UNIVERSITY OF SCIENCE

VIETNAM NATIONAL UNIVERSITY

IMAGE PROCESSING

Lecturer

Vu Quoc Hoang
Nguyen Van Quang Huy
Le Thanh Tung
Phan Thi Phuong Uyen

Ho Chi Minh City, June 24th, 2022

TABLE OF CONTENTS

IMAGE PROCESSING

Applied Mathematics And Statistics

1. INFORMATION

Lecterer:

- Vu Quoc Hoang
- Nguyen Van Quang Huy
- Le Thanh Tung
- Phan Thi Phuong Uyen

Class: 20CLC08

Student:

- **ID:** 20127039
- **Name:** Tran Dam Gia Huy

2. CONTENTS

2.1.Approach and Implementation

2.1.1. Library Support

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

2.1.2. Image Handle

```
def ReadImage(imgName):
    imgOrg=Image.open(imgName)
    return imgOrg

def ImageToArray(imgName):
    return np.array(imgName)
```

Handle image

- Use the Image in PIL Library
- *ImageToArray*(imgName): Read a image by PIL library on device by passing imgName as parameter which inputted by user
- *ImageToArray*(imgName): Convert an imgName to 3D array by array in numpy

2.1.3. Brightness Conversion

Approach

- Add to each pixel's color with an number
- If the number is positive, the image is brighter. Otherwise, it's is darker

Implement

- brightRatio is ratio of brightness that user inputs
- To make conversion of brightness, I add each pixel's color with brightRatio value
- I used this instruction to make sure that the range of color is in [0,255]

```
max(0, min(255, int(value+brightRatio))
```

2.1.4. Contrast Conversion

Approach

- Multiply each pixel's color with an number to make more distance between colors

Implement

- contrastRatio is ratio of contrast that user inputs
- To make conversion of constrast, I change each pixel's color with formular (this formular I refer from the *Dreamland Fantasy Studio* (1), but the implement of program I do it by myself)

Red = **F** (Red – 128) + 128
Green, Blue is similar to Red

With **F** = $\frac{259(contrastRatio+255)}{255(259-contrastRatio)}$

- I used loop to access each element in image array and change each pixel's color by above formular
- I used this instruction to make sure that the range of color after changed is in [0,255]

```
max(0, min(255, int(128 + Formula*(value-128))))
```

2.1.5. Grayscale Conversion

Approach

Here, it has two method of ***Grayscale Conversion***

Average method: Replace each pixel's color by the average of three colors.

Grayscale = (R + G + B) / 3

Weighted method: It decreases the contribution of red color, and increase the contribution of the green color, and put blue color contribution in between these two. Replace each pixel's color by the below proportion

Grayscale = (0.3 * R) + (0.59 * G) + (0.11 * B)

Implement

Both of method, to implement, I use loop to access each element in image array and change the red, green, blue color by the corresponding formular. Because of the gray image, so red = green = blue = Grayscale

- Average method**:** red = green = blue = (R + G + B) / 3
- Weighted method**:** red = green = blue = (0.3 * R) + (0.59 * G) + (0.11 * B)

2.1.6. Flip Conversion

Approach

- **Horizontal flip:** Copying each pixel's color of orginal image in the forward of row and reverse of column to imgResult
- **Vertical flip:** Copying each pixel's color of orginatl image in the reverse of row and forward of column to imgResult

Implement

Horizontal flip

- Use two loops to access each pixel's color of original image
- The first loop is forward row access used for both array (orginal image and imgResult)
- The second loop is backward column access used for original array

- Because of the column of imgResult is forward access while the second loop is backward. So I need an variable index_col to access the column of imgResult in forward and increase by 1 at each element in second loop

Horizontal flip

- Use two loops to access each pixel's color of original image
- The first loop is forward column access used for both array (orginal image and imgResult)
- The second loop is backward row access used for original array
- Because of the row of imgResult is forward access while the second loop is backward. So I need an variable index_row to access the row of imgResult in forward and increase by 1 at each element in second loop

