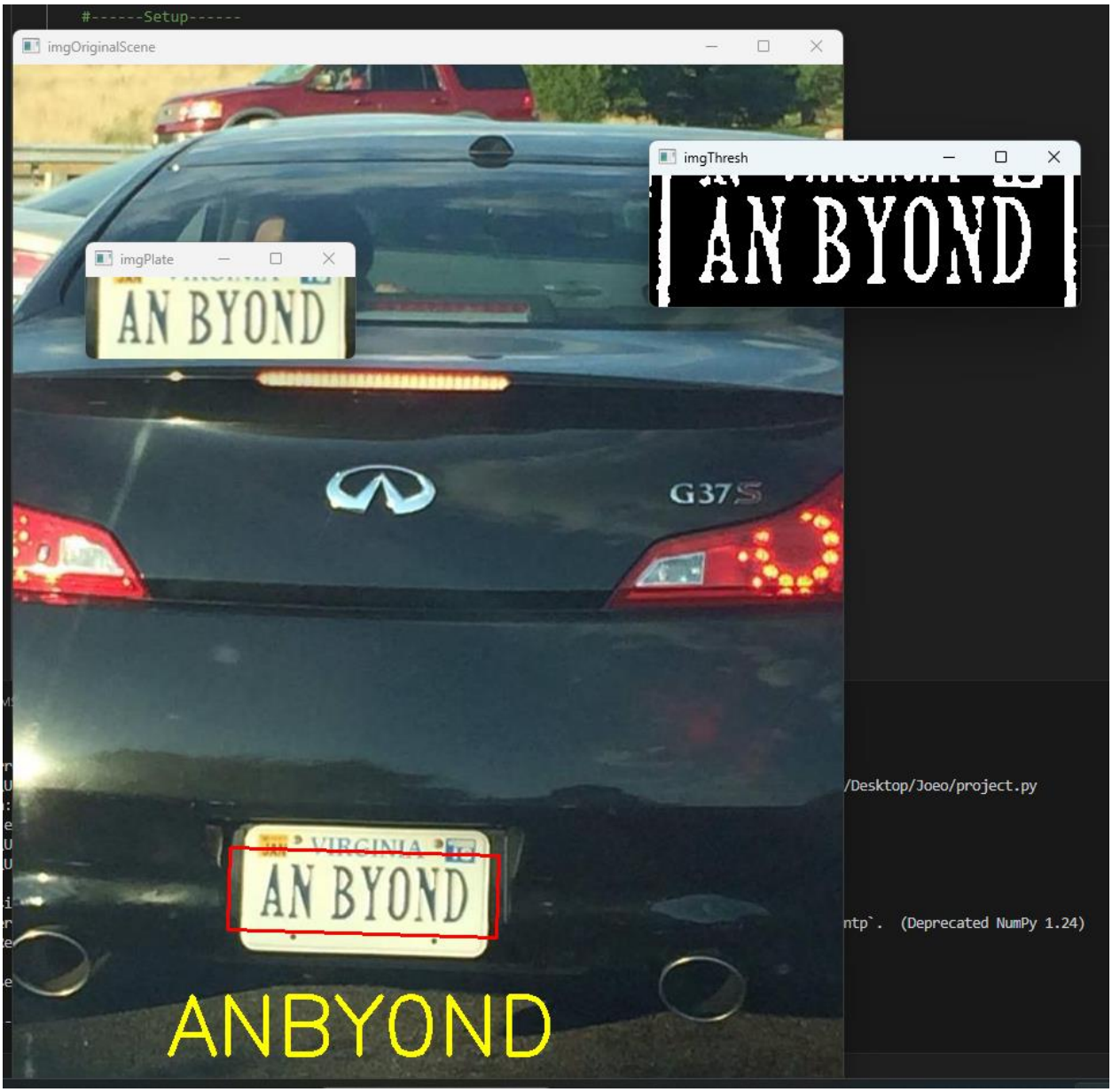
13. Gaussian Blur



14. Threshold(insert value)

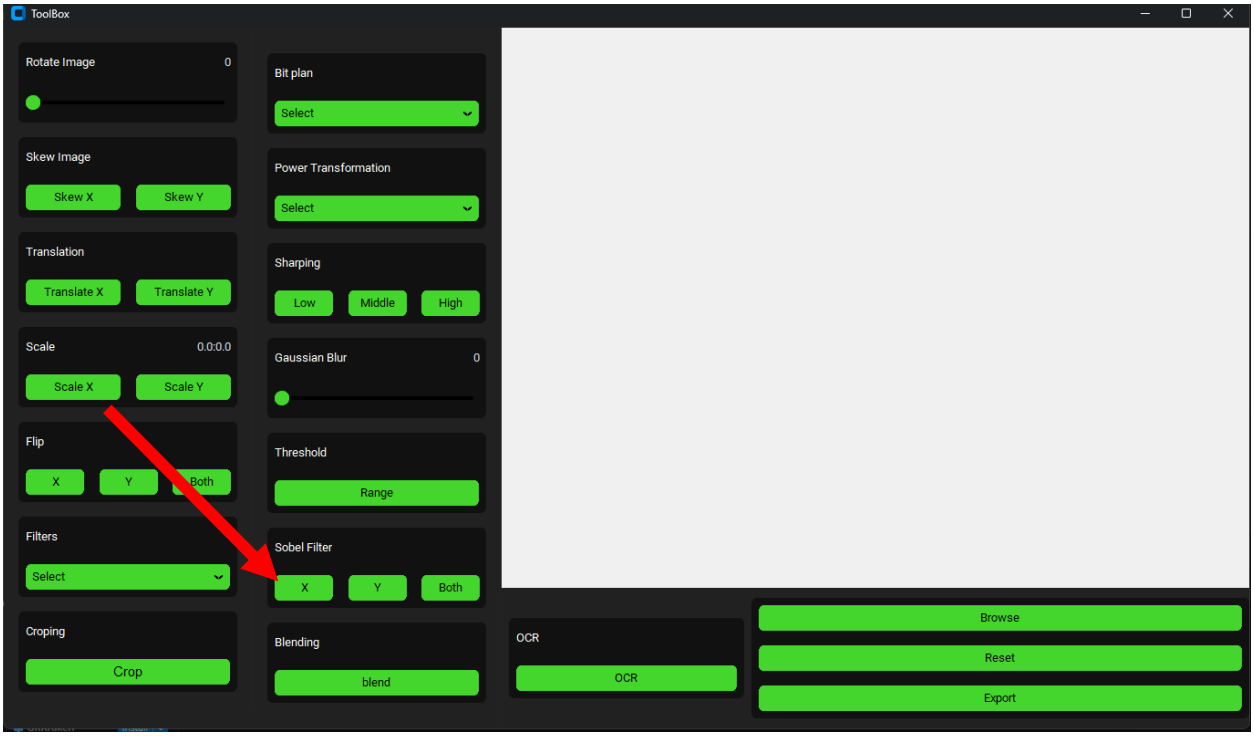


15. Car Plate Detection

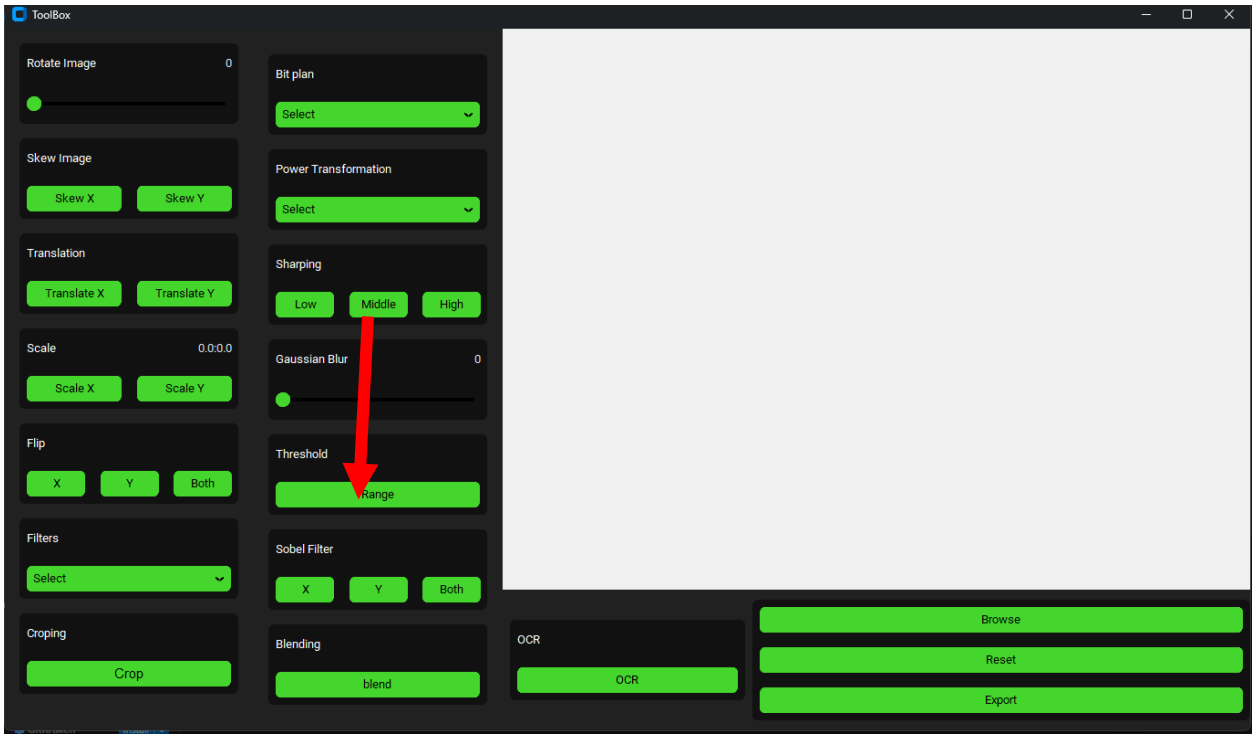


Edge detection part

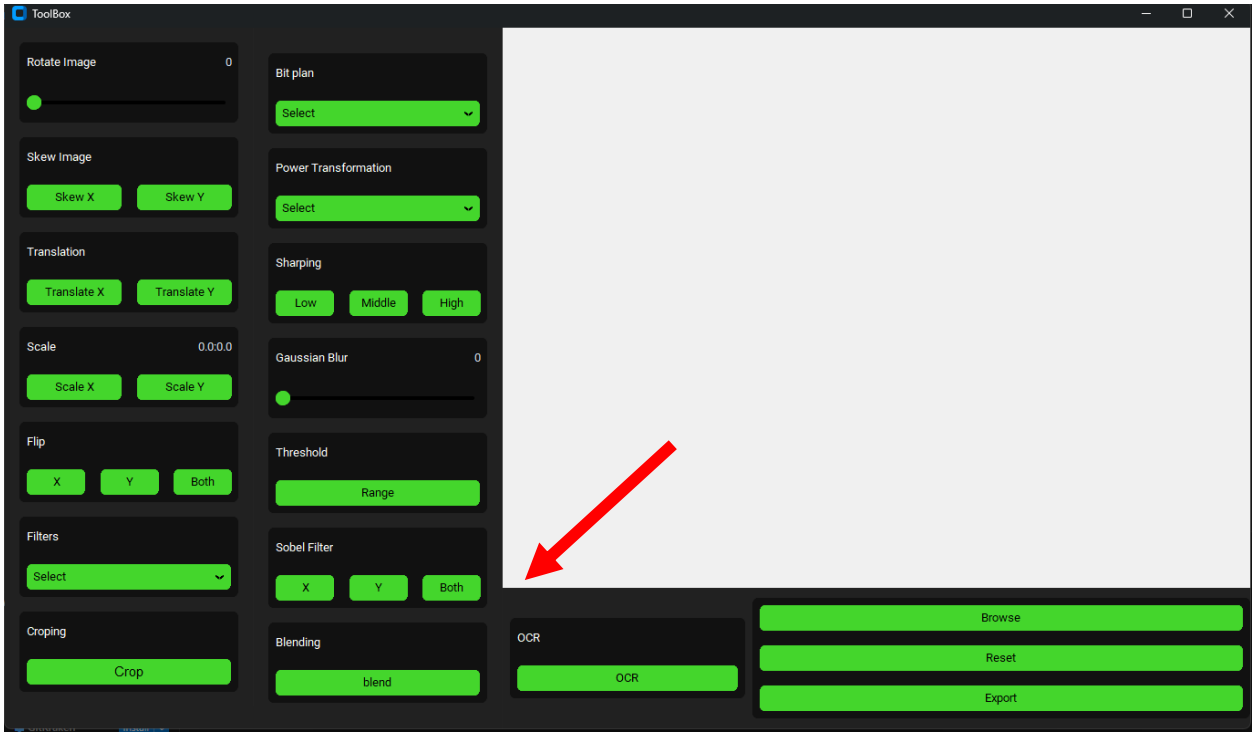
16. Sobel edge horizontal



17. Sobel edge vertical

