BBM413-BBM415 - IMAGE PROCESSING COURSE PROJECT

Emre HANCI - 21604552
Sadık Can ACAR - 21626843

December 14, 2021

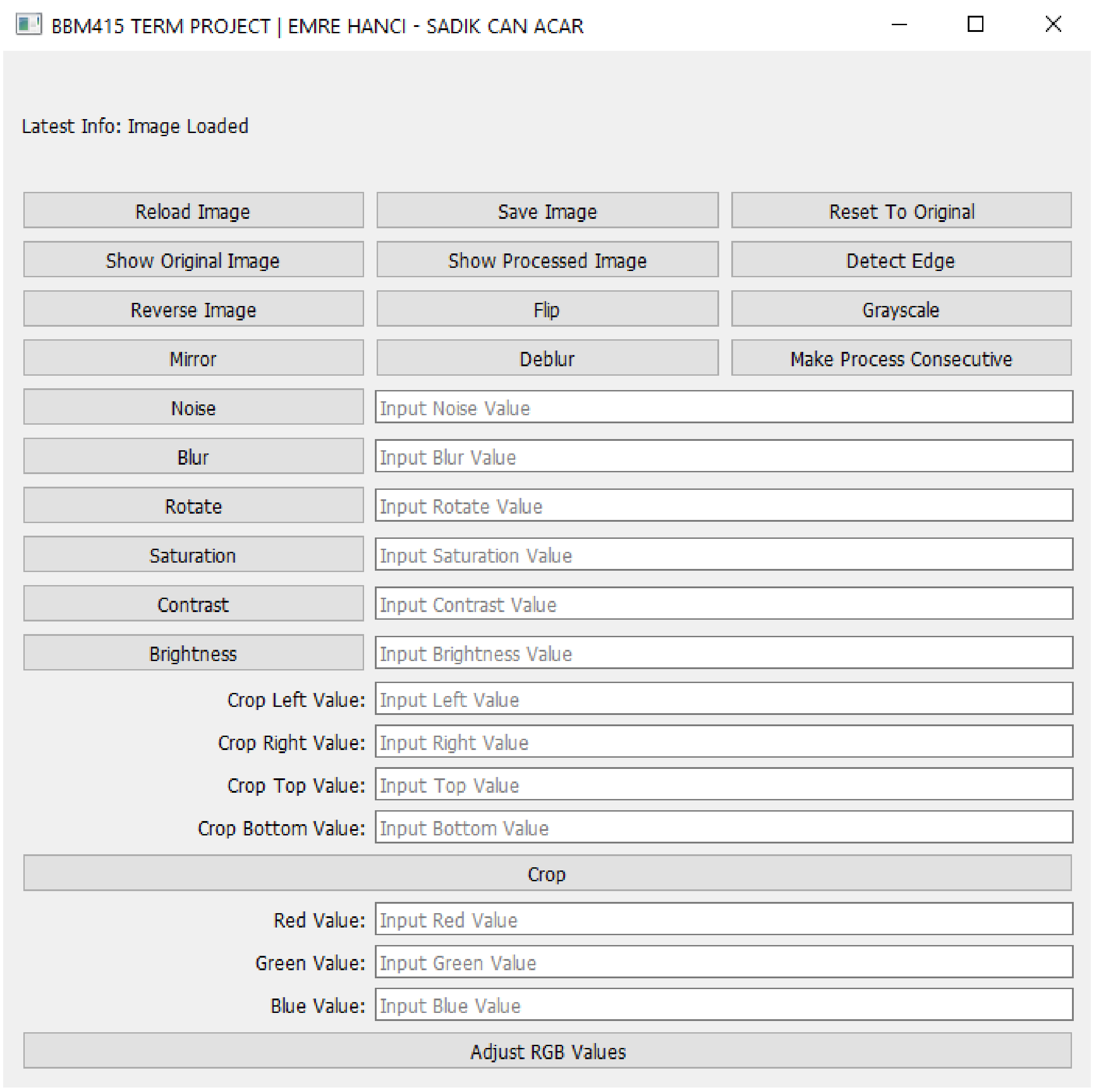
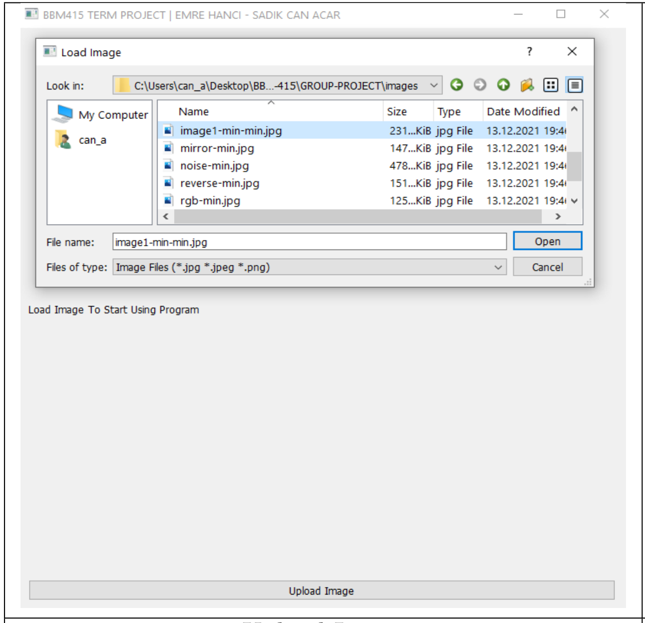


