Applied mathematics *University of Science*

FACULTY OF INFORMATION TECHNOLOGY

REPORT PROJECT 02
IMAGE PROCESSING

Lecturer: Nguyễn Văn Quang Huy
Phan Thị Phương Uyên
Vũ Quốc Hoàng
Lê Thanh Tùng

Student's Information:

Student's name: Nguyễn Thoại Đăng Khoa

Student's ID: 20127043

JULY 1ST, 2022
VNUHCM – UNIVERSITY OF SCIENCE

TABLE OF CONTENTS

I. INTRODUCTION

Image processing is a technique to play out a certain procedure on a picture, to get an upgraded picture, or to extricate some helpful data from it. These days, image processing is among quickly developing advancements. It plays an important role in designing and software engineering disciplines as well.

Image processing basically includes the following **three** steps: **Importing** the image via image acquisition tools; **Analysing** and manipulating the image; **Output** in which result can be altered image or report that is based on image analysis.

In this project, we will learn how to process images through some operations such as: changing the brightness/contrast of an image, flipping an image (vertically / horizontally), converting an RGB image to grayscale, blending overlapping images, and blurring an image.

II. COMPLETION PROGRESS

NO.	FUNCTION	DETAIL	COMPLETE
1	Change the brightness	Increase the intensity of each pixel by a constant.	100%
2	Change the contrast	Change the contrast of the image	100%
3	Flip an image	Flip an image vertically / horizontally	100%
4	Convert RGB image to grayscale	Convert to grayscale based on weighted or average method	100%
5	Blend overlapping images	Mixing two images of the corresponding pixel values to create a new target image.	100%
6	Blur an image	Blur an image by using box blur or gaussian 3x3 kernel	100%
7	Crop an image into Circle	Crop the image into circle frame	100%
8	Crop an image into Ellipse	Crop the image into two intersect ellipses frame	100%
TOTAL COMPLETION			100%

III. PROCESSING DETAILS

3.1. Pre-processing

In this project, only the following libraries are allowed: PIL, numpy, matplotlib.

Import Library

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
```

Then, we need to open image and convert to numpy arrays.

Open an image and convert to numpy arrays

```
def openImg():
  # Input
  img = input('Enter name of an image: ')
  # Open img
  image = Image.open(img).convert("RGB")
  # Convert to numpy arrays (3d matrix)
  image = np.array(image)
  return image, img
```

There are 3 main steps:

- Enter name of an image.
- Open that image and convert to RGB (use Image in PIL library).
- Convert that image to 3D matrix (numpy array) by using np.array() function.

3.2. Change the brightness

IDEA

- Increase the intensity of each pixel by a constant.
- Keep the color value in the valid range from 0 to 255.
- If color value < 0 then value = 0.
- If color value > 255 then value = 255.
- Negative values will darken the image and, conversely, positive values will brighten the image.

IMPLEMENT

Change the brightness

```
def changeTheBrightness(image, bright = 50):
  # make a hard copy
  outputImage = image.copy()
  # shape decay
  rows = outputImage.shape[0]
  cols = outputImage.shape[1]
  channels = outputImage.shape[2]
  # adjust the brightness
  for y in range(rows):
    for x in range(cols):
        for c in range(channels):
          if (outputImage[y,x,c] + bright > 255):
            outputImage[y,x,c] = 255
          elif (outputImage[y,x,c] + bright < 0):
            outputImage[y,x,c] = 0
          else:
            outputImage[y,x,c] += bright
  return outputImage
```

- **Input:** image (converted to 3D matrix), **bright** parameter is ratio of brightness of user input (default = 50).
- Make a hard copy by copy from original image array to new output image array (avoid shallow copy).
- Decay the image shape into rows, cols, and channels.
- Use for loop to access each element in image.
- Add each pixel's color with ratio of brightness value.
- If value of that pixel's color > 255 then value = 255.
- Else if value of that pixel's color < 0 then value = 0.
- Else the value += ratio of brightness.
- **Output:** processed output image array.

3.3. Change the contrast

IDEA

- Change the value of the max and min intensity pixel.
- Use a contrast correction factor (F) given by the following formula:

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

(C: contrast ratio)

- Perform the actual contrast adjustment itself (R')

$$R' = F(R - 128) + 128$$

The above formula shows the adjustment in contrast being made to the Red component of a colour (Green, Blue is similar too)

- Keep the value in the valid range from 0 to 255.
- If color value < 0 then value = 0.
- If color value > 255 then value = 255.
- Negative values will decrease the amount of contrast and, conversely, positive values will increase the amount of contrast.
- All these formular, I refer to this link (1): **https://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-5-contrast-adjustment/**

IMPLEMENT

Change the contrast

```
def changeTheContrast(image, contrast = 50):
  # make a hard copy
  outputImage = image.copy()
  # shape decay
  rows = outputImage.shape[0]
  cols = outputImage.shape[1]
  channels = outputImage.shape[2]
  # adjust the contrast
  factor = (259 * (contrast + 255)) / (255 * (259 - contrast))
  actual_contrast = 0
  for y in range(rows):
    for x in range(cols):
        for c in range(channels):
          actual_contrast = factor*(outputImage[y,x,c] - 128) + 128
          if (actual_contrast > 255):
            outputImage[y,x,c] = 255
          elif (actual_contrast < 0):
            outputImage[y,x,c] = 0
          else:
            outputImage[y,x,c] = actual_contrast
  return outputImage
```

- **Input:** image (converted to 3D matrix), **contrast** parameter is ratio of contrast of user input (default = 50).
- Make a hard copy by copy from original image array to new output image array (avoid shallow copy).
- Decay the image shape into rows, cols, and channels.
- Calculate a contrast correction factor (factor).
- Use for loop to access each element in image.
- Change each pixel's color with contrast value by using above formula. (The result is stored in actual_contrast variable)
- If contrast value > 255 then contrast value = 255.
- Else if contrast value < 0 then contrast value = 0.
- Else the pixel's color = contrast value.
- **Output:** processed output image array.

3.4. Flip an image (vertically / horizontally)

IDEA

- To flip vertically: reverse rows in array.
- To flip horizontally: reverse pixels in rows.
- Convert array to list.
- Use method reversed() of list to handle.
- Reconvert to numpy array.

IMPLEMENT

Flipping Image