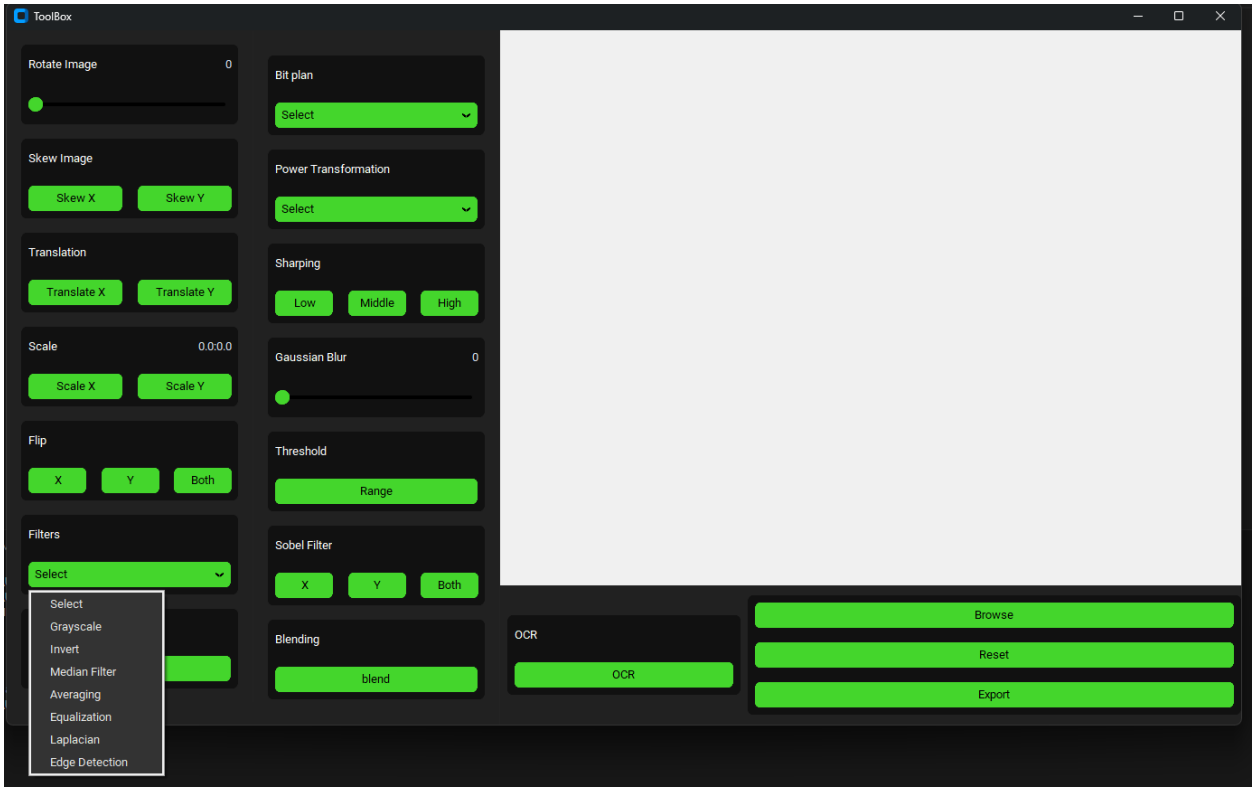


18. Sobel Both

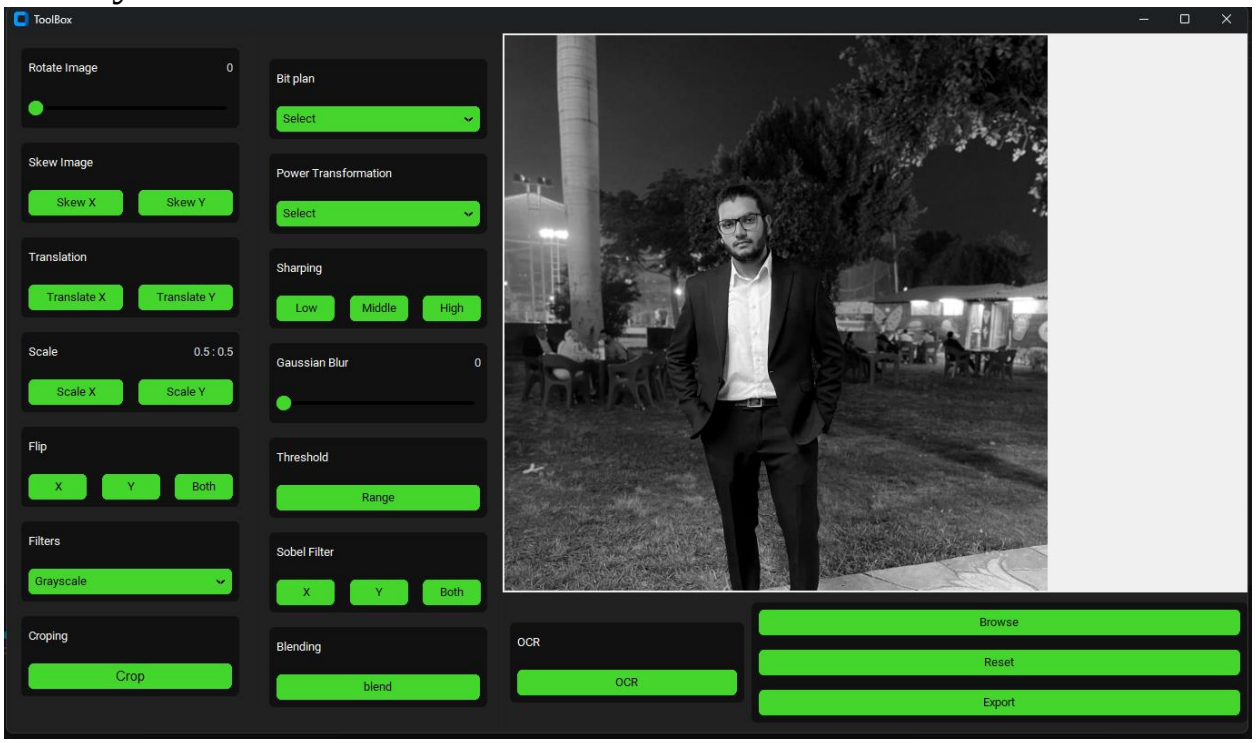


Filter part

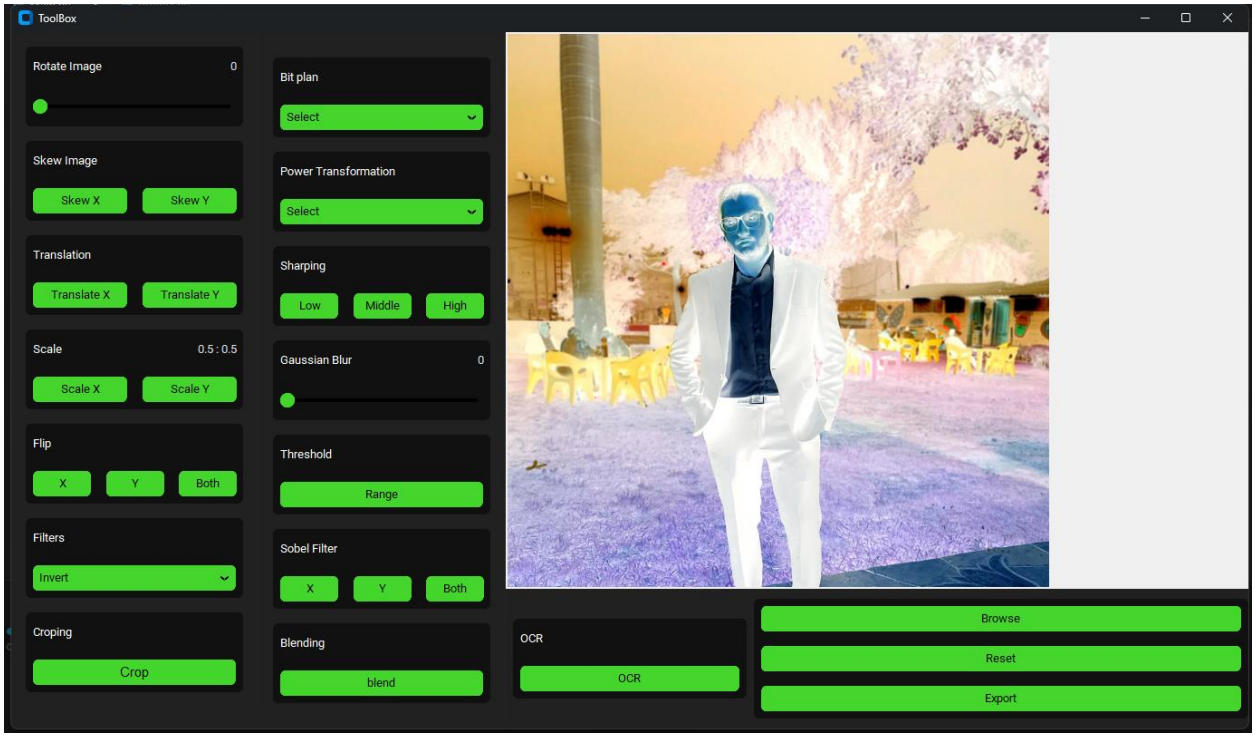
19. All available filters (Gray scale, invert, Median filter, Averaging filter, Equalization, Laplacian, Edge Detection) and scale to Motion blur filter.