Figure 1: Image Editor Graphical User Interface



BBM415 TERM PROJECT | EMRE HANCI - SADIK CAN ACAR

Load Image

Look in: C:\Users\can_a\Desktop\BB...-415\GROUP-PROJECT\images

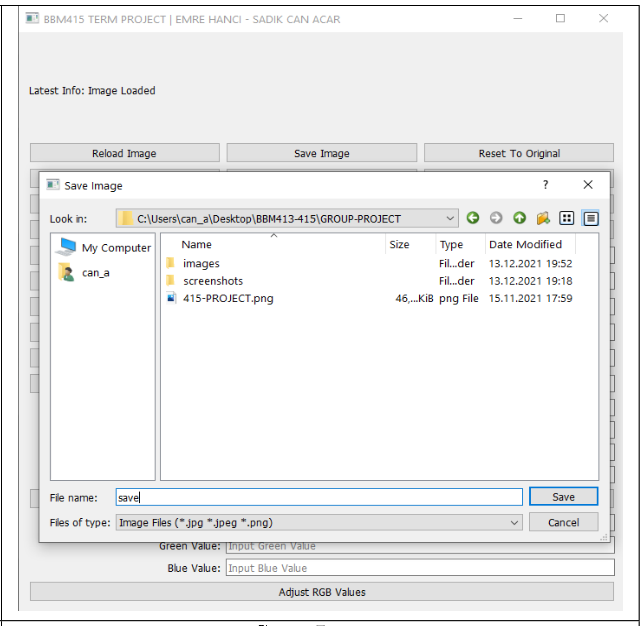
My Computer
can_a

Name	Size	Type	Date Modified
image1-min-min.jpg	231...KiB	jpg File	13.12.2021 19:4
mirror-min.jpg	147...KiB	jpg File	13.12.2021 19:4
noise-min.jpg	478...KiB	jpg File	13.12.2021 19:4
reverse-min.jpg	151...KiB	jpg File	13.12.2021 19:4
rgb-min.jpg	125...KiB	jpg File	13.12.2021 19:4

File name: image1-min-min.jpg
Files of type: Image Files (*.jpg *.jpeg *.png)

Load Image To Start Using Program

Upload Image



BBM415 TERM PROJECT | EMRE HANCI - SADIK CAN ACAR

Latest Info: Image Loaded

Save Image

Look in: C:\Users\can_a\Desktop\BBM413-415\GROUP-PROJECT

My Computer
can_a

Name	Size	Type	Date Modified
images		Fil...der	13.12.2021 19:52
screenshots		Fil...der	13.12.2021 19:18
415-PROJECT.png	46,...KiB	png File	15.11.2021 17:59

File name: save
Files of type: Image Files (*.jpg *.jpeg *.png)

Green Value: Input Green Value
Blue Value: Input Blue Value

Save Image

In our Image Editor, if we try to load an image into the editor to manipulate it, we will see the image above on the left. When we select any image from our computer and click open, the editor in Figure 1 will appear. If we want to save the image after performing operations on the uploaded image, we will see the image on the right by pressing the "Save Image" button in the image in Figure 1. We will be able to save the image to a section on our computer with the name we want by pressing save.

Also, our Image Editor has some functions that do not take any action on the image.

Reload Image is the function that helps us to load a new image for processing in our editor.

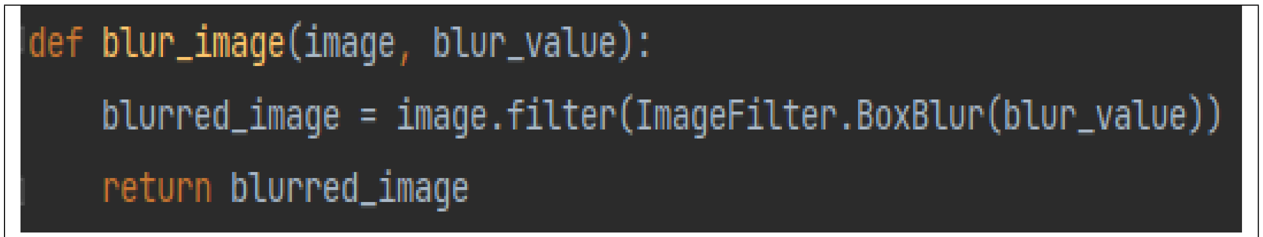
Reset To Original is the function that converts the image we are processing to its original state.

Show Original Image is the function that shows us the original state even if we process the uploaded image.

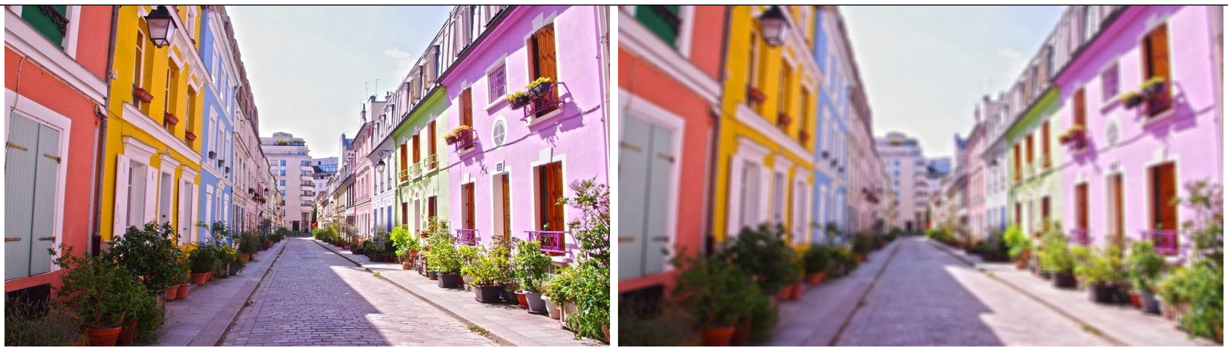
Show Processed Image is the function that shows us the image we are processing on.

Make Process Consecutive is the function that allows us to apply operations on the image in a cumulative way. If the button says Make Process Non-Consecutive, after flipping an image, we can convert the flipped image to grayscale. If the button says Make Process Consecutive, the processes we do will not progress cumulatively.