```
def flippingImage(image, flipping_type = "vertically"):
  # make a hard copy
  outputImage = image.copy()
  # flipping image
  if (flipping_type == "vertically"): outputImage = list(reversed(outputImage)) # flip vertically
  else:
    outputImage = [list(reversed(row)) for row in outputImage] # flip horizontally
  return np.array(outputImage)
```

- **Input:** image (converted to 3D matrix), **flipping_type** parameter is type of flipping of user input (default type is vertically).
- Make a hard copy by copy from original image array to new output image array (avoid shallow copy).
- Convert array to list and use reversed() to create list in reverse order.
- If **flipping_type** = vertically, reverse rows in array (convert to list and reverse).
- Else if **flipping_type** = horizontally, reverse pixels in rows (using for loop to access each row in output image array and reverse pixels in row).
- Reconvert to numpy array.
- **Output:** processed output image array.

3.5. Convert an RGB image to a grayscale

IDEA

- There are 2 methods to convert it. These methods are: Average method, Weighted method (luminosity method)

Average method

- Take the average of 3 colors.
- The grayscale can be calculated as:

$$\text{Grayscale} = (R + G + B) / 3$$

- Problem: We are taking 33% of each, that means, each of the portion has same contribution in the image, but this turned out to be a rather black image (Because different colors have different wavelength)

Weighted method

- Decrease the contribution of red color, and increase the contribution of the green color, and put blue color contribution in between these two.
- The grayscale can be calculated as:

$$\text{Grayscale} = (0.3 * R) + (0.59 * G) + (0.11 * B)$$

- Red has contribute 30%, Green has contributed 59% which is greater in all three colors and Blue has contributed 11%.
- As compare to the result of average method, this image is more brighter.

- All these formulas, I referred to link (2): **https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.htm**

IMPLEMENT

```
def weightedGrayscaleMethod(colors):
    return (0.299*colors[0] + 0.587*colors[1] + 0.114*colors[2])

def avgGrayscaleMethod(colors):
    return (colors[0] + colors[1] + colors[2])/3
```

- Implement 2 functions as 2 above methods.
- 2 methods are applied with those formula and return the new color of that pixel (colors).

```
def convertToGrayscale(image, method = "weighted"):
  # make a hard copy
  outputImage = image.copy()
  # shape decay
  rows = outputImage.shape[0]
  cols = outputImage.shape[1]
  # convert to grayscale
  for i in range(rows):
    for j in range(cols):
        if (method == "weighted"):
            outputImage[i][j] = weightedGrayscaleMethod(outputImage[i][j])
        else:
            outputImage[i][j] = avgGrayscaleMethod(outputImage[i][j])
  return outputImage
```

- **Input:** image (converted to 3D matrix), **method** parameter is method that user wants to use (default is weighted).
- Make a hard copy by copy from original image array to new output image array (avoid shallow copy).
- Decay the image shape into rows, cols, and channels.
- Calculate a contrast correction factor (factor).
- Use for loop to access each element in image.
- Change each pixel's color with value calculated from 2 above methods.
- **Output:** processed output image array.

3.6. Blend overlapping images

IDEA

- Mixing two images of the corresponding pixel values to create a new target image.
- Add each color of pixel of the first image to each color of pixel of the second image with below formula:

$$g(x) = (1 - \alpha) f_0(x) + \alpha f_1(x)$$

- Keep the value in the valid range from 0 to 255.

IMPLEMENT

Blending Image

```
def blendingImage(image_1, image_2, alpha = 0.5):
  # initialize outputImage with 0
  outputImage = np.zeros(image_1.shape)
  # shape decay
  rows = image_1.shape[0]
  cols = image_1.shape[1]
  for i in range(rows):
    for j in range(cols):
      outputImage[i][j] = image_1[i][j] * (1.0 - alpha) + image_2[i][j] * alpha
  return outputImage
```

- **Input:** image 1 and image 2 (converted to 3D matrix)
- Initialize the outputImage array with the same shape as the original image and initialize it to all 0.
- Decay the image shape into rows, cols.
- Use for loop to access each element in image.
- Color of pixel of image 1 * (1.0 – alpha) + color of pixel of image 2 * alpha = color of pixel of result image (I choose fixed alpha = 0.5).
- This above formula, I refer to these links (3), (4):
  1. **https://docs.opencv.org/3.4/d5/dc4/tutorial_adding_images.html**
  2. **https://github.com/t3bol90/ST-MA-Lab03**
- **Output:** processed output image array.

3.7. Blur an image

IDEA

- Use convolution technique: change the value of a pixel according to the values of its surrounding pixels (The new value of each pixel is a weighted average of the neighbouring pixel's original values).
- Then, use that technique with a kernel (box blur and Gaussian blur 3x3)
- **Kernel of Box blur** **Kernel of Gaussian blur 3x3**

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

- A kernel matrix is created and then this kernel will be overlayed onto a block of pixels in the image, and each of value is multiplied with the corresponding pixel value. Next, we sum all of these to find the new value of the block's center pixel.
- The kernel then shifts by one pixel and the operation is repeated.
- It can be summarized by this pseudocode:
  (Refer to **https://en.wikipedia.org/wiki/Kernel_(image_processing)** (5))