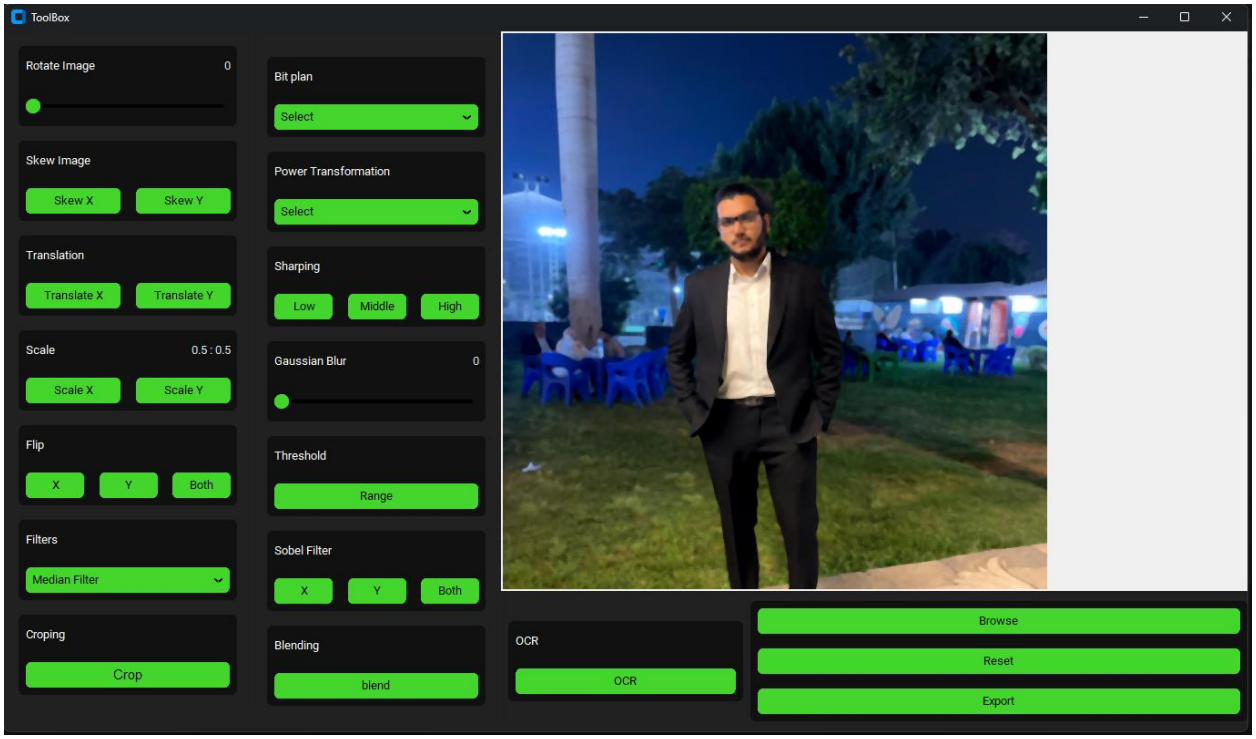
1.GrayScale



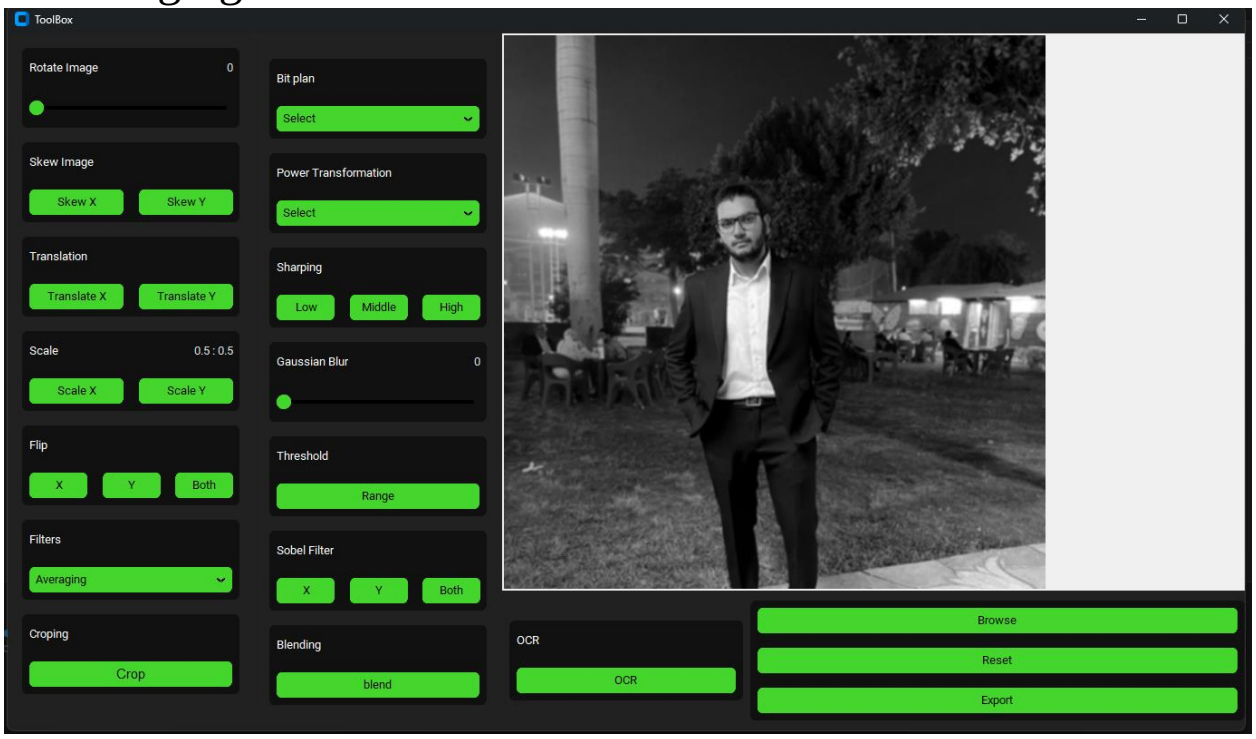
2.invert



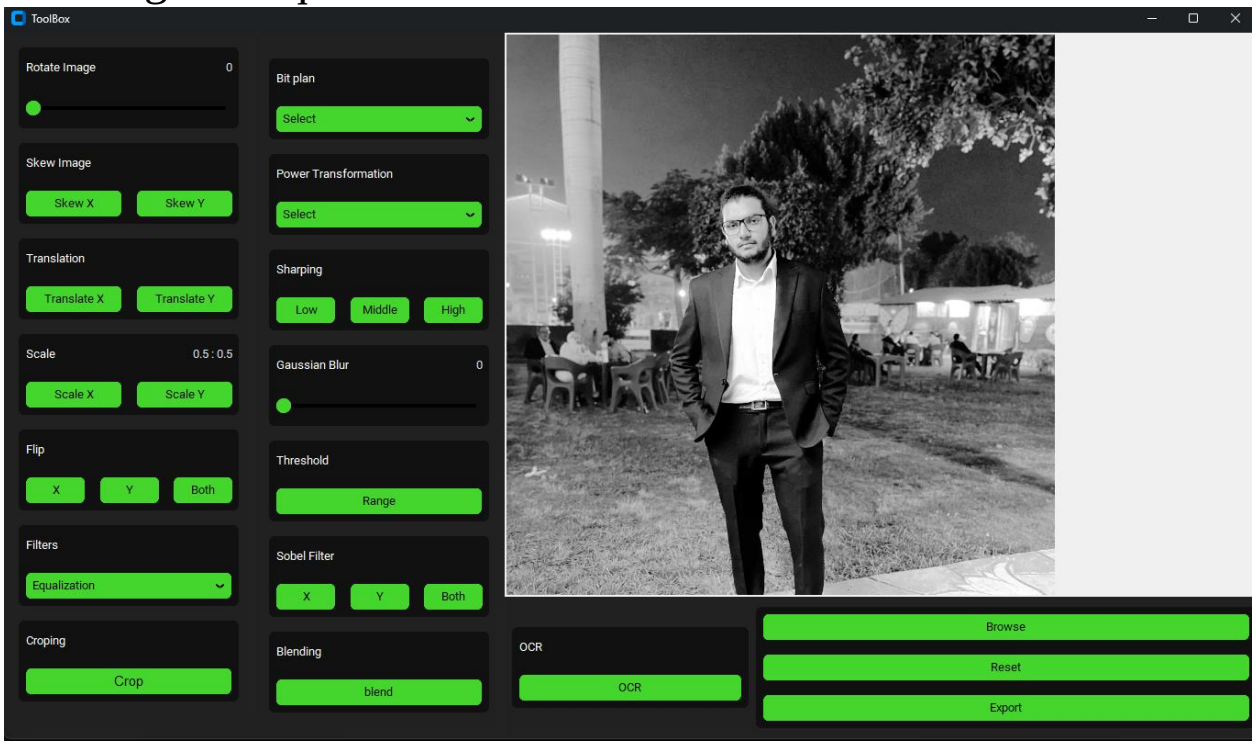
3.Median Filter



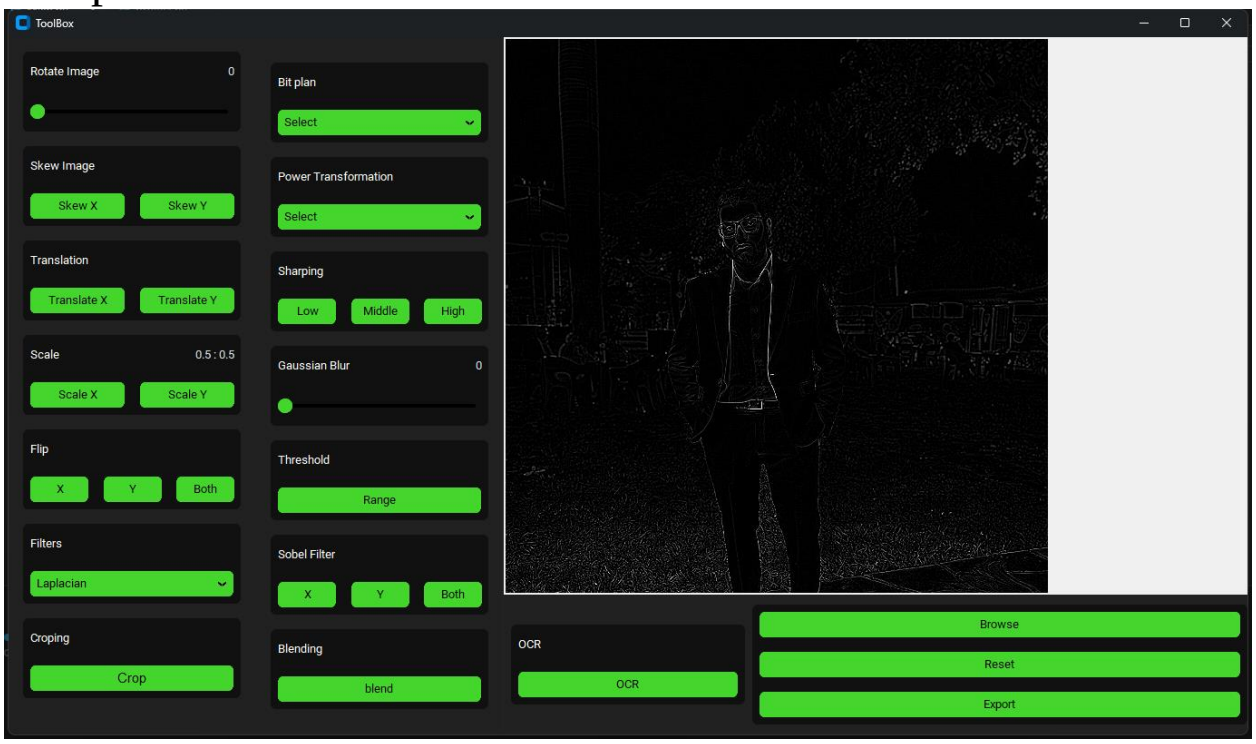
4.Averaging Filter