2.1.7. Combine Image

Approach

Plus each pixel's color of two image array and save it to result image array

Implement

- Use the imgResult copying first image
- Add each pixel's color of second image to imgResult by two loop accessing row and column of two image array
- While executing adding, ensure that each pixel's color of imgResult is in range [0.255] by max(0,min(255, color))

2.1.8. Blur image

Approach

Using box blur to blur image

- This is accomplished by doing a convolution between the kernel and an image
- The kernel is:

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- Replace the color in each element in matrix by calculate sum of color arround it (Because each element in kernel is 1) and divide by 9

Original Matrix				Kernel				Blur Matrix		
1	2	3	→	1	1	1	→	1	2	3
8	**7**	6		1	1	1		8	**19/3**	6
9	10	11		1	1	1		9	10	11

To replace the values at index (1,1) at original matrix to blur matrix, calculate it by:

```
blurMatrix[1][1] = (1*1 + 2*1 + 3*1 + 6*1 + 7*1 + 8*1 + 9*1 + 10*1 + 11*1 ) / 9 = 19/3
```

This is an simple example in replacement of an element at index (1,1) in matrix using box blur. To accomplish the blur processing, we need to do it for all element in the matrix

Drawback

As we saw, to accomplish the blue processing, we need to access all elements of the matrix and calculate sum of arround value. It causes the problem in time complexity because of the waste of time to sum most of arround values that have been already calculated in the previous steps. So, I suggest a solution to solve this problem by dynamic programming to store the sum of area in another matrix. And when execute box blur function, we just get the values in this matrix without calculating at each step like the previous method. *This idea I refer from The Inside Code channel (2) on youtube, but the implement of program I do it by myself*

Implement

Sum of area matrix

Orginal Matrix					Area Matrix				
1	2	3	4	21	1	3	6	10	31
8	**7**	6	5	22	9	18	27	36	79
9	10	11	12	23	18	37	57	78	144
13	14	15	16	24	31	64	99	136	226
17	18	19	20	25	48	99	153	210	325

The first element is unchanged

```
areaMatrix[0][0] = originalMatrix[0][0]
```

In the first row (i=0 and j!=0)

```
areaMatrix[i][j] = originalMatrix[i][j] + areaMatrix[i][j-1]
```

Example

- areaMatrix[0][1] = 2 + 1 = 3
- areaMatrix[0][2] = 3 + 3 = 6

In the first column (i!=0 and j=0)

```
areaMatrix[i][j] = originalMatrix[i][j] + areaMatrix[i-1][j]
```

Example

- areaMatrix[2][0] = 9 + 9 = 18
- areaMatrix[3][0] = 13 + 18 = 31

Other case (i!=0 and j!=0)

```
areaMatrix[i][j] = originalMatrix[i][j] + areaMatrix[i-1][j] + areaMatrix[i][j-1] - areaMatrix[i-1][j-1]
```

Example

- areaMatrix[3][2] = 15 + 57 + 64 – 37 = 99
- areaMatrix[2][3] = 12 + 36 + 57 – 27 = 78

From area matrix to calculate sum of arround color at corresponding pixel

Orginal Matrix

1	2	3	4	21
8	7	6	5	22
9	10	11	12	23
13	14	15	16	24
17	18	19	20	25

Area Matrix

1	3	6	10	31
9	18	27	36	79
18	37	57	78	144
31	64	99	136	226
48	99	153	210	325

Result Matrix

1	2	3	4	21
8	7	6	5	22
9	10	32/3	12	23
13	14	15	16	24
17	18	19	20	25

To replace the color of pixel at index [i][j]. I calculated it from AreaMatrix by the formular

```
imgResult[i][j] = (areaMatrix[i+1][j+1] + areaMatrix[i-2][j-2] - areaMatrix[i+1][j-2] - areaMatrix[i-2][j+1])/9
```

Example

- imgMatrix[2][2] = (136 + 1 - 10 – 31) /9 = 32/3 => It just calls the result with simple addition from areaMatrix that calculated before. Use loop from 2 to row-1 and column-1 to make sure that at any location has enough neighbors on areaMatrix to calculate.
- In the traiditional calculation, it means imgMatrix[2][2] = (7 + 6 + 5 + 10 + 11 + 12 + 14 + 15 + 16)/ 9 = 32/3 => Waste of time to sum most of arround values that have been already calculated in the previous steps.