```
for each image row in input image:
    for each pixel in image row:

        set accumulator to zero

        for each kernel row in kernel:
            for each element in kernel row:

                if element position  corresponding* to pixel position then
                    multiply element value  corresponding* to pixel value
                    add result to accumulator
                endif

         set output image pixel to accumulator
```

IMPLEMENT

```
def blurringImage(image, kernel_type = "box_blur"):
  # make a hard copy
  outputImage = image.copy()
  rows = outputImage.shape[0]
  cols = outputImage.shape[1]
  # create kernel
  if (kernel_type == "gauss_3x3"): kernel = gaussian_kernel
  else: kernel = box_kernel
  # Middle of the kernel
  mid_kernel = len(kernel) // 2
  kernel_size = len(kernel)
  # Processing
  for i in range(mid_kernel, rows - mid_kernel):
      for j in range(mid_kernel, cols - mid_kernel):
          accumulator = np.array([0,0,0])
          for a in range(kernel_size):
              for b in range(kernel_size):
                  accumulator = accumulator + kernel[a][b]*image[i - mid_kernel + a][j - mid_kernel + b]
          outputImage[i][j] = accumulator
  return outputImage
```

- **Input:** image (converted to 3D matrix), **kernel_type** is the kernel that user wants to use.
- Make a hard copy by copy from original image array to new output image array (avoid shallow copy).
- Decay the image shape into rows, cols.
- Create a kernel base on the type of user input.

Implement explanation:

- For example: we start from mid_kernel which is calculated by len(kernel) / 2 and rounded down ➔ It will start at red point I colored. Assume that we have a box blur kernel, and it was overlayed onto that block of pixels in the image ➔ The blue pixel is the new value of that pixel by using weighted average of the neighbouring pixel's original values (calculate sum of color arround it and divide by 9)

Original matrix

0	0	0	0	0	0	0
0	1	1	1	1	0	0
0	1	2	2	1	1	0
0	1	2	2	2	1	0
0	0	1	2	2	1	0
0	0	1	1	1	1	0
0	0	1	1	1	1	0

➔

Box blur kernel

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

➔

Output matrix

0	0	0	0	0	0	0
0	5/9	1	1	1	0	0
0	1	2	2	1	1	0
0	1	2	2	2	1	0
0	0	1	2	2	1	0
0	0	1	1	1	1	0
0	0	1	1	1	1	0

- Then, the kernel then shifts by one pixel and the operation is repeated.
- Use for loop to access the pixel we want to change in image.
- Variable i starts from mid_kernel to rows – mid_kernel (red axis)
- Variable j starts from mid_kernel to cols – mid_kernel (blue axis)

Original matrix

0	**0**	**0**	**0**	**0**	**0**	**0**
0	**1**	**1**	**1**	**1**	**0**	**0**
0	**1**	**2**	**2**	**1**	**1**	**0**
0	**1**	**2**	**2**	**2**	**1**	**0**
0	**0**	**1**	**2**	**2**	**1**	**0**
0	**0**	**1**	**1**	**1**	**1**	**0**
0	**0**	**1**	**1**	**1**	**1**	**0**

Implement the convolution technique

- Create an accumulator color channel array (fills with 0)
- Multiply element value (kernel) corresponding to pixel value and add result to accumulator array. This can be done by using this formula:

```
accumulator = accumulator + kernel[a][b]*image[i - mid_kernel + a][j - mid_kernel + b]
```

(i: index of rows of image, j: index of cols of image)

(a: index of rows of kernel; b: index of cols of kernel)

- By using this formula, when the kernel shifts by one pixel, we can access the elements in that block correctly.
- After that, we have an accumulator color channel array fills with new value for the center pixel.
- This above formula, I refer to this link (6): **https://www.youtube.com/watch?v=BPBTmXKtFRQ&t=94s**

Original matrix

0	0	0	0	0	0	0
0	1	1	1	1	0	0
0	1	2	2	1	1	0
0	1	2	2	2	1	0
0	0	1	2	2	1	0
0	0	1	1	1	1	0
0	0	1	1	1	1	0

➔

Output matrix

0	0	0	0	0	0	0
0						0
0						0
0						0
0						0
0						0
0	0	1	1	1	1	0

- **Finally,** the output matrix will replace the value of pixel that I colored above, and the output image will be blurry.
- **Output:** processed output image array.

Note: The convolution of a Gaussian kernel is identical to the Box Blur kernel. With Box Blur as dividing the sum of the kernel values, which totals 9. In the Gaussian kernel, the sum of the kernel values is 16. However, Gaussian blur mixes the values of the neighboring pixels depending on the distance to the reference pixel. The pixels right next to the reference pixel get more of the colors from the reference pixel while the pixels farther away get less of the colors.

3.8. Crop an image into Circle

IDEA

- **Processing only on square images.**
- Use the equation of the circle (calculate distance) to determine the relative position of each point.
- Blackout those points outside the circle.
- It can be summarized by this formula:

Outside: $x_1^2 + y_1^2 - r^2 > 0$

On: $x_1^2 + y_1^2 - r^2 = 0$

Inside: $x_1^2 + y_1^2 - r^2 < 0$

IMPLEMENT

Crop image into Circle