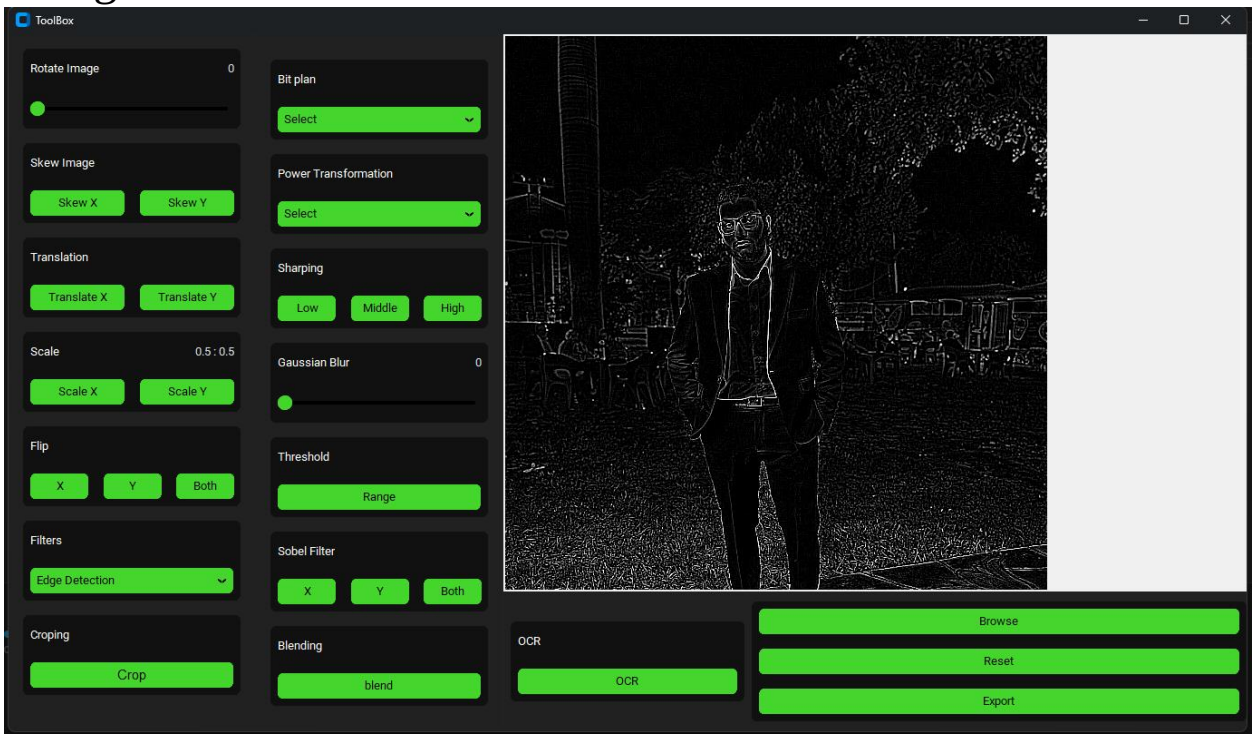
5.Histogram Equalization



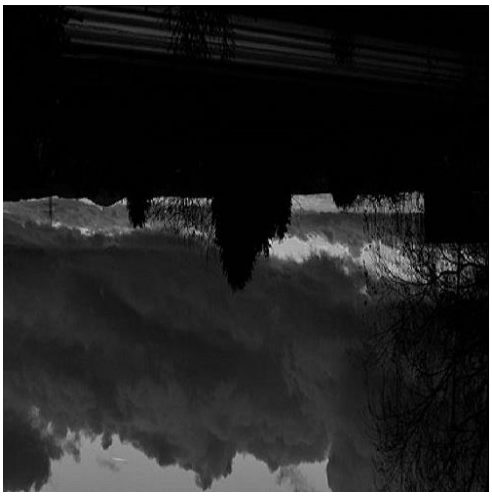
6.Laplacian



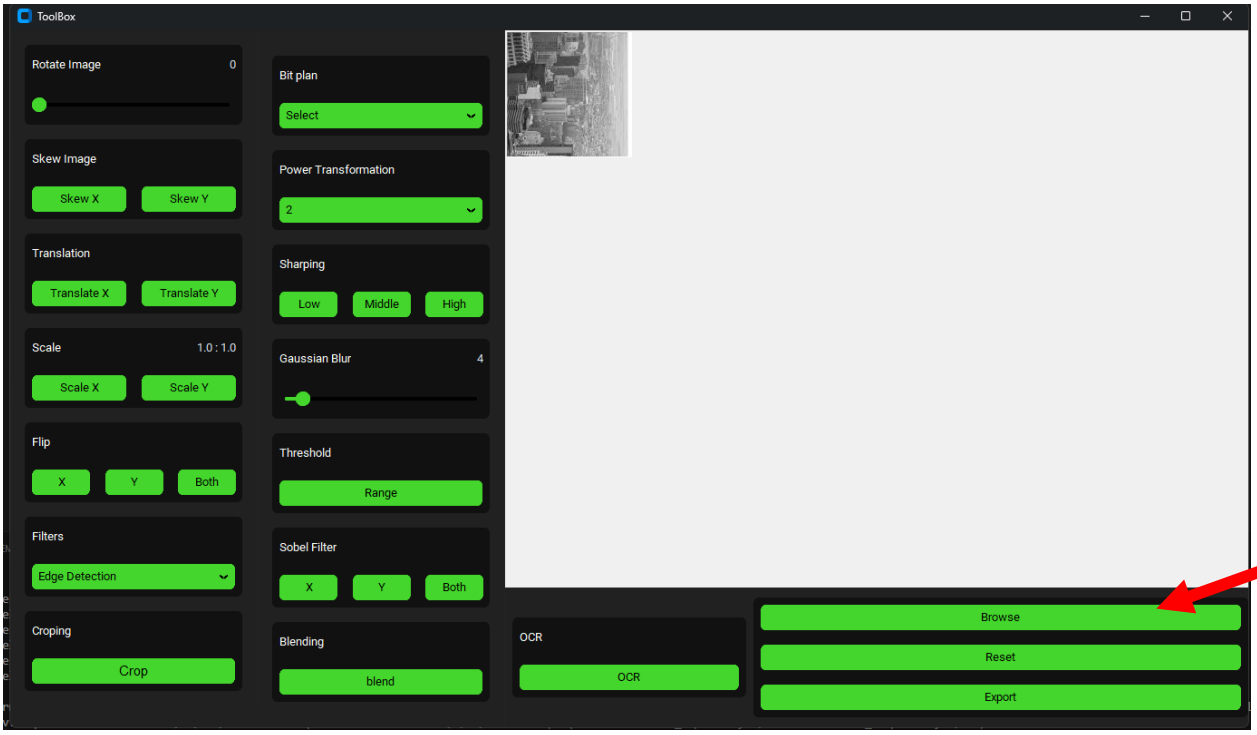
7.Edge Detection



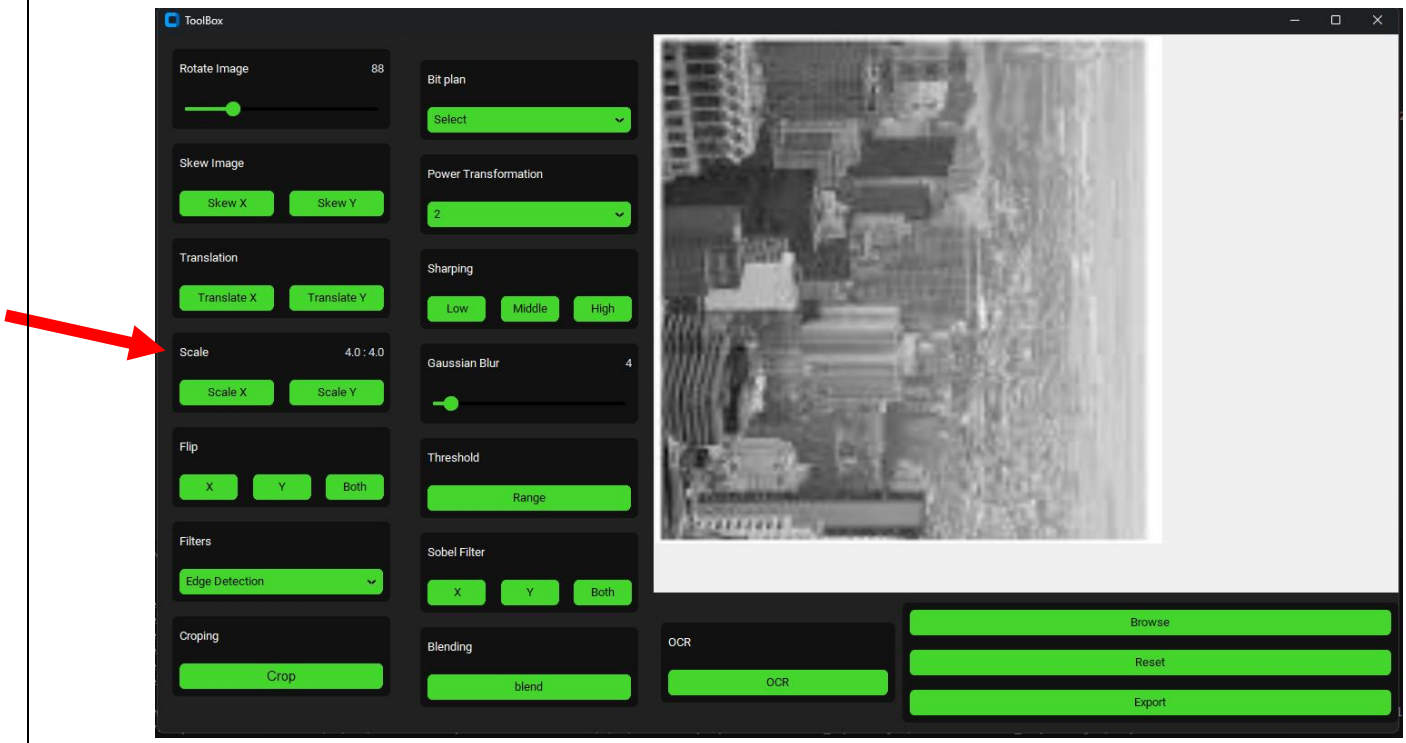
Let’s Try to Enhance These Two images