Overall, I implement the areaMatrix and use for it later to replace pixel's color is an effective way to solve the time complexity in algorithms

2.1.9. Circle Frame

Approach

- The pixel'color at any position that is outside the circle is set to black
- Use euclidean distance to calculate distance from all positions to center of circle

Implement

- Height, width of shape are equal to img.shape[0] and img.shape[1] (The number of rows and columns)
- Calculate center[x,y] of circle by dividing height and width with 2 (in order x,y)
- Calculate radius of circle by dividing min(height, width) with 2 because I want to have a generality equation in both rectangle and square shape. It means rectangle I get the radius by height/2, if I calculate it by width/2, the circle is too big. By the view of square shape, it has no problem because all edges are equal
- Calculate the euclidean distance between each pixel'color position to center by two loop accessing. If the distance is greater than than radius of circle, it means this position is outside the circle, I set the pixel's color of it to black ([0,0,0])

2.1.10.Eclipse Frame

Approach

- The pixel'color at any position that is outside the two eclipses is set to black
- Using relative position of the eclipse and a point to determine whether it is outside or inside the eclipse
- Using complement of vene diagram to handle two eclipse's black replacement

Implement

The non-canonical ellipse equation. *This equation I refer from stackexchange forum (3), but the implement of program I do it by myself*

$$\frac{((x-h)\cos(A)+(y-k)\sin(A))^2}{a^2}+\frac{((x-h)\sin(A)-(y-k)\cos(A))^2}{b^2}=1$$

(h,k) is the position of eclipse center. (a,b) is semi-axis of eclipse. A is the angle measured from x axis with trigonometric circle direction

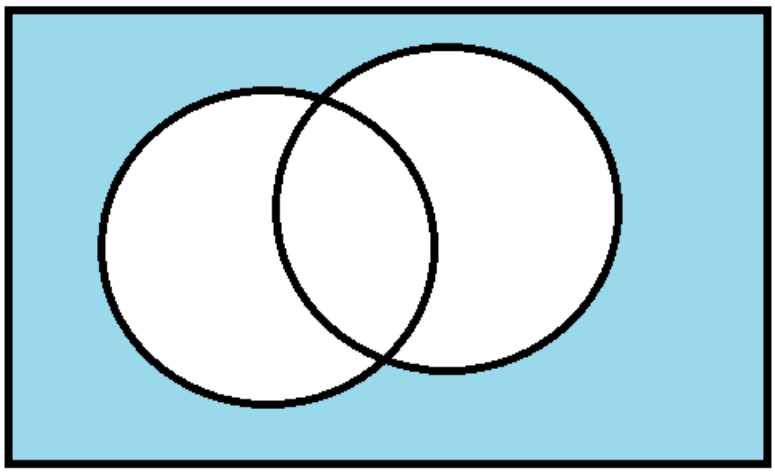
- Heigth, width of shape are equal to img.shape[0] and img.shape[1] (The number of rows and columns)
- Calculate center[x,y] of eclipse by dividing height and width with 2 (in order x,y)
- After the number of times doing trial and error, I finally found the proportion of a,b (semi-axis) for all images. I use min() to generlize equation for rectangles and squares

a = min(height, width)*45/128
b = a$\sqrt{3}$

This project requires two 45° rotated eclipses intersected at center. In trigonometric circle, the first eclipse is rotated 135° as $\frac{3\pi}{4}$. The another one is rotated 45° as $\frac{\pi}{4}$

I use loop to accessing each element and calculate the result by the non-canonical ellipse equation above to check whether this position is outside the eclipse or not

After that, I replace all positions are outside the two eclipses (eclipse_1 > 1 and eclipse_2 > 1) by black ([0,0,0]). It is similar to the complement of vene diagram below



2.2. Result

2.2.1. Brightness conversion

Orginal image

brightRatio = 50 (brighter)

brightRatio = -50 (darker)

