Project 02: Image Processing

Name: Châu Tấn Kiệt

ID: 21127329

Class: 21CLC04

1. Task completion:

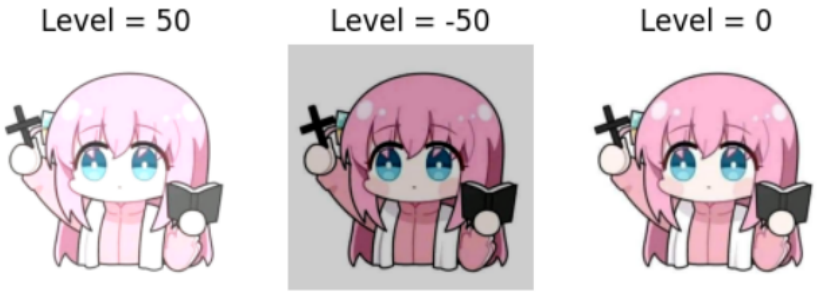
Task no:	Task name:	Status
1	Change brightness	Completed
2	Change contrast	Completed
3	Flip image	Completed
4	Convert RGB to grayscale / sepia	Completed
5	Sharpen / Blur image	Completed
6	Crop center	Completed
7	Crop circle	Completed
8	Main function	Completed
Bonus	Crop with 2 ellipses	Completed

2. Solution and function explanations:

Change brightness:

- Idea: Increase / decrease the intensity of each pixel by a constant.

```
def brightness(c: int) -> float:
        return 128 + level + (c - 128)
```



Change contrast:

- Step 1: Calculate a contrast correction factor which is given by the following formula:

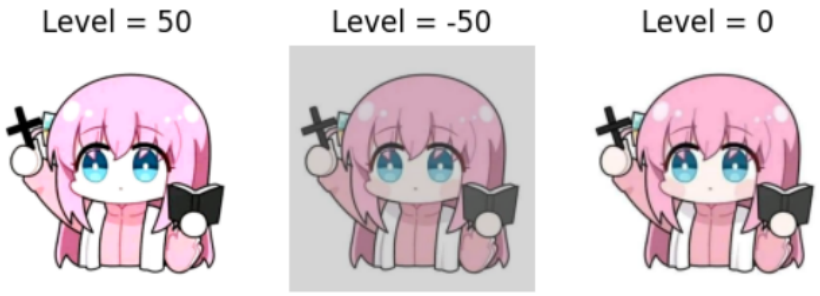
$$F = \frac{259(255 + C)}{255(259 - C}$$

```
factor = (259 * (level + 255)) / (255 * (259 - level))
```

- Step 2: Perform the contrast adjustment for every color component using the formula:

$$Component = F(Component - 128) + 128$$

```
def contrast(c: int) -> int:
        return int(128 + factor * (c - 128))
```



Flip image:

- Flip vertically: Reverse the matrix column-wise from up to down.

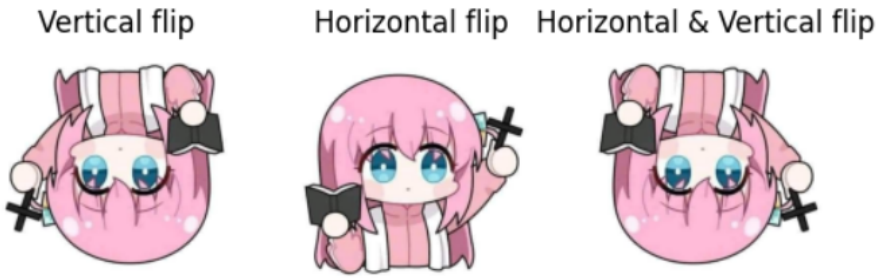
```
vertical_image = np.flipud(img)
```

- Flip horizontally: Reverse the matrix row-wise from left to right.

```
horizontal_image = np.fliplr(img)
```

- Flip both vertically and horizontally: Combine the two methods.

```
hori_ver_image = np.flip(img, (0, 1))
```



Convert RGB to grayscale / sepia:

- RGB to grayscale: Convert each color component into grayscale by using the formula:

$$Grayscale = 0.2989 * R + 0.5870 * G + 0.1140 * B)$$

```
img_out = np.dstack((0.2989 * R + 0.5870 * G + 0.1140 * B, \
                        0.2989 * R + 0.5870 * G + 0.1140 * B, \
                        0.2989 * R + 0.5870 * G + 0.1140 * B))
```

- RGB to sepia: Convert each color component into tR, tG, tB using the formula:

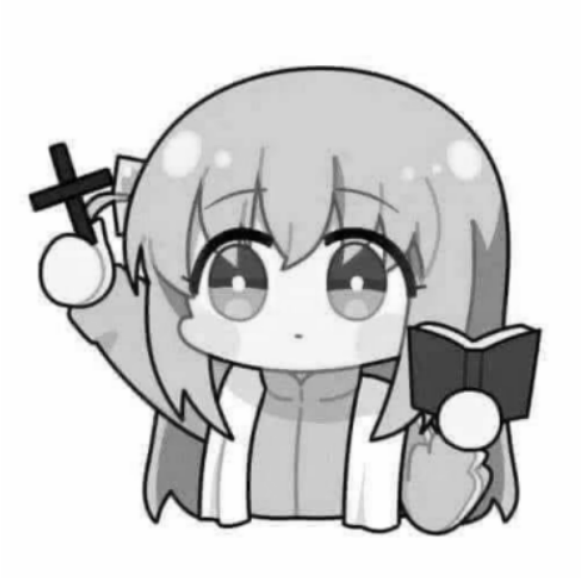
$$tR = 0.393 * R + 0.769 * G + 0.189 * B$$

$$tG = 0.349 * R + 0.686 * G + 0.168 * B$$

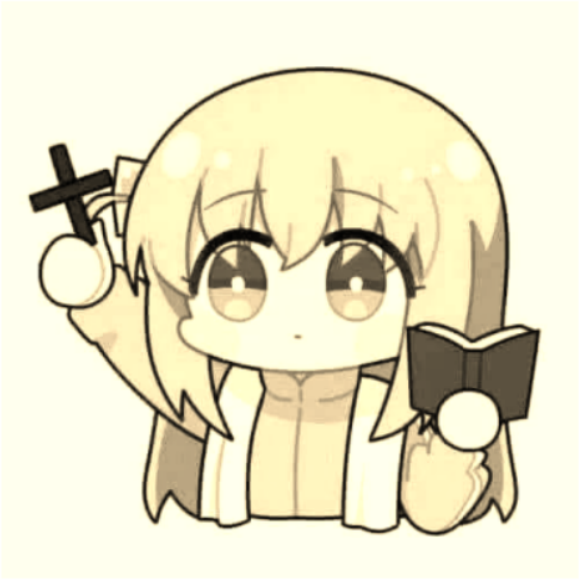
$$tB = 0.272 * R + 0.534 * G + 0.131 * B$$

```
img_out = np.dstack((0.393 * R + 0.769 * G + 0.189 * B, \
                     0.349 * R + 0.686 * G + 0.168 * B, \
                     0.272 * R + 0.534 * G + 0.131 * B))
```

RGB To Grayscale



RGB To Sepia



Sharpen / Blur image:

- Step 1: Get the list of kernels.

```
sharpen = np.array([[0, -1, 0],
                    [-1, 5, -1],
                    [0, -1, 0]])
blur = np.array([[0.0625, 0.125, 0.0625],
                 [0.125, 0.25, 0.125],
                 [0.0625, 0.125, 0.0625]])
box_blur = np.array([[0.1111, 0.1111, 0.1111],
                     [0.1111, 0.1111, 0.1111],
                     [0.1111, 0.1111, 0.1111]])
gaussian_blur = np.array([[0.0625, 0.125, 0.0625],
                          [0.125, 0.25, 0.125],
                          [0.0625, 0.125, 0.0625]])
```

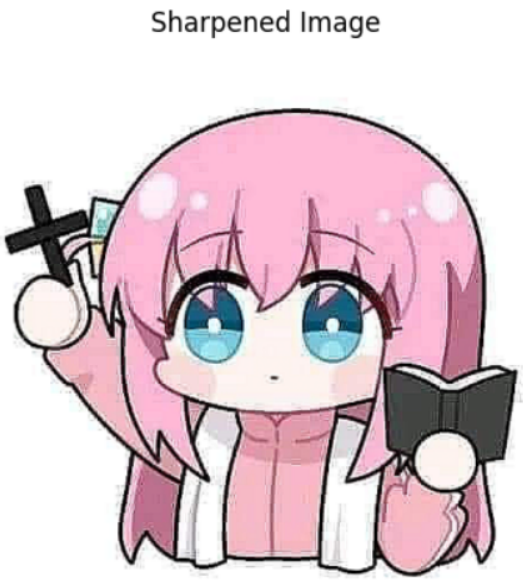
- Step 2: Adding each element of the image to its local neighbors, weighted by the kernels, also known as convolution process.