BLUR IMAGE

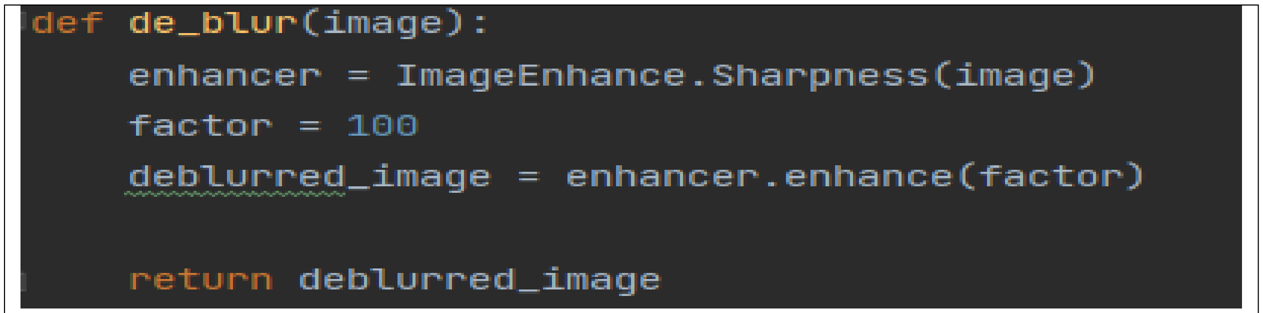


```
def blur_image(image, blur_value):
    blurred_image = image.filter(ImageFilter.BoxBlur(blur_value))
    return blurred_image
```



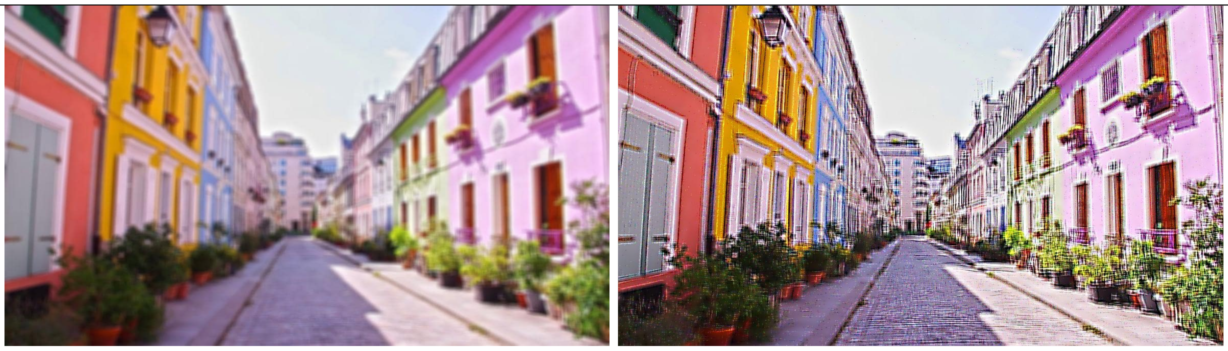
Blur image function takes a value from the user via the GUI and blurs the image according to that value. We used the PIL library to do the blur operation. In the above output, the value given for blur is 5. Blur values that can be entered start from 0. A value of 0 does not change the image.

DEBLUR IMAGE



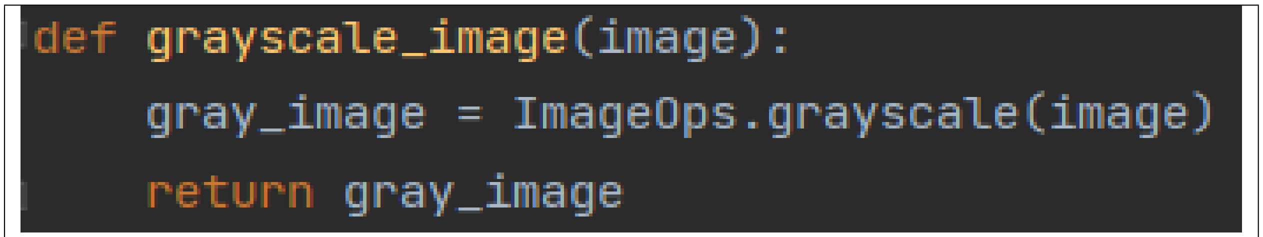
```
def de_blur(image):
    enhancer = ImageEnhance.Sharpness(image)
    factor = 100
    deblurred_image = enhancer.enhance(factor)

    return deblurred_image
```

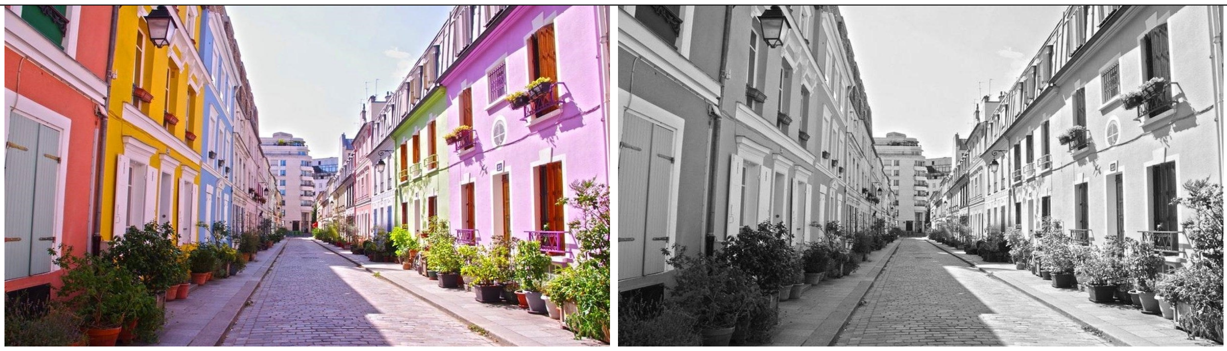


The deblur image function relatively removes the blur of the blurred image and sharpens the edges in the image. We used the ImageEnhance.Shapness function of the PIL library in the deblur or sharpening function. We set the sharpening factor to 100.

CONVERT IMAGE TO GRAYSCALE

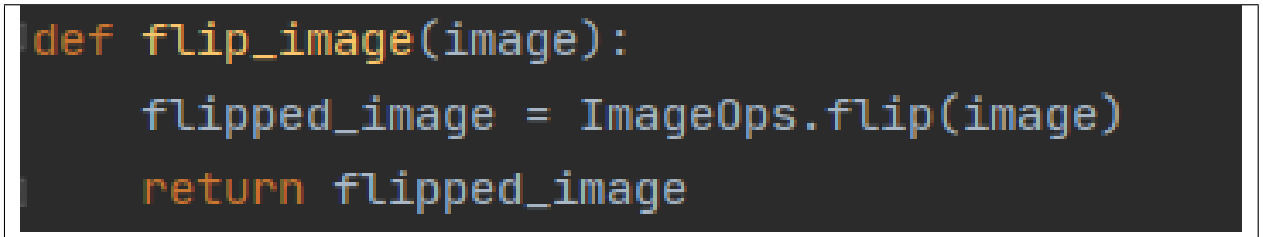


```python
def grayscale_image(image):
    gray_image = ImageOps.grayscale(image)
    return gray_image
```