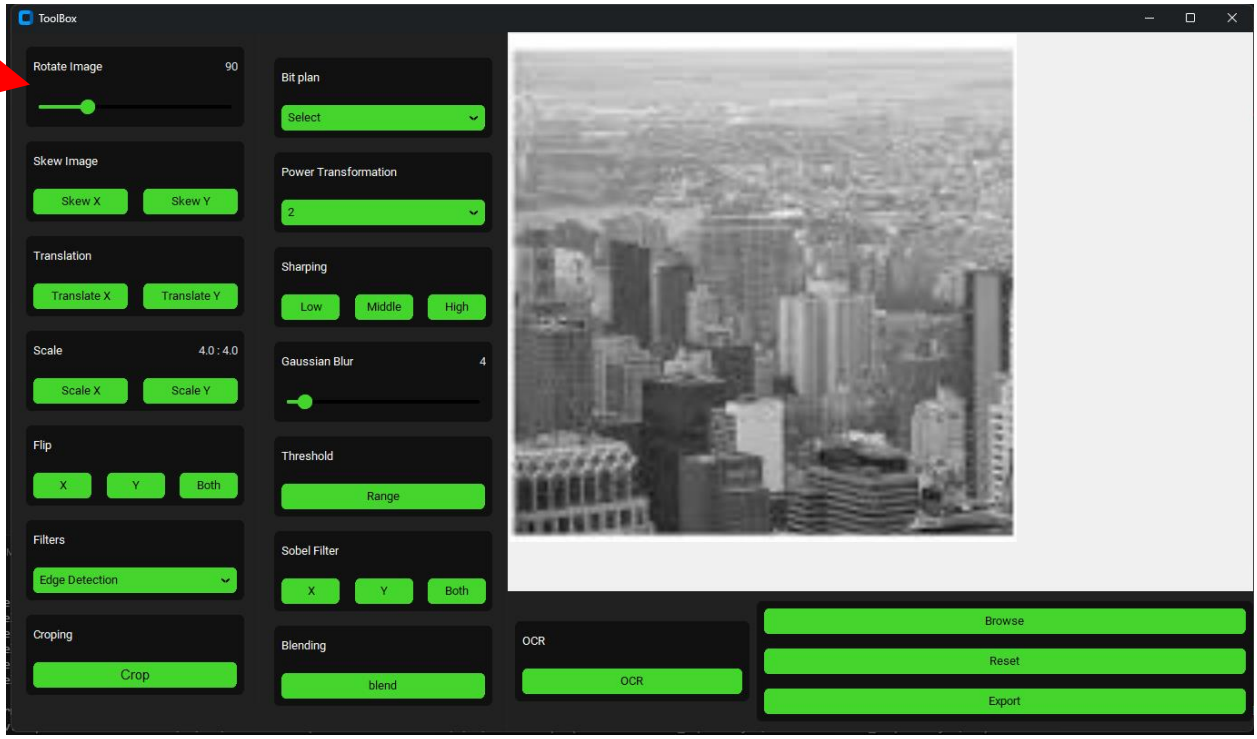
1.First Browse and select the image



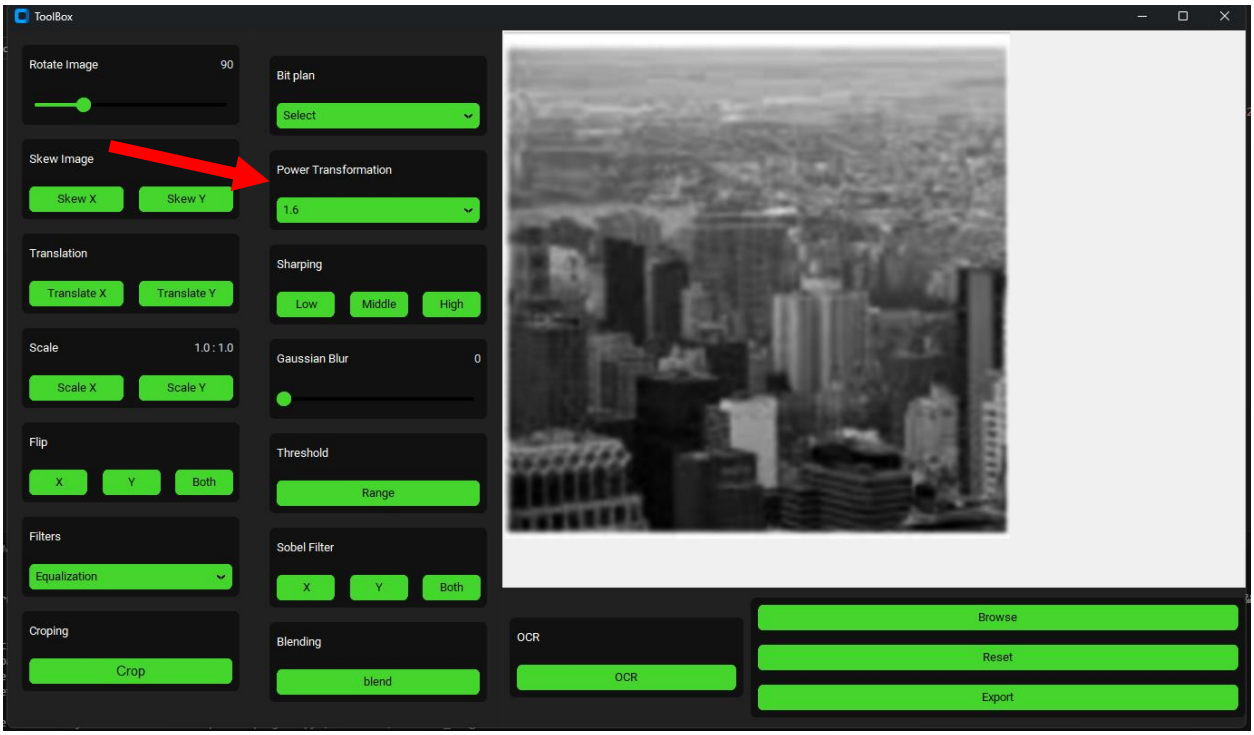
2.Scale X by 4 And Y by 4



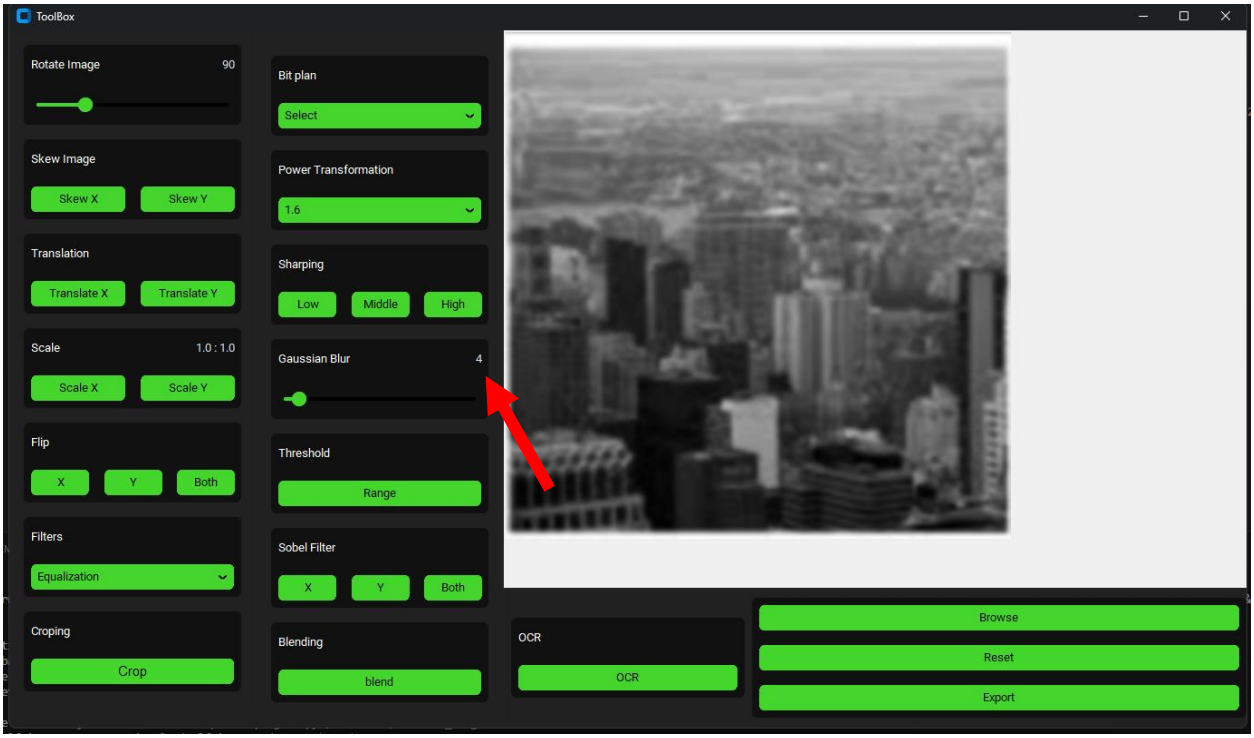
3. Rotate by 90



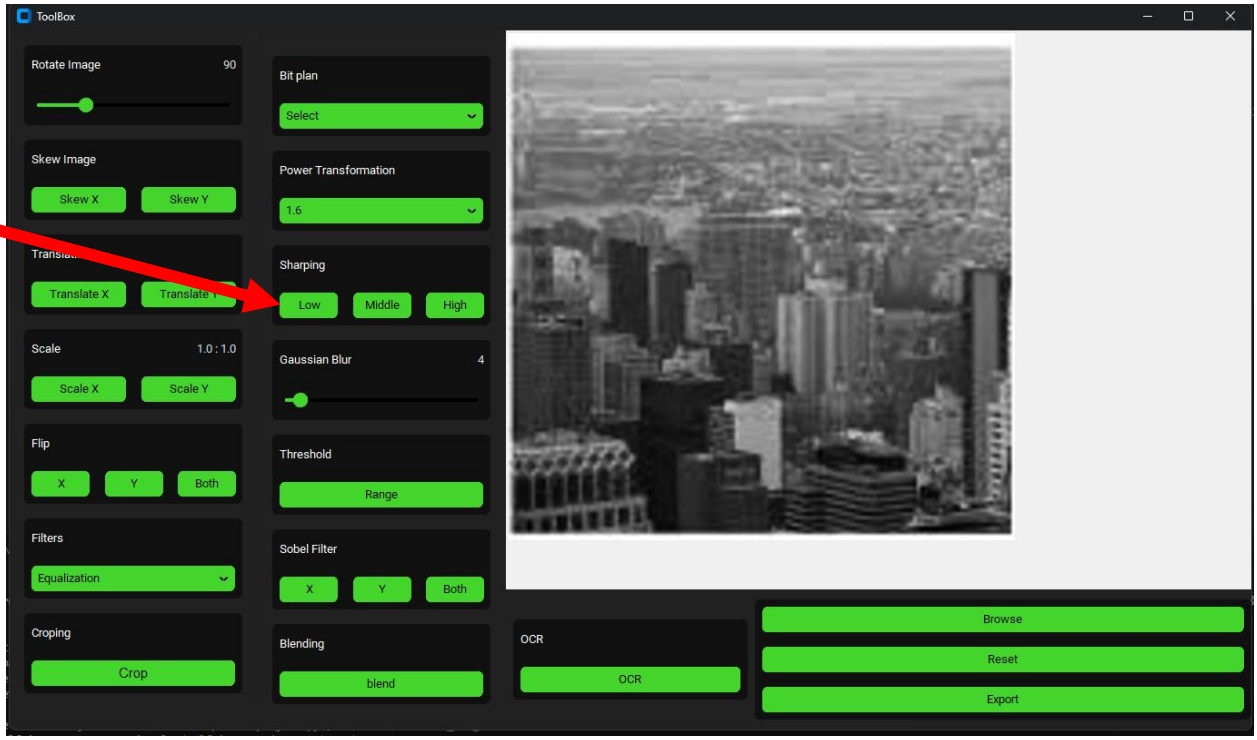