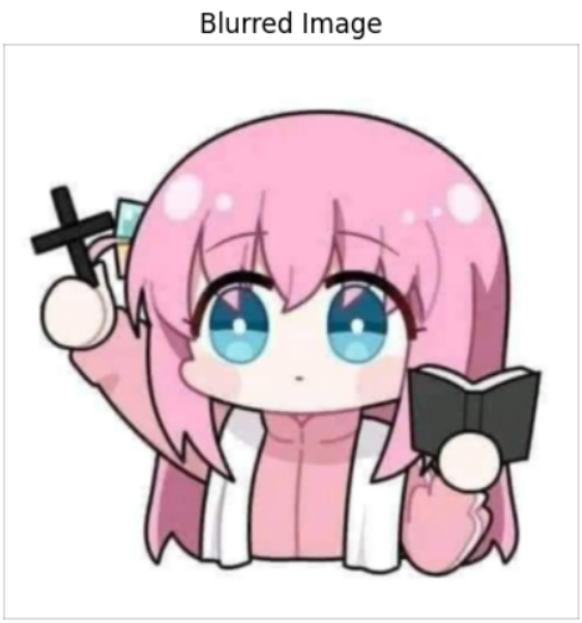
```
    for x in range(i_width):
        for y in range(i_height):
            weighted_pixel_sum = 0

            for kx in range(-(k_width // 2), k_width - 1):
                for ky in range(-(k_height // 2), k_height - 1):
                    pixel = 0
                    pixel_x = x - kx
                    pixel_y = y - ky
                    if (pixel_y >= 0) and (pixel_y < i_height) and (pixel_x >= 0) and (pixel_x < i_width):
                       pixel = image[pixel_x, pixel_y]
```

```
                    weight = kernel[ky + (k_height // 2), kx + (k_width // 2)]

                    weighted_pixel_sum += pixel * weight
            filtered[x, y] = weighted_pixel_sum / kernel_sum
```

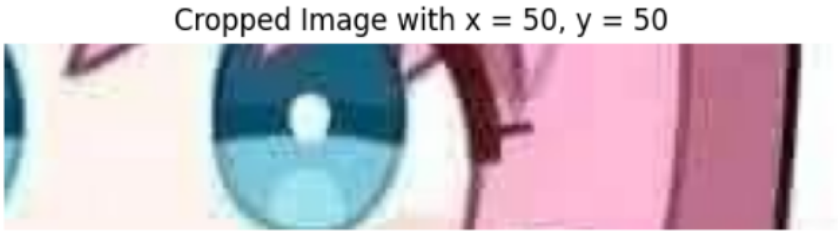




Crop center:

- Slice the image along the points defined as x and y.

```
startx = x // 2 - (cropx // 2)
starty = y // 2 - (cropy // 2)
crop_img = img[startx: -startx, starty:-cropy, :]
```



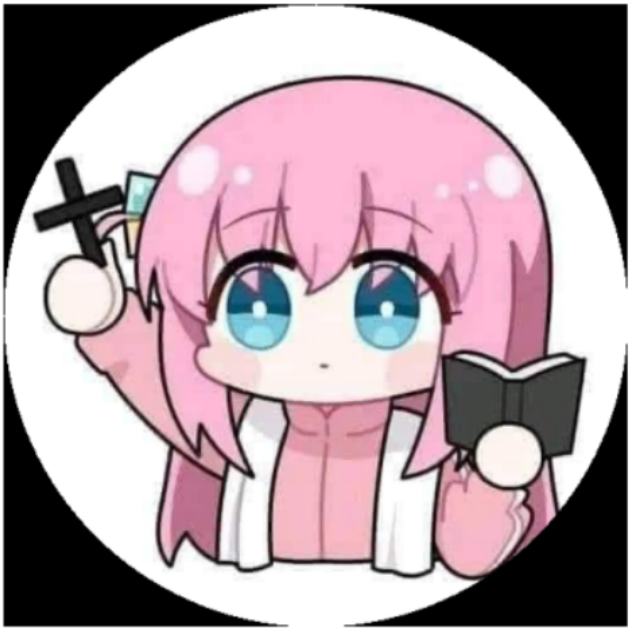
Crop circle:

- Step 1: Create the pixel coordinates) of the image, and then check each pixel coordinate to see if it's inside or outside the circle. In order to tell whether it's inside the center, we can simply find the Euclidean distance from the center to every pixel location, and then if that distance is less than the circle radius, we'll mark that as *included* in the mask, and if it's greater than that, we'll *exclude* it from the mask.
- Step 2: Check each pixel coordinate to see if it's inside or outside the circle by find the Euclidean distance from the center to every pixel location, if the distance is less than the circle radius then the pixel is included in the mask, and if it's greater than that, the pixel is excluded from the mask.

```
def create_circular_mask(h, w, center=None, radius=None):
    if center is None: # Use the middle of the image
        center = (int(w/2), int(h/2))
    if radius is None: # Use the smallest distance between the center and image walls
        radius = min(center[0], center[1], w-center[0], h-center[1])

    y, x = np.ogrid[:h, :w]
    dist_from_center = np.sqrt((x - center[0])**2 + (y-center[1])**2)

    mask = dist_from_center <= radius
    return mask
```



Main function: Allow users to enter the image processing selection.

1. Change brightness
2. Change contrast
3. Flip image
4. Convert RGB to grayscale/sepia
5. Sharpen / blur
6. Crop center/circle/ellipses
7. Execute all of the features

Bonus: Crop with 2 ellipses:

The implementation is the same as cropping into a circle, but there will be two masks for two ellipses.

The formula for each of the ellipses area is shown as:

$$(a^2+b^2)((x-x0)^2+(y-y0)^2)+2(b^2-a^2)(x-x0)(y-y0)\geq 2(ab)^2$$

$$(a^2+b^2)((x-x0)^2+(y-y0)^2)+2(a^2-b^2)(x-x0)(y-y0)\geq 2(ab)^2$$

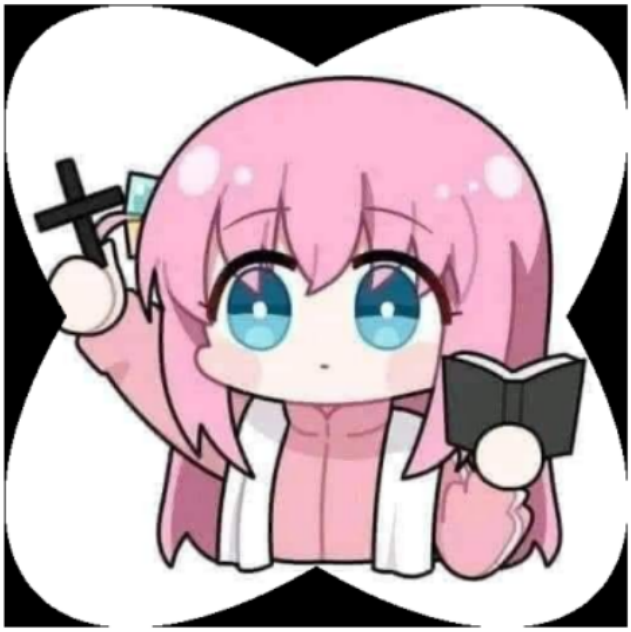
To make the ellipses tangent to the edges of the square, the values of a and b to s (s is the length of the edge of the square) is shown as:

$$s = \sqrt{2}\sqrt{a^2 + b^2} \rightarrow \frac{s^2}{2} = a^2 + b^2$$

Which means the values a and b is in the range of:

$$\begin{cases} a = \frac{s}{\sqrt{2}} cos(\alpha) \\ b = \frac{s}{\sqrt{2}} sin(\alpha) \end{cases}$$

```
def create_ellipse_mask(h, w, center=None, radius=None):
    if center is None: # Use the middle of the image
        center = (int(w/2), int(h/2))
    a = h/np.sqrt(2) * np.cos(90)
    b = h/np.sqrt(2) * np.sin(90)
    y, x = np.ogrid[:h, :w]
    dist_from_center1 = np.sqrt((a**2 + b**2)*((x - center[0])**2 + (y-center[1])**2) + 2*((b**2 - a**2))*((x - center[0]))*(y-center[1]))
    dist_from_center2 = np.sqrt((a**2 + b**2)*((x - center[0])**2 + (y-center[0])**2) + 2*((a**2 - b**2))*((x - center[0]))*(y-center[0]))
    mask1 = dist_from_center1 < np.sqrt(2*((a*b)**2))
    mask2 = dist_from_center2 < np.sqrt(2*((a*b)**2))
    return mask1 + mask2
```



3. References:

Kernel (image processing) - Wikipedia

calculus - how to calculate the bounding square of an ellipse? - Mathematics Stack Exchange

Algorithms for Adjusting Brightness and Contrast of an Image – The IE Blog (nitk.ac.in)

RGB to Grayscale Conversion Calculator (had2know.org)

Image Processing Algorithms Part 5: Contrast Adjustment | Dreamland Fantasy Studios (dfstudios.co.uk)