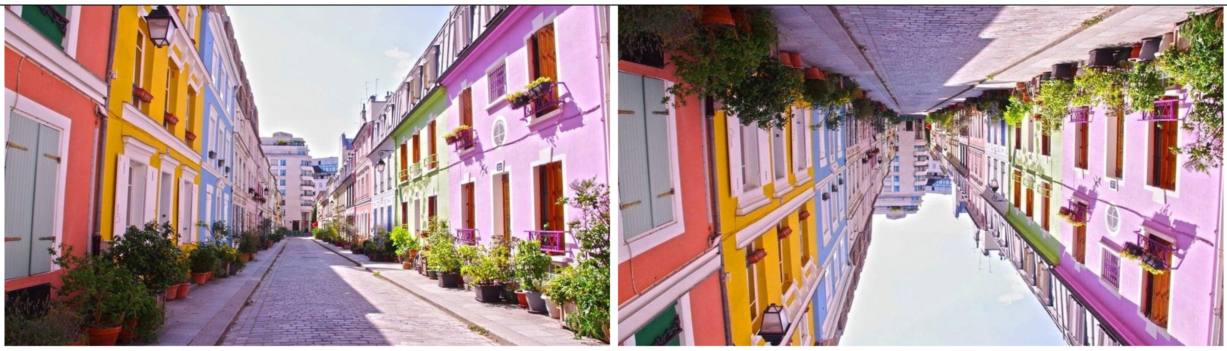


The grayscale image function converts the loaded image to grayscale. It does not take any other parameters. We used the PIL library for the grayscale conversion function.

FLIP IMAGE

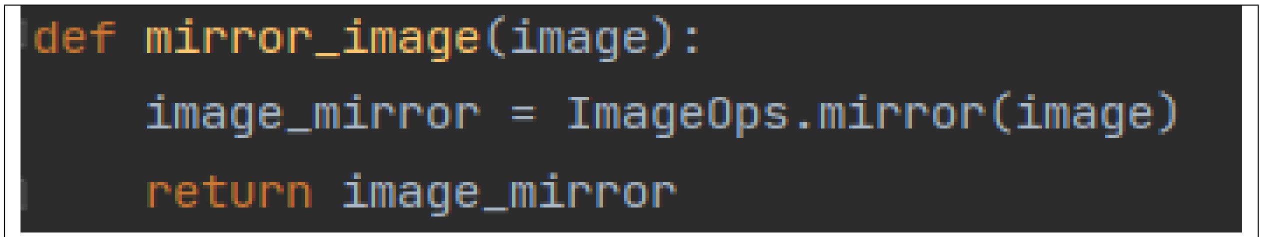


```python
def flip_image(image):
    flipped_image = ImageOps.flip(image)
    return flipped_image
```

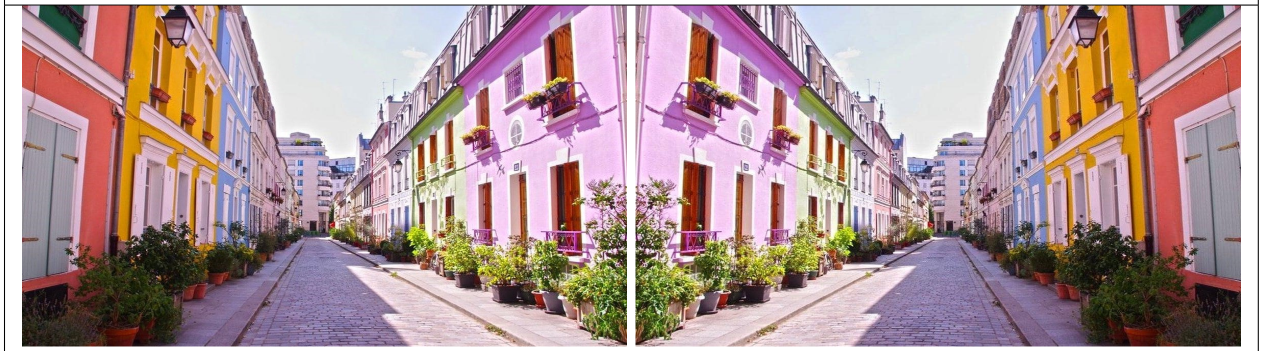


Flip image function rotates the uploaded image 180 degrees. We used the PIL library for the flip image function.

MIRROR IMAGE

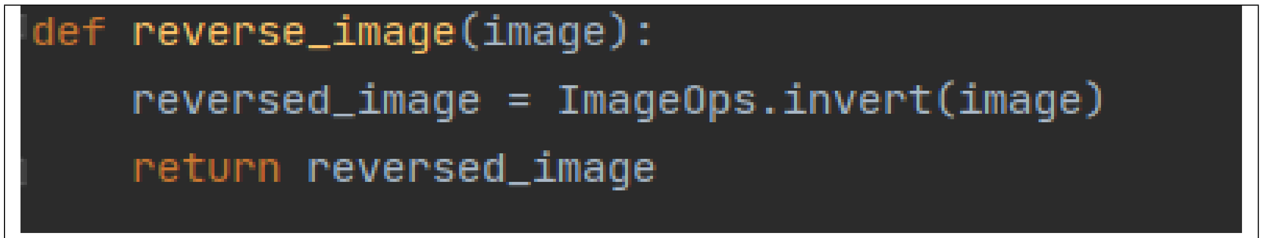


```
def mirror_image(image):
    image_mirror = ImageOps.mirror(image)
    return image_mirror
```