```
def cropImageIntoCircle(image):
  # make a hard copy
  outputImage = image.copy()
  # shape decay
  rows = outputImage.shape[0]
  cols = outputImage.shape[1]
  channels = outputImage.shape[2]
  # radius
  a,b = rows / 2, cols / 2
  # processing
  for i in range(rows):
    for j in range(cols):
      if (i - a) **2 + (j - b)**2 > (rows*cols)/4:
        for k in range(channels):
          outputImage[i,j,k] = 0
  return outputImage
```

- **Input:** image (converted to 3D matrix)
- Make a hard copy by copy from original image array to new output image array (avoid shallow copy).
- Decay the image shape into rows, cols.
- Calculate the radius coordinate (a, b) by using rows / 2 and cols / 2
- Use for loop to access each coordinate of point (i,j) and compare it to R (R = rows / 2 or cols / 2, ➔ R^2 = $rows^2$ / 4 = rows * cols / 4)
- If $(i - a)^2 + (j - b)^2 > R^2$, then **blackout** this point (i,j) by setting color of that point to 0 ([0,0,0]).
- **Output**: processed output image array.

3.9. Crop an image into Ellipse

IDEA

- **Processing only on square images.**
- Use the equation of the ellipse (calculate distance) to determine the relative position of each point.

$$\left(\frac{\cos^2\alpha}{a^2}+\frac{\sin^2\alpha}{b^2}\right)x^2+2\cos\alpha\sin\alpha\left(\frac{1}{a^2}-\frac{1}{b^2}\right)xy+\left(\frac{\sin^2\alpha}{a^2}+\frac{\cos^2\alpha}{b^2}\right)y^2=1$$

- Ellipse 1 can be drawn by using above formula with $\alpha = \frac{\pi}{4}$
- Ellipse 2 can be drawn by using above formula with $\alpha = \frac{-\pi}{4}$

For example, I use *desmos.com* to visualize ($\alpha_1 = \frac{\pi}{4}$, $\alpha_2 = \frac{-\pi}{4}$, a = 5, b = 3):

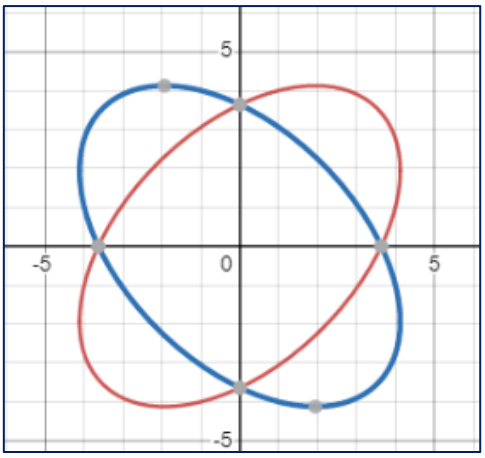
Equation of ellipse 1 (red)