2.2.2. Contrast conversion

Orginal image

contrastRatio = 20

contrastRatio = 100

2.2.3. Grayscale conversion

Orginal image

Average method

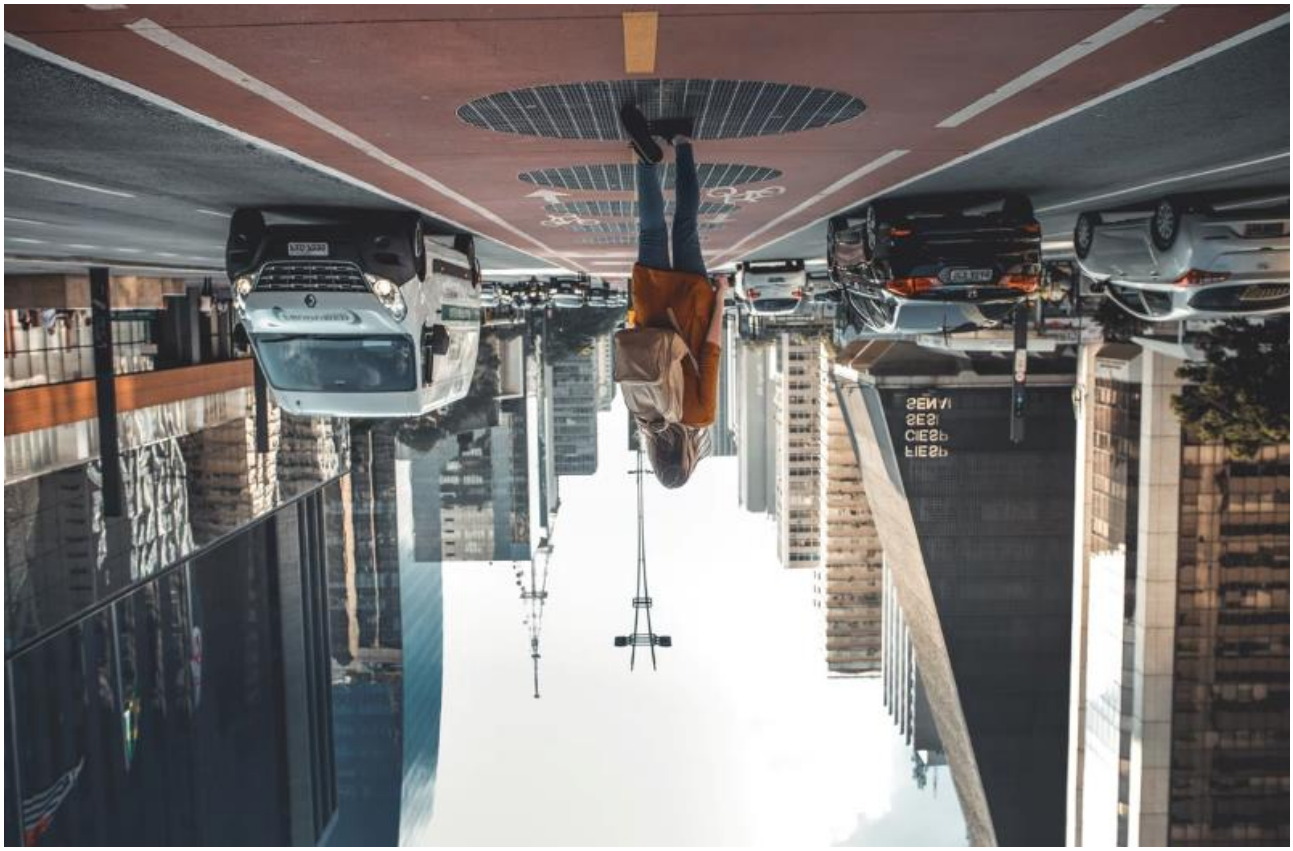
Weighted method

2.2.4. Flip conversion

Orginal image

Horizontal flip

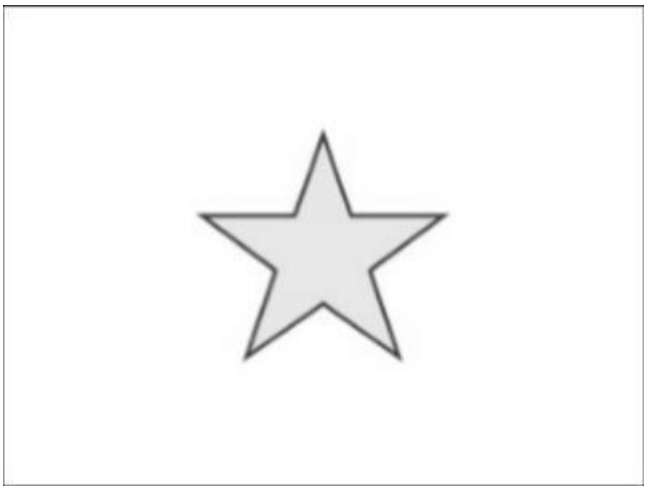
Vertical flip



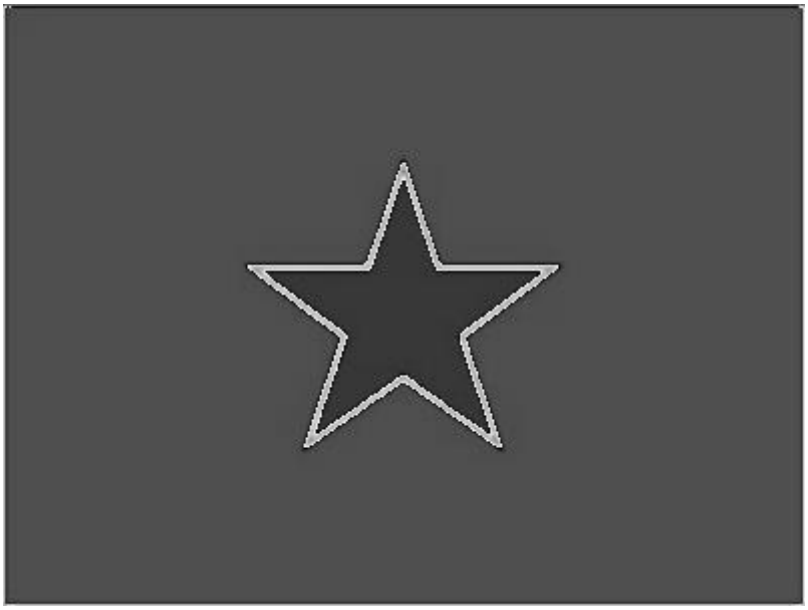
2.2.5. Combine images

Image 1

Image 2



After combine two images



2.2.6. Blur image

Original image

Size of image is 800 x 533. Time executing: ~6s for AMD R5 5500U

Orginal image

Size of image is 512x512. Time executing ~4s for AMD R5 5500U

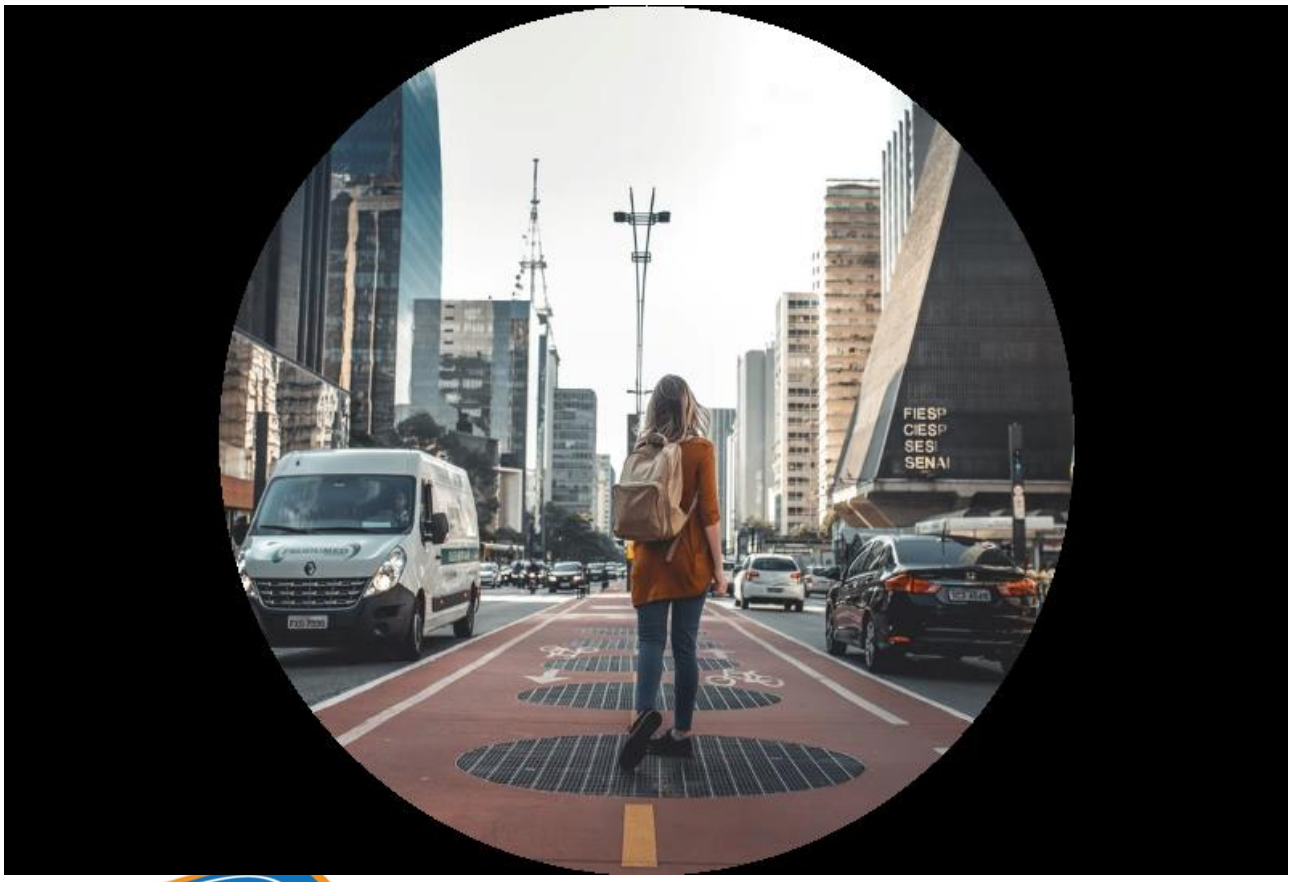
2.2.7. Cricle Frame

Square image

Rectangle Image