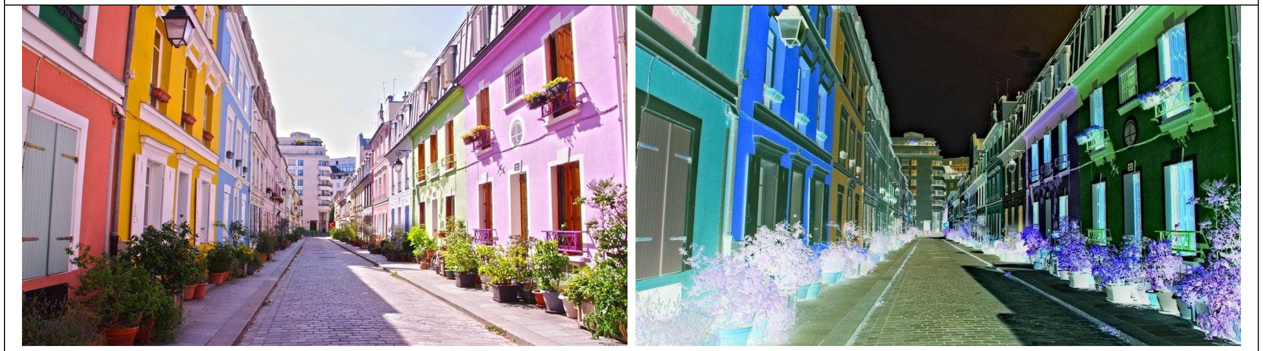


The Mirror image function replaces the uploaded image with a mirror image thanks to the ImageOps.mirror function of the PIL library.

REVERSE IMAGE

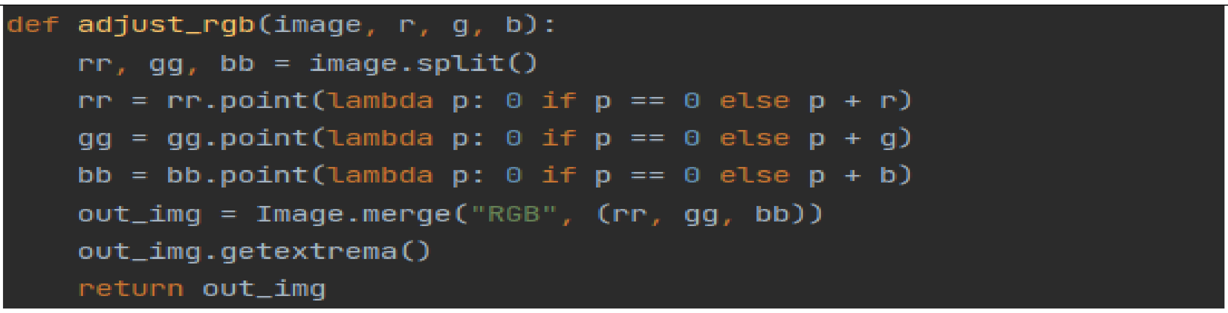


```
def reverse_image(image):
    reversed_image = ImageOps.invert(image)
    return reversed_image
```

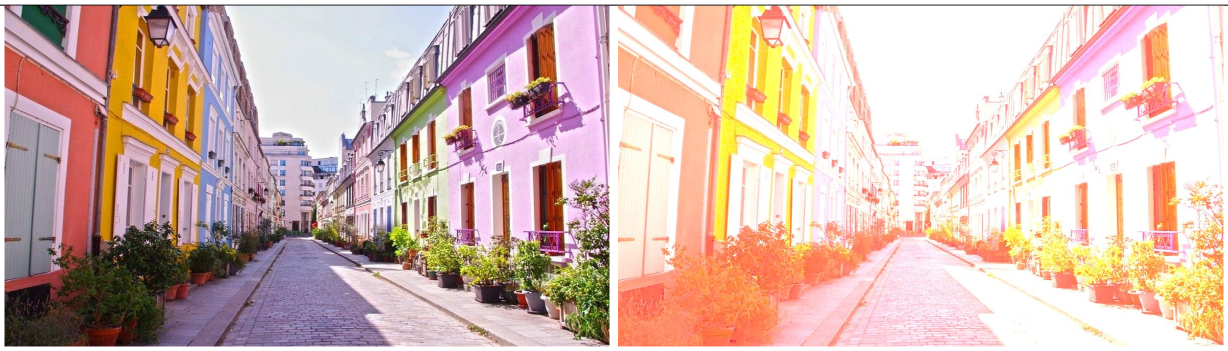


The reverse image function inverts the colors in the loaded image with the help of the ImageOps.invert function of the PIL library.

ADJUST RGB VALUES

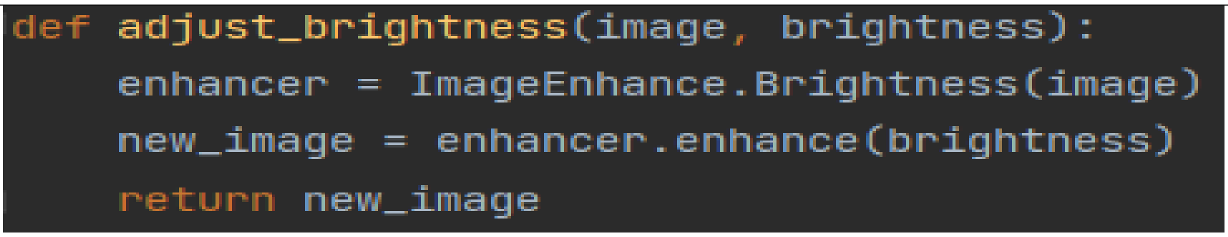


```python
def adjust_rgb(image, r, g, b):
    rr, gg, bb = image.split()
    rr = rr.point(lambda p: 0 if p == 0 else p + r)
    gg = gg.point(lambda p: 0 if p == 0 else p + g)
    bb = bb.point(lambda p: 0 if p == 0 else p + b)
    out_img = Image.merge("RGB", (rr, gg, bb))
    out_img.getextrema()
    return out_img
```

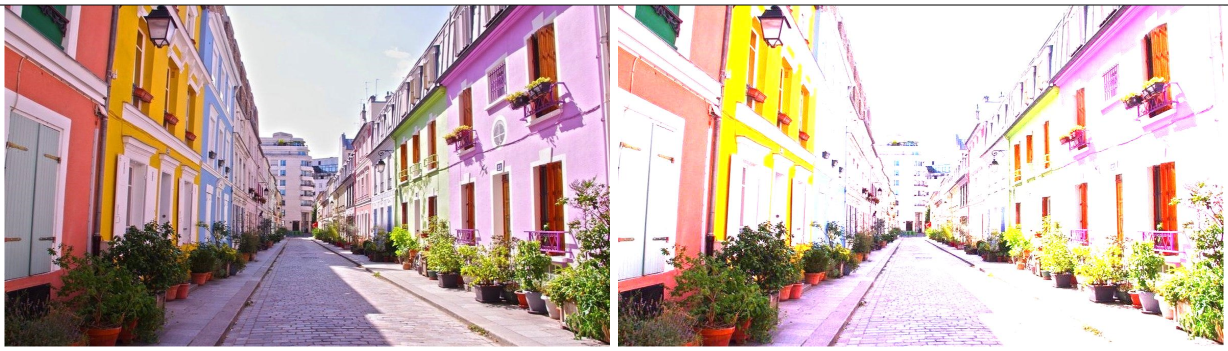


The Adjust RGB values function splits the uploaded image into R, G, B channels and replaces it with R, G, B values obtained from the user via GUI. It then merges the modified R, G, B channels and returns the new image. In the above output, the values given for the R, G, B channels are rgb(255, 99, 71).