$$\left(\frac{\left(\cos\left(\frac{\pi}{4}\right)\right)^2}{5^2}+\frac{\left(\sin\left(\frac{\pi}{4}\right)\right)^2}{3^2}\right)x^2+2\cos\left(\frac{\pi}{4}\right)\sin\left(\frac{\pi}{4}\right)\cdot\left(\frac{1}{5^2}-\frac{1}{3^2}\right)xy+\left(\frac{\left(\sin\left(\frac{\pi}{4}\right)\right)^2}{5^2}+\frac{\left(\cos\left(\frac{\pi}{4}\right)\right)^2}{3^2}\right)y^2=1$$

Equation of ellipse 2 (blue)

$$\left(\frac{\left(\cos\left(\frac{-\pi}{4}\right)\right)^2}{5^2}+\frac{\left(\sin\left(\frac{-\pi}{4}\right)\right)^2}{3^2}\right)x^2+2\cos\left(\frac{-\pi}{4}\right)\sin\left(\frac{-\pi}{4}\right)\cdot\left(\frac{1}{5^2}-\frac{1}{3^2}\right)xy+\left(\frac{\left(\sin\left(\frac{-\pi}{4}\right)\right)^2}{5^2}+\frac{\left(\cos\left(\frac{-\pi}{4}\right)\right)^2}{3^2}\right)y^2=1$$

Result:



- But since the formula only applies to the center of the coordinate axis, we need to subtract (x, y) with (h, k) which is the position of eclipse center (center of 2 ellipses must be in the center of the square).
- Blackout those points outside the ellipse.

- It can be summarized by this formula:

If the point (x_1, y_1) lies outside, on or inside the ellipse $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$ then $\frac{x_1^2}{a^2} + \frac{y_1^2}{b^2} - 1 >, =$ or < 0.

- The general equation of an Ellipse, I refer to this link (7): **https://www.maa.org/external_archive/joma/Volume8/Kalman/General.html#:~:text=The%20standard%20equation%20for%20an,parallel%20to%20the%20coordinate%20axes.**

IMPLEMENT

Because the implementation part is quite long, I would like to summarize in words:

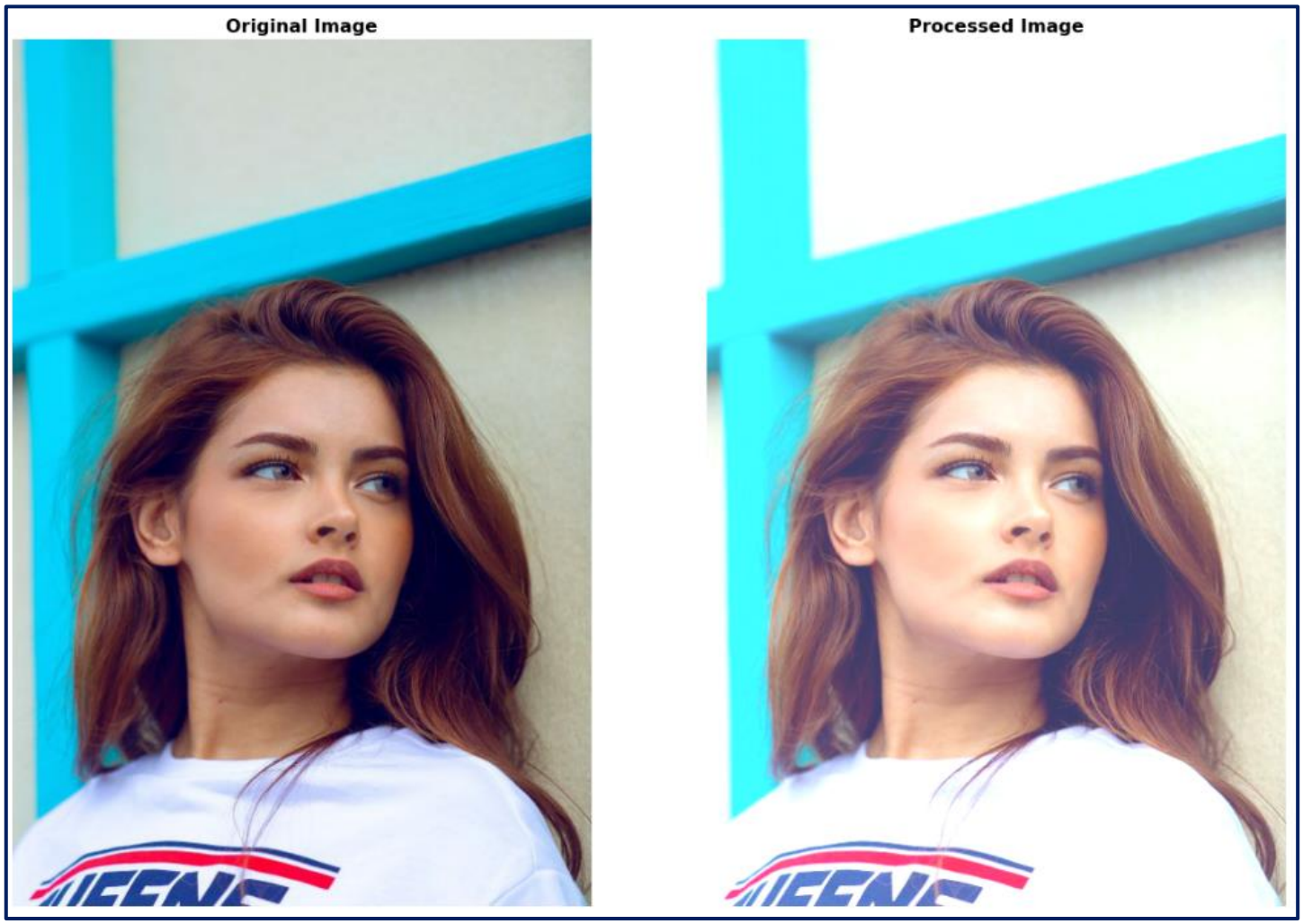
- **Input:** image (converted to 3D matrix)
- Make a hard copy by copy from original image array to new output image array (avoid shallow copy).