Square image

Rectangle Image



2.2.8. Eclipse image

Square image

Rectangle Image

Square Image

Rectangle Image

3. COMPLETION

NO.	FUNCTION	DESCRIPTION	COMPLETE
1	Bright conversion	Make the image brighter or darker	100%
2	Contrast conversion	Change the contrast of the image	100%
3	Grayscale	Change RGB image to gray image by average method and weighted method	100%
4	Flip conversion	Flip the image by horizontal and vertical	100%
5	Combine image	Combine two gray images	100%
6	Blur image	Using box blur to blur image and the time executing is under 20s	100%
7	Bonus	Crop the image with circle frame and two intersect eclipse frame	100%
Total completion			**100%**

4. REFERENCES

- Contrast conversion (1): https://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-5-contrast-adjustment/
- Box blur (2): https://www.youtube.com/watch?v=4Eh0y3LHTNU
- Non-canonical ellipse equation (3): https://math.stackexchange.com/questions/426150/what-is-the-general-equation-of-the-ellipse-that-is-not-in-the-origin-and-rotate
- Kernel (Image processing): Kernel (image processing) - Wikipedia
- Gray scale: https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.htm
- Numpy library: https://numpy.org/doc/stable/reference/generated/numpy.array.html
- Pillow library: https://pillow.readthedocs.io/en/stable/reference/Image.html
- Link image: https://unsplash.com/photos/SaVlzqe9068