4.Set Power Transformation on 1.6



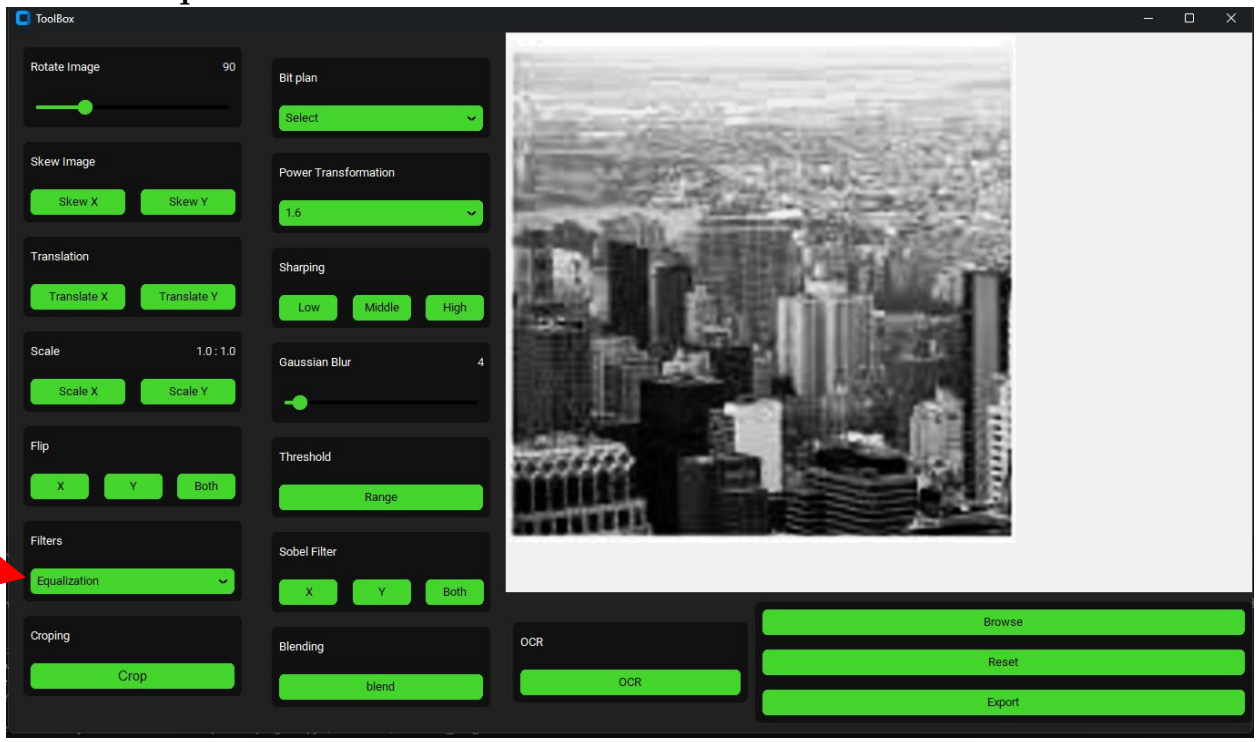
5. Set Gaussian blur on 4



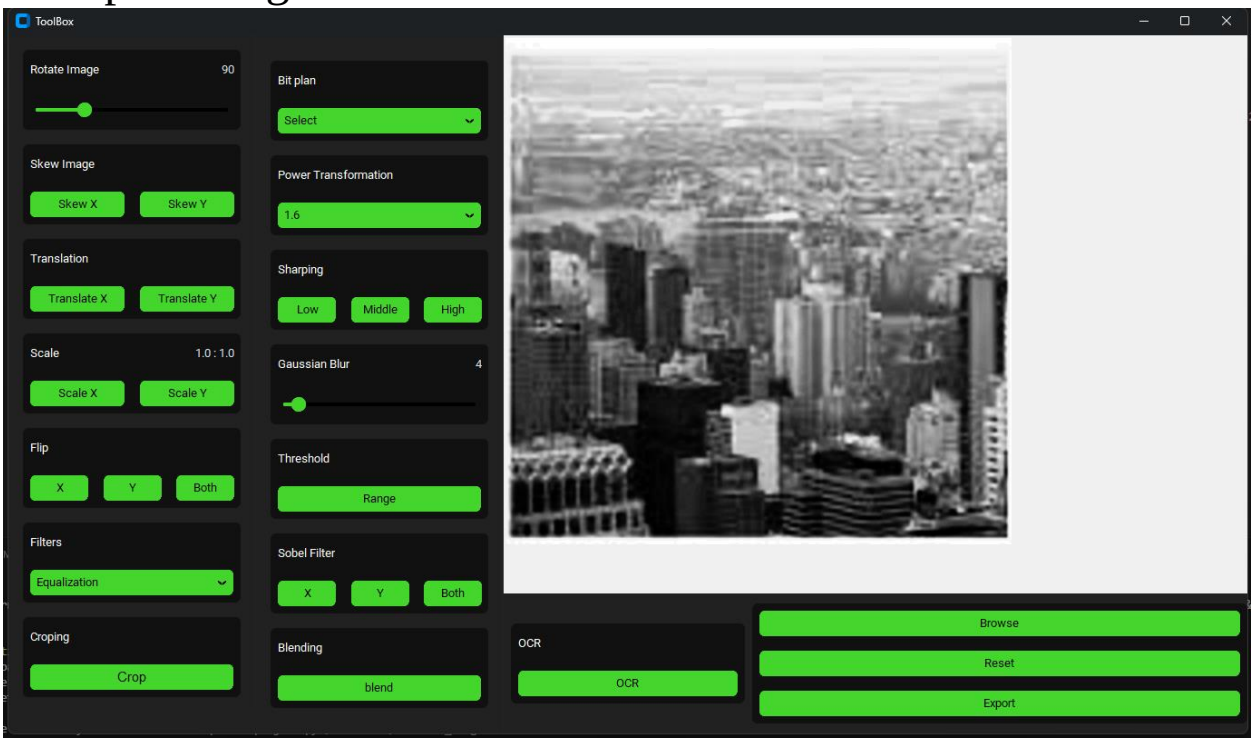
6. Apply Low Sharp



7.Add Equalization Filter



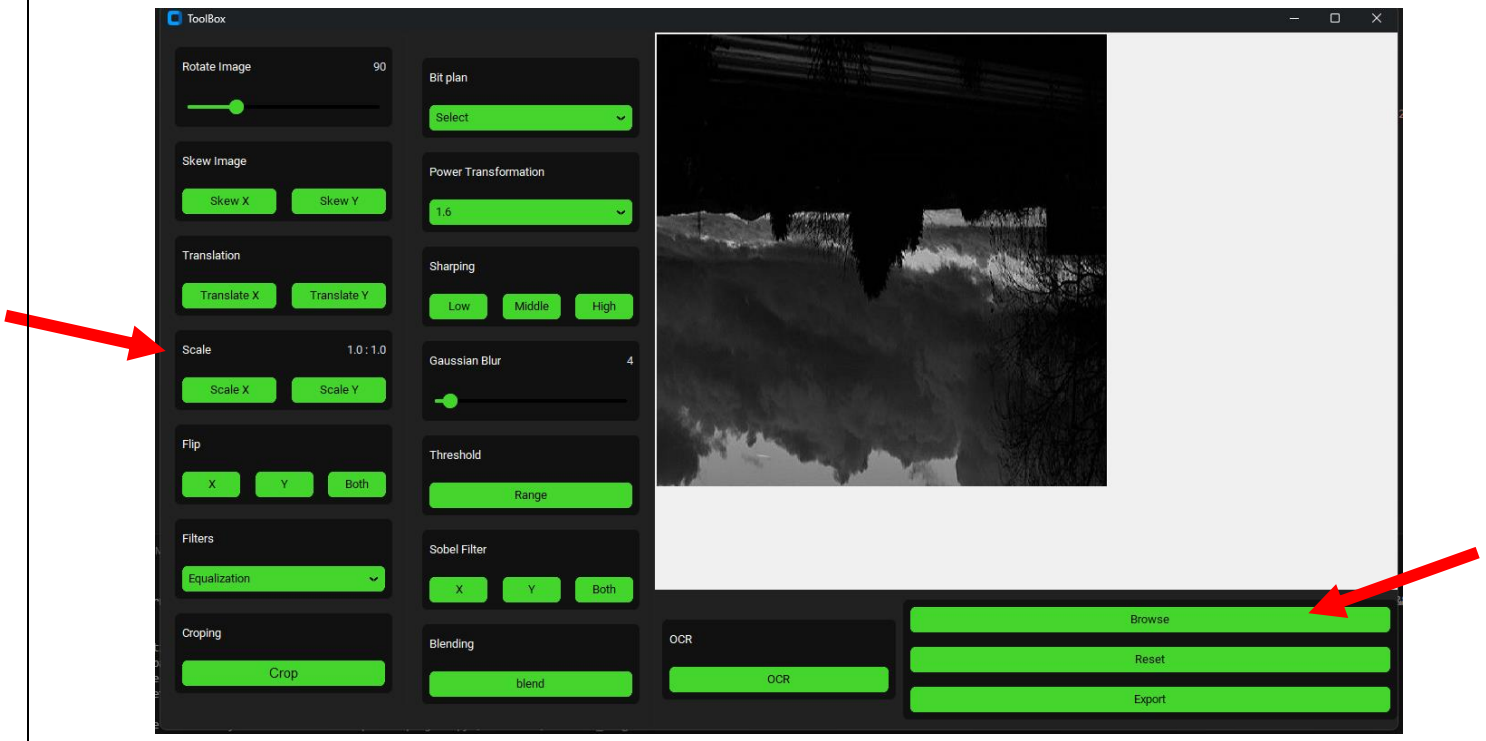
8. Export image