- Decay the image shape into rows, cols.
- Calculate the ellipse center (h, k) by using rows / 2 and cols / 2.
- After applying to the image size 512x512, I chose a = 316 and b = 172 so that the semi major axis and semi minor axis correspond to the desired result. Then divide a, b by h, k to get a general factor for all other images. For other images, just multiply a and b by the constant factor, and we get the semi major axis and the semi minor axis.
- Use for loop to access each coordinate of point (i,j) and check whether it lies outside the ellipse or not.
- Apply the formula to calculate the relative position of that point to both ellipses.
- If the point is outside both ellipses, then **blackout** this point (i,j) by setting color of that point to 0 ([0,0,0]).
- **Output**: processed output image array.

IV. RESULT ANALYSIS

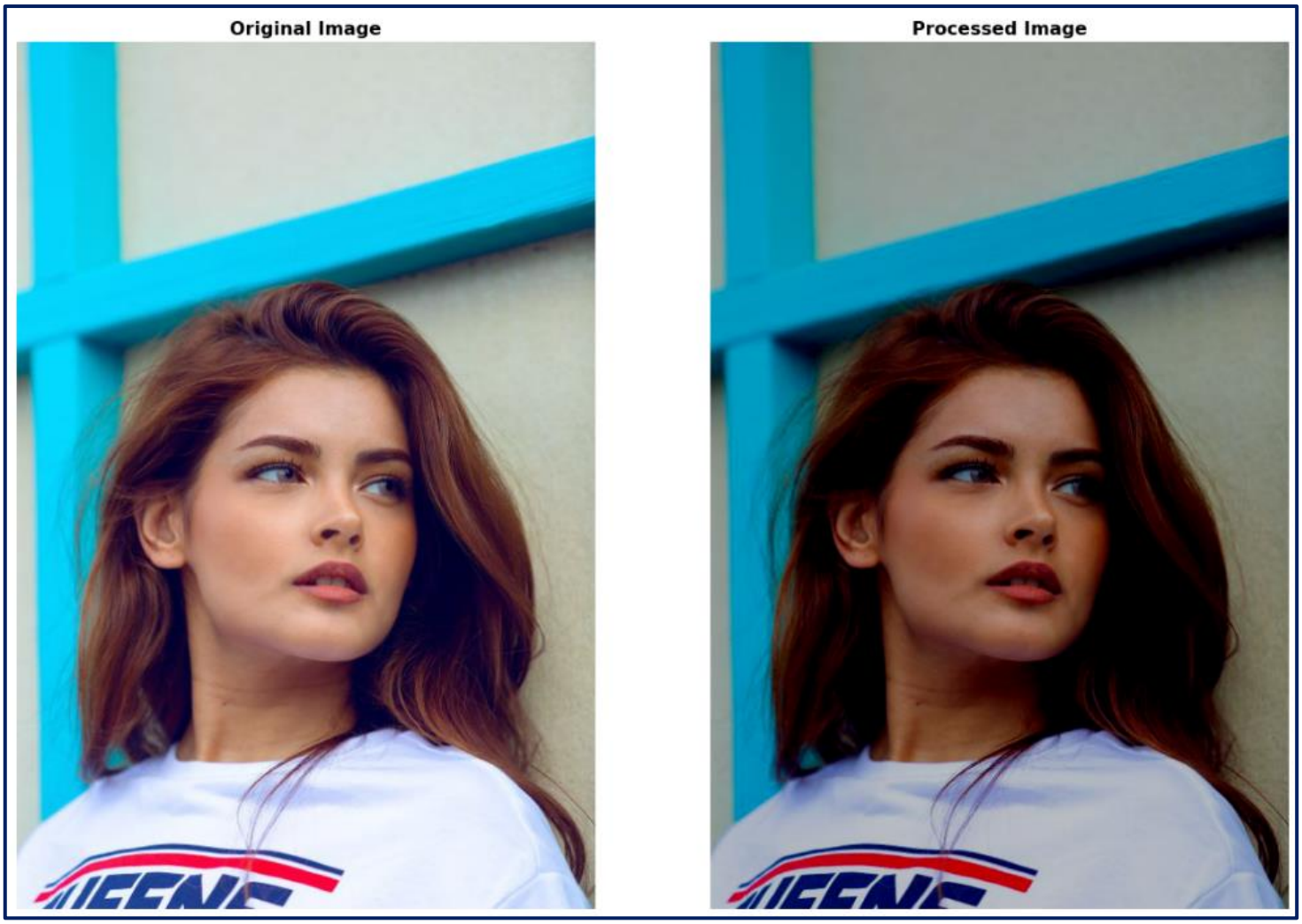
All image sources I take from unsplash.com (8), and reference documents from the lecturer.

4.1. Change the brightness

Brightness Ratio = 60



Brightness Ratio = - 60



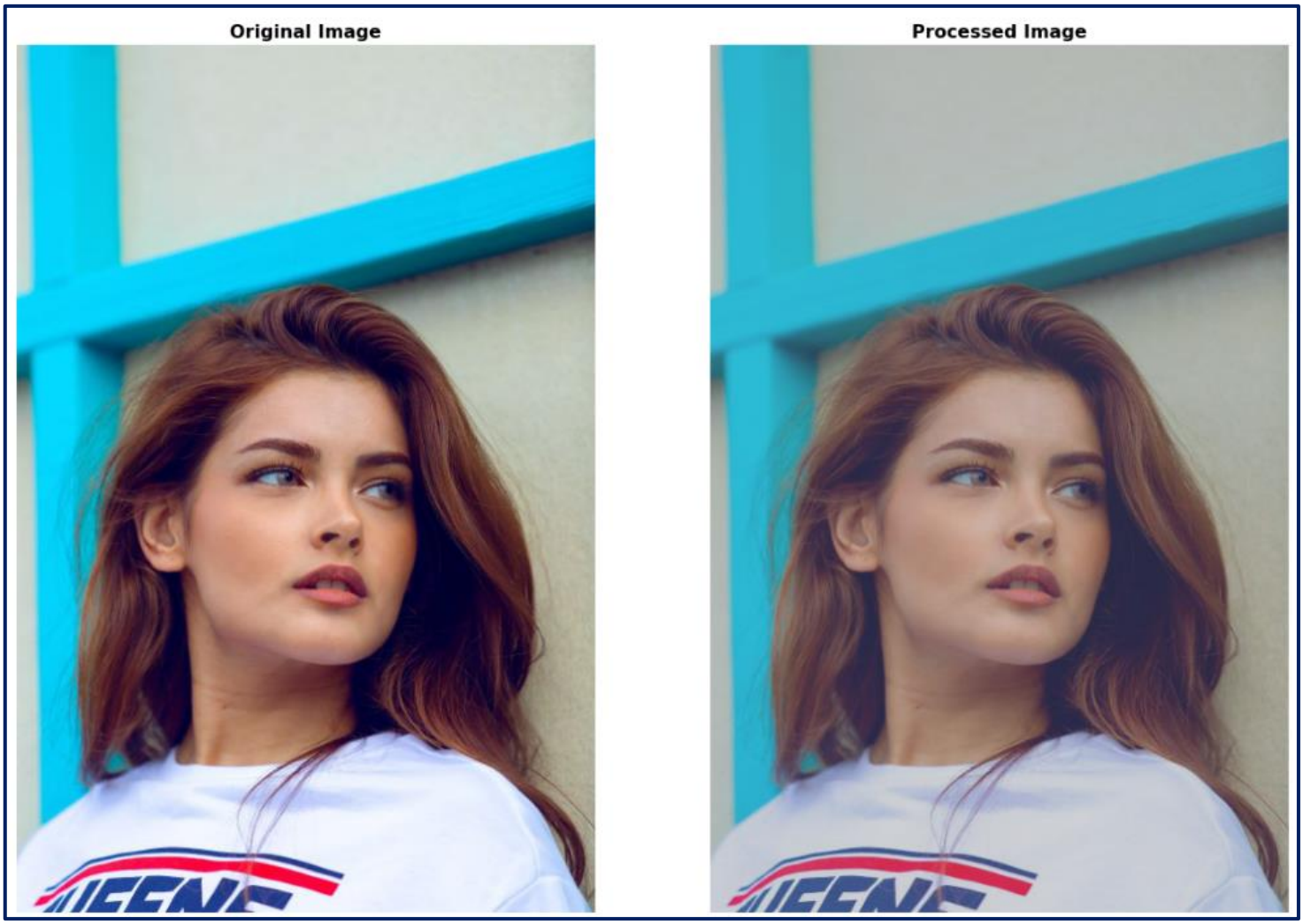
Comment:

Positive brightness ratio will brighten the image and conversely, negative brightness ratio will darken the image.

4.2. Change the contrast

Contrast Ratio = 50

Contrast Ratio = - 50



Comment:

Negative contrast ratio will decrease the amount of contrast and, conversely, positive contrast ratio will increase the amount of contrast.

4.3. Flip an image

Vertically

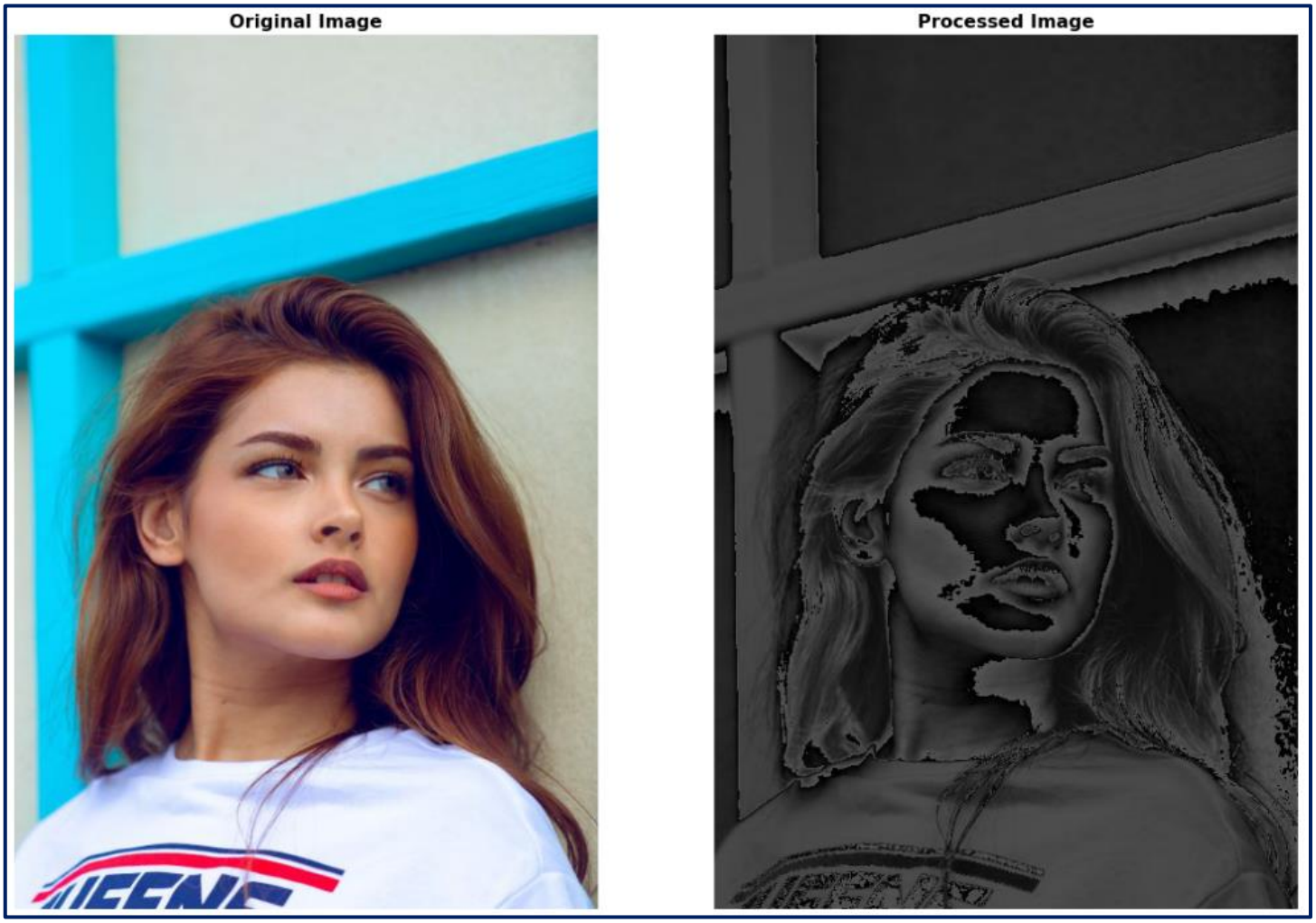
Horizontally

Comment:

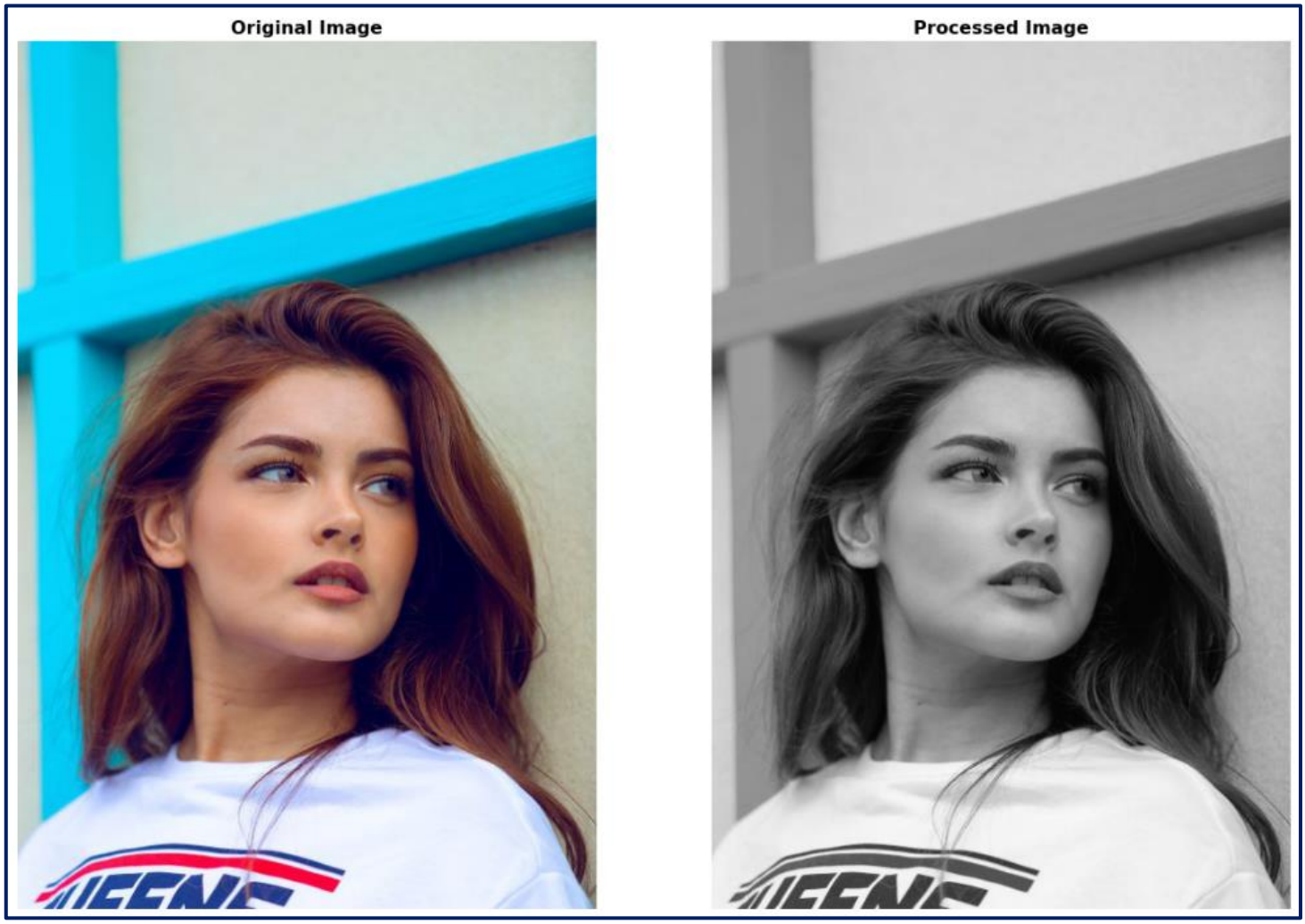
- To flip vertically: reverse rows of pixels in image.
- To flip horizontally: reverse pixels in rows in image.

4.4. Convert an RGB image to a grayscale

Average method



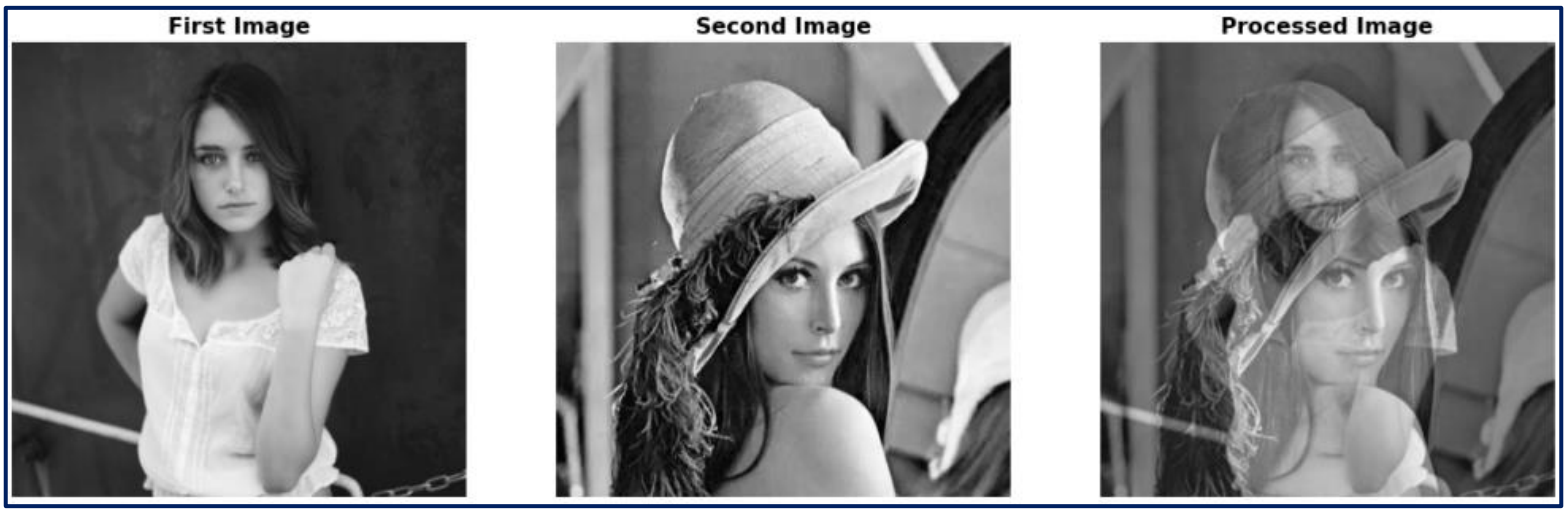