ADJUST BRIGHTNESS

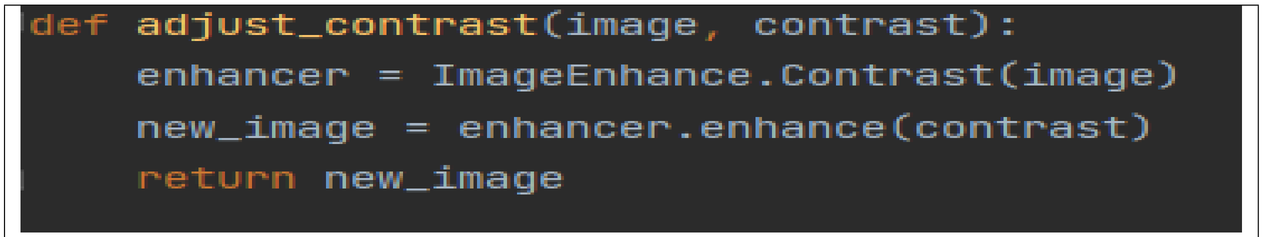


```python
def adjust_brightness(image, brightness):
    enhancer = ImageEnhance.Brightness(image)
    new_image = enhancer.enhance(brightness)
    return new_image
```

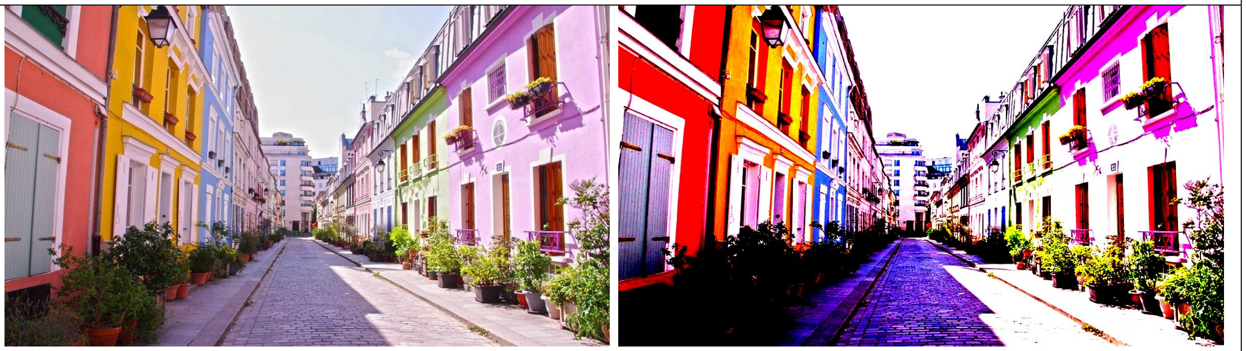


Adjust brightness function changes the brightness of the loaded image according to the value received from the user with the help of GUI. In the output above, the value given for brightness is 2. The values that can be given for Brightness start from 0. A value of 0 changes the image to a black image. A value of 1 does not make any changes. As values greater than 1 increase, the brightness of the image increases.

ADJUST CONTRAST

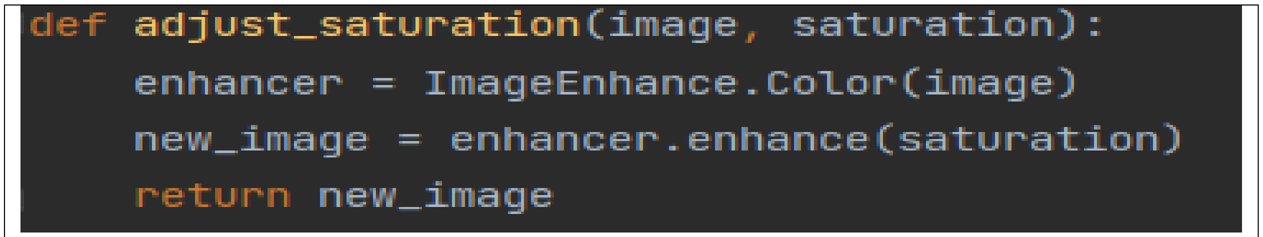


```
def adjust_contrast(image, contrast):
    enhancer = ImageEnhance.Contrast(image)
    new_image = enhancer.enhance(contrast)
    return new_image
```

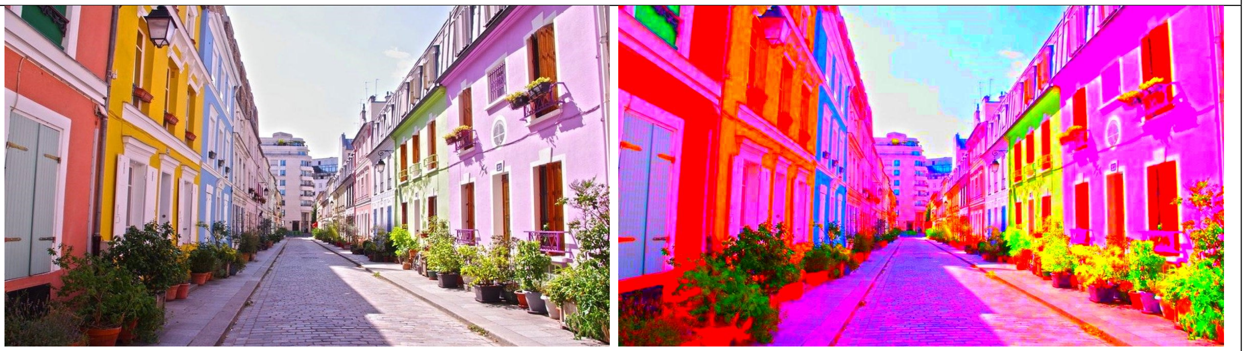


Adjust contrast function changes the contrast of the loaded image according to the value received from the user with the help of GUI. The values that can be entered for adjust contrast start from 1. The higher the value, the higher the contrast of the image. In the output above, the value given for contrast is 5.