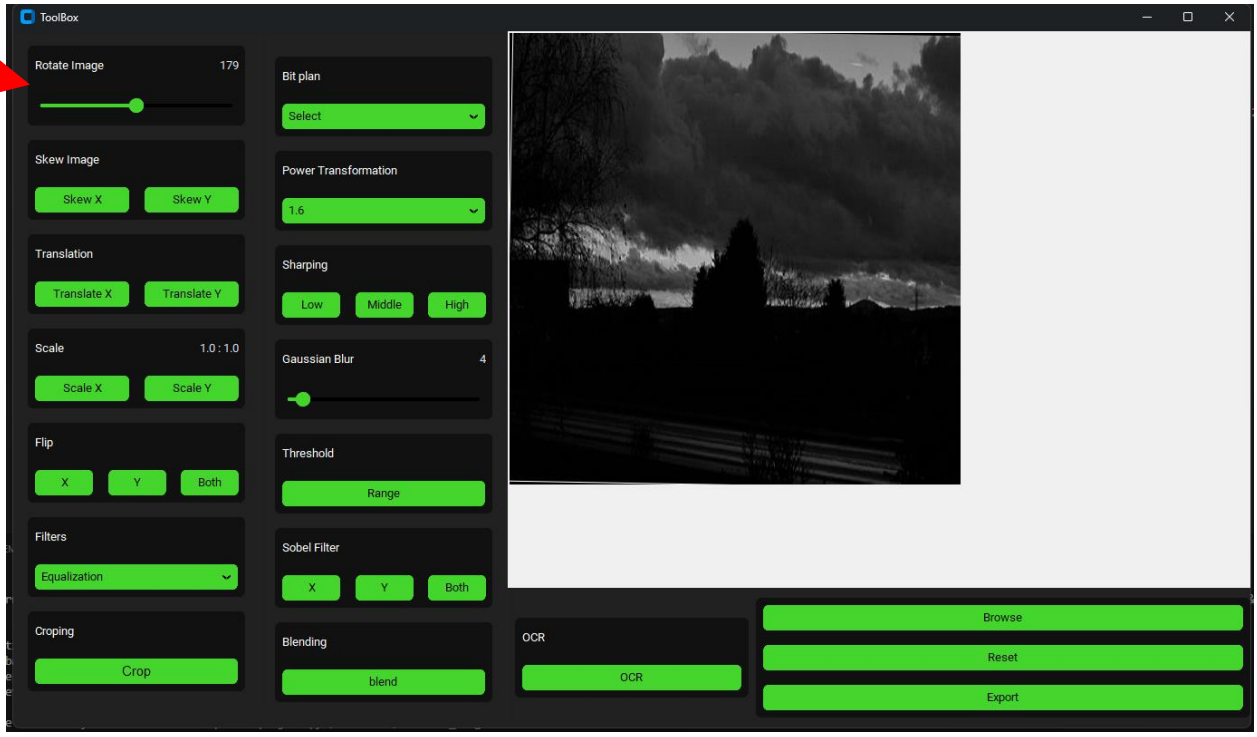
Here is The Final Result

Let’s try to enhance the second image

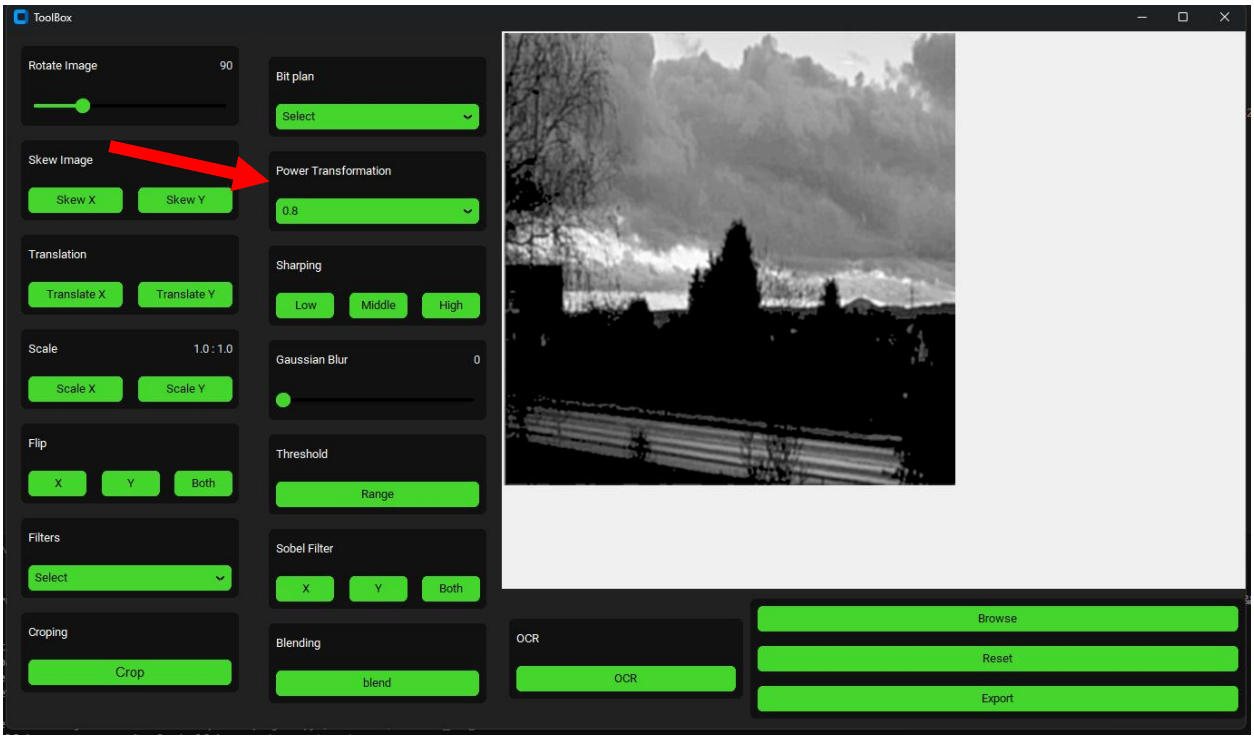
1.First Browse and select the image



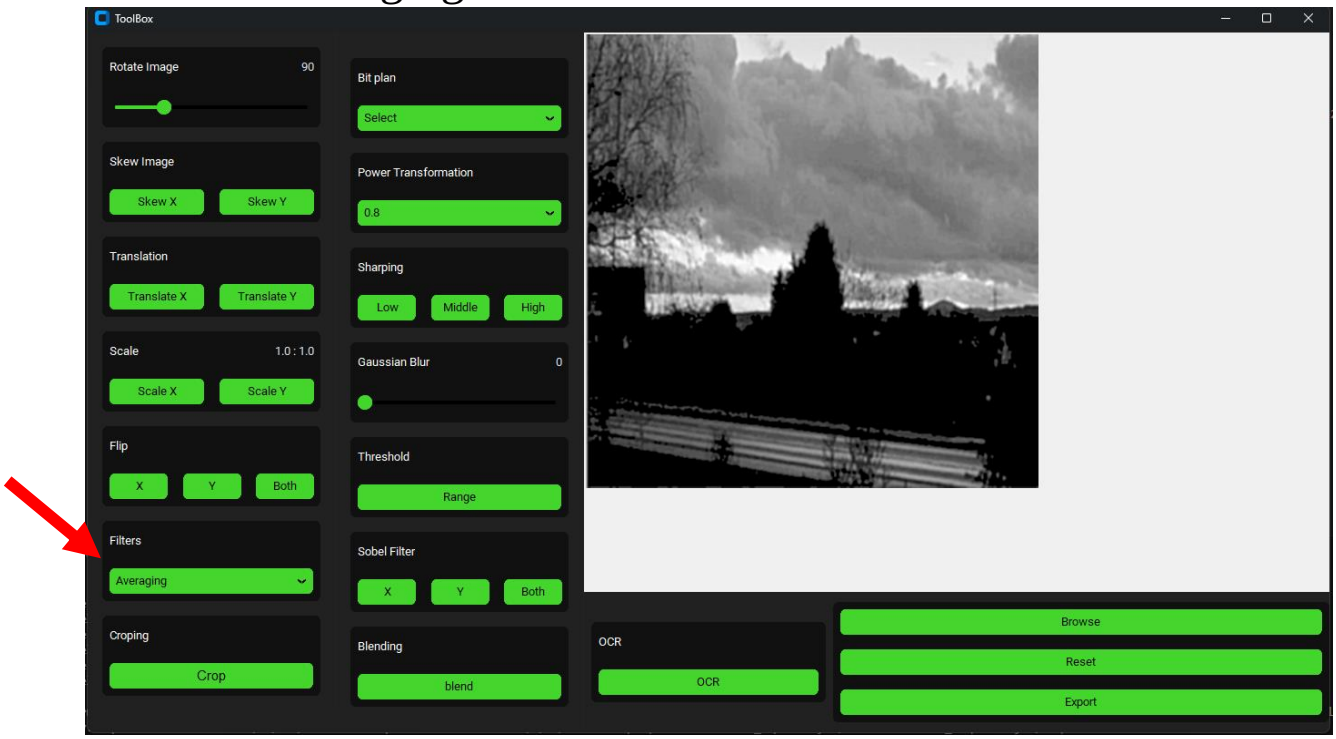
2. Rotate by 180