Weighted method



Comment:

- The average method, each of the color has same contribution in the image, it turned out to be a black image.
- With weighted method, since red color has more wavelength of all the three colors, and green is the color that has not only less wavelength than red color but also green is the color that gives more soothing effect to the eyes, so Red has contribute 30%, Green has contributed 59% which is greater in all three colors and Blue has contributed 11%.
- As compared to the average method, this image using weighted method is brighter and easier to see.

4.5. Blend overlapping images



4.6. Blur an image

Box blur kernel

Gaussian blur 3x3 kernel

Gaussian blur 5x5 kernel

Comment:

- There has a difference between box blur kernel and gaussian blur kernel (3x3) because: With Box Blur, divide the sum of the kernel values, which totals 9. Gaussian blur mixes the values of the neighboring pixels depending on the distance to the reference pixel. The pixels right next to the reference pixel get more of the colors from the reference pixel while the pixels farther away get less of the colors.
- If we want the image to be blurrier, we need to increase the size of the kernel, larger kernels have more values factored into the average, and this implies that **a larger kernel will blur the image more than a smaller kernel**.

4.7. Crop an image into Circle

Comment:

- The image is cropped into a circle, the points outside the cropped circle will be blacked out.
- Processing only on square images.

4.8. Crop an image into Ellipse

Comment:

- The image is cropped into 2 ellipses intersect, the points outside the cropped ellipse will be blacked out.
- Processing only on square images.

V. INSTRUCTIONS FOR USING PROGRAM