ADJUST SATURATION

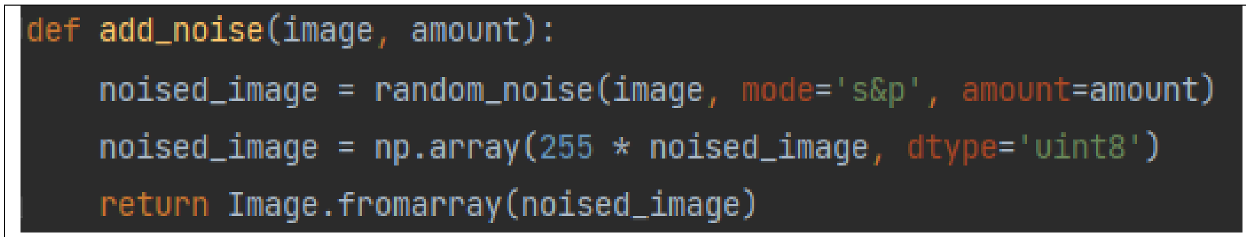


```
def adjust_saturation(image, saturation):
    enhancer = ImageEnhance.Color(image)
    new_image = enhancer.enhance(saturation)
    return new_image
```

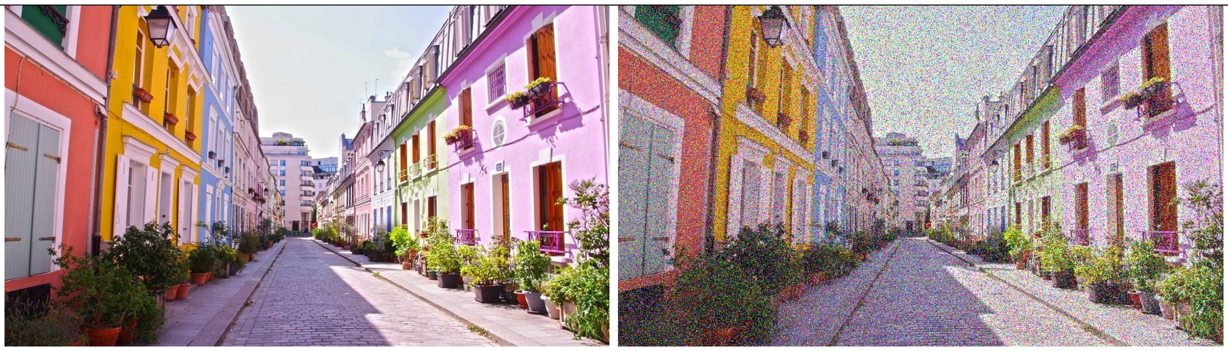


The Adjust saturation function changes the saturation of the loaded image according to the value received from the user via the GUI. The values that can be entered for Saturation start from 0. A value of 0 converts the image to grayscale. While a value of 1 does not make any changes, the higher the value, the greater the saturation of the colors in the image. In the output above, the saturation value is 10.

ADD NOISE

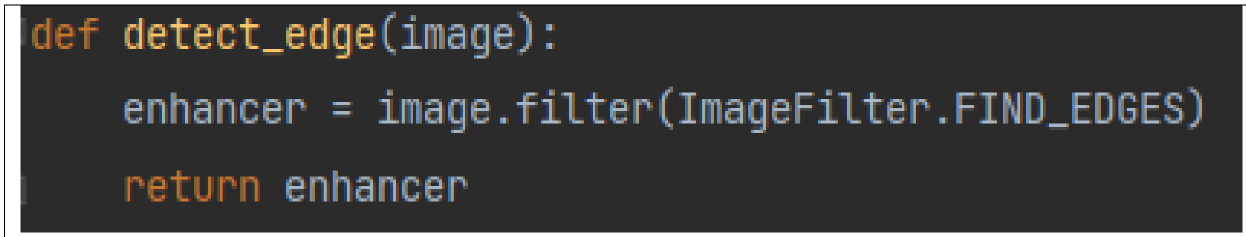


```
def add_noise(image, amount):
    noised_image = random_noise(image, mode='s&p', amount=amount)
    noised_image = np.array(255 * noised_image, dtype='uint8')
    return Image.fromarray(noised_image)
```

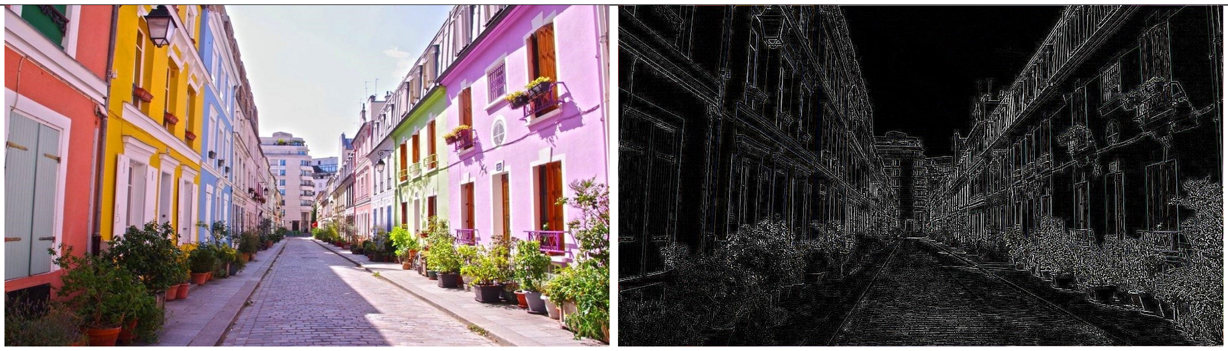


The add noise function adds noise to the loaded image based on the value received from the user. If the entered values are higher than 1, the image will only consist of noise. In order to obtain noise in the image, values between 0-1 must be entered. In the output above, the noise value entered is 0.25.