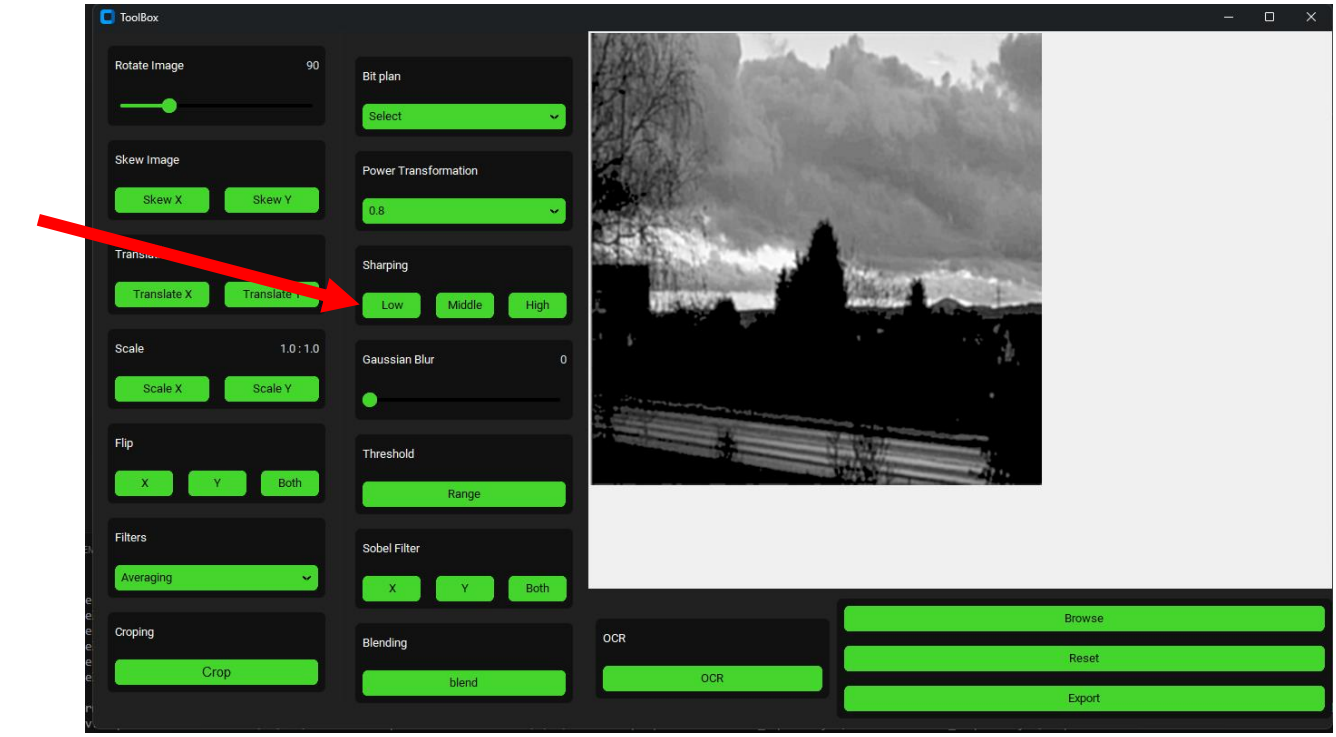
4.Set Power Transformation on 0.8



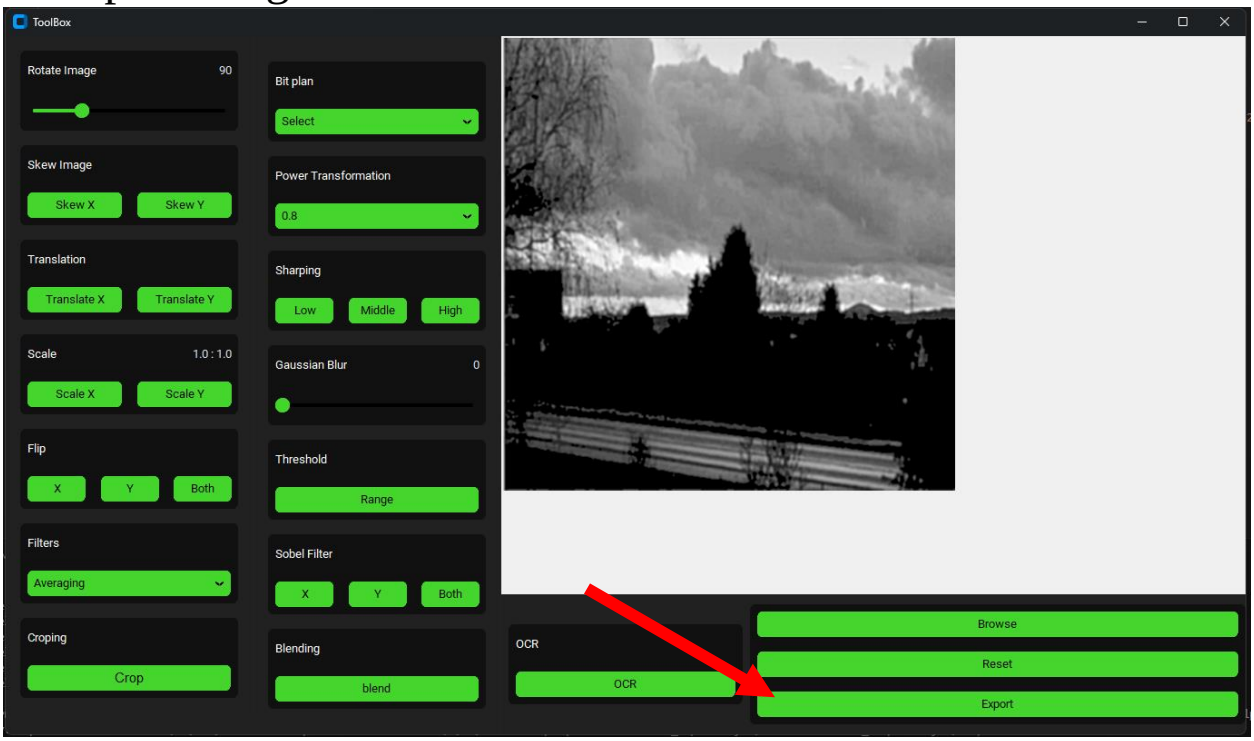
5. Select Averaging Filter



6. Apply Low Sharp



7. Export image



Here is The Final Result

End of Documentation.

Thank You ♥