```
0. All Options
1. Adjust Brightness Of Image
2. Adjust Contrast Of Image
3. Flip An Image (Vertically / Horizontally)
4. Convert An RGB Image To A Grayscale
5. Blend Overlapping Images
6. Blur An Image
7. Crop An Image Into Circle
8. Crop An Image Into Ellipse
9. Exit
================================================
```

The program will create a menu of options for the user to choose to handle. All options require the user to enter an input image name. In specific, each option will have specific requirements for that function:

- Option 1: require the name of image, image brightness (from -255 to 255). If choose 0, default "50" value will be applied.
- Option 2: require the name of image, image contrast (from -127 to 127). If choose 0, default "50" value will be applied.
- Option 3: require the name of image, type user want to flip (horizontally / vertically).
- Option 4: require the name of image, method to convert: average or weighted.
- Option 5: require the name of image 1, the name of image 2, and the size of 2 images must be similar or the program will occur errors.
- Option 6: require the name of image, kernel user want to use: box_blur or gauss_3x3. Default kernel is box_blur.
- Option 7: require the name of image.
- Option 8: require the name of image.

After handle, the image will be exported to file with format: "name" + "type of handle" + "format type".

Note:

If user choose **option 0,** the program will sequentially execute options from 1 to 8, the user only needs to enter the name of the image to be processed once, the program will go through each option and specific requirements for that option.

VI. REFERENCES

1. https://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-5-contrast-adjustment/
2. https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.htm
3. https://docs.opencv.org/3.4/d5/dc4/tutorial_adding_images.html
4. https://github.com/t3bol90/ST-MA-Lab03
5. https://en.wikipedia.org/wiki/Kernel_(image_processing)
6. https://www.youtube.com/watch?v=BPBTmXKtFRQ&t=94s
7. https://www.maa.org/external_archive/joma/Volume8/Kalman/General.html#:~:text=The%20standard%20equation%20for%20an,parallel%20to%20the%20coordinate%20axes.
8. Image source: https://unsplash.com/