DETECT EDGES

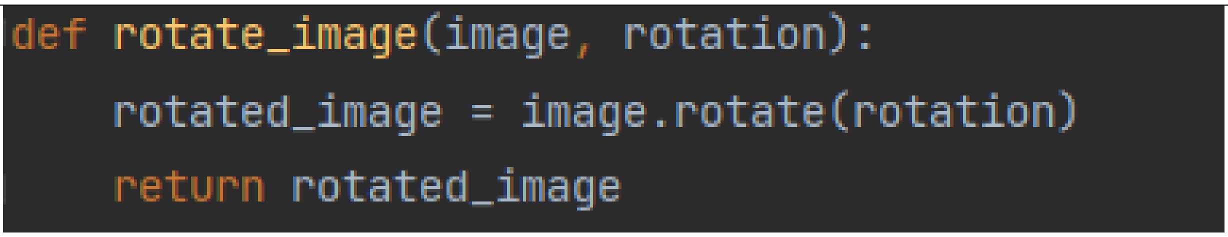


```
def detect_edge(image):
    enhancer = image.filter(ImageFilter.FIND_EDGES)
    return enhancer
```

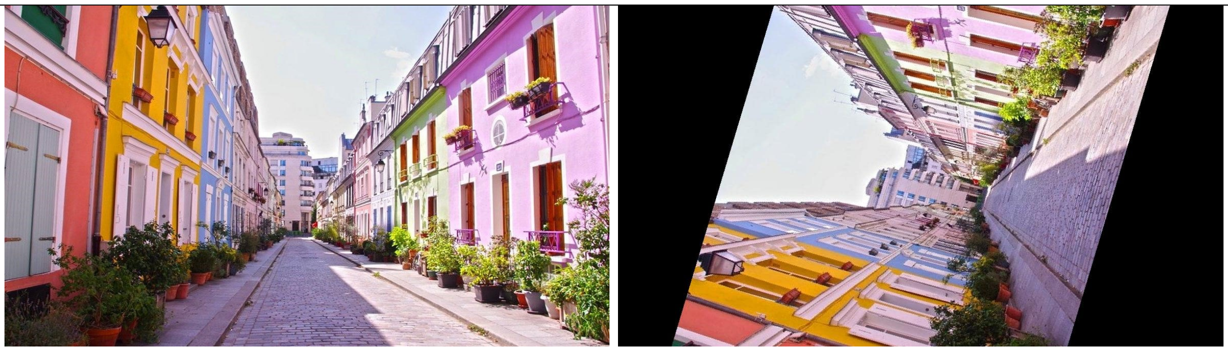


The detect edges function detects the edges in the loaded image. It does not take any parameters. We used the PIL library to detect edges.

ROTATE IMAGE

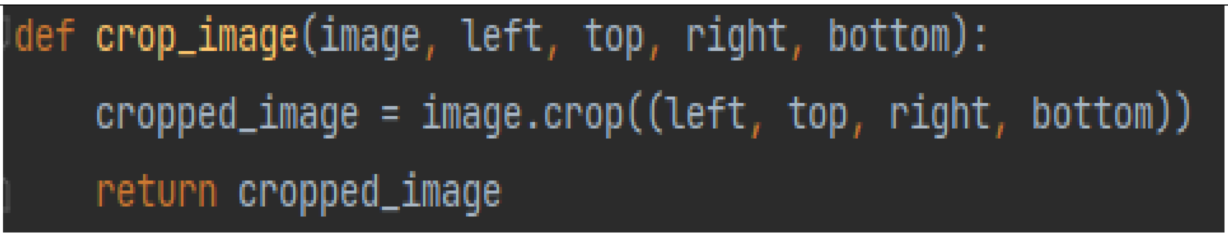


```
def rotate_image(image, rotation):
    rotated_image = image.rotate(rotation)
    return rotated_image
```

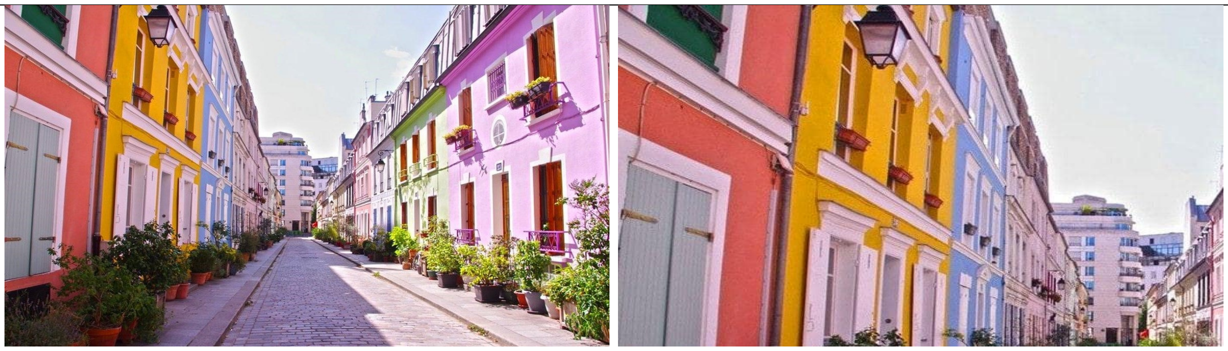


The rotate image function rotates the loaded image counterclockwise by the value received from the user via the GUI. The values that can be entered for Rotate are between 0-360. Values after 360 are again between 0-360. In the output above, the value entered for rotate is 75.

CROP IMAGE



```
def crop_image(image, left, top, right, bottom):
    cropped_image = image.crop((left, top, right, bottom))
    return cropped_image
```



The crop image function is implemented with 4 values retrieved from the user via the GUI. The user enters the left, top, right and bottom values of the part he wants to crop from the image. For example, if the image he wants to crop is 1100x800 pixels, if the user wants to crop from the upper left corner to 700 pixels to the right and 500 pixels to the bottom, he can crop the image by entering left = 0, top = 0, right = 700, bottom = 500. These values must be related to each other, otherwise the editor will throw an error. In the above output, the values left = 0, top = 0, right = 700, bottom = 500 have been applied to the loaded image. Image size is (1186x758).