

ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

---

## BÁO CÁO ĐỒ ÁN 02 IMAGE PROCESSING

---

MÔN HỌC: TOÁN ỨNG DỤNG VÀ THỐNG KÊ CHO  
CÔNG NGHỆ THÔNG TIN

Sinh viên thực hiện:  
Đoàn Thị Yên Nhi  
MSSV: 21127660

Giảng viên hướng dẫn:  
Phan Thị Phương Uyên

Thành phố Hồ Chí Minh, tháng 8 năm 2023



## Mục lục

1 Các chức năng đã hoàn thành	2
2 Thay đổi độ sáng cho ảnh	2
3 Thay đổi độ tương phản	4
4 Lật ảnh ngang - dọc	6
5 Chuyển đổi ảnh RGB thành ảnh xám/sepia	7
6 Làm mờ/sắc nét ảnh	9
7 Cắt ảnh theo kích thước (cắt ở trung tâm)	11
8 Cắt ảnh theo khung hình tròn	12
9 Cắt ảnh có khung là 2 hình ellip chéo nhau	13
10 Hàm main xử lý	14
11 Tài liệu tham khảo	14

# 1 Các chức năng đã hoàn thành

STT	Chức năng	Mức độ hoàn thành
1	Thay đổi độ sáng cho ảnh	100%
2	Thay đổi độ tương phản	100%
3	Lật ảnh	Ngang
		Dọc
4	Chuyển đổi ảnh RGB thành ảnh	Xám
		Sepia
5	Làm	Mờ
		Sắc nét ảnh
6	Cắt ảnh theo kích thước (cắt ở trung tâm)	100%
7	Cắt ảnh theo khung hình tròn	100%
8	Cắt ảnh theo khung là 2 hình ellipse chéo nhau	90%

Bảng 1: Các chức năng đã hoàn thành

## 2 Thay đổi độ sáng cho ảnh

### 2.1 Ý tưởng thực hiện

Để thay đổi độ sáng của pixels, chỉ cần tăng hay giảm giá trị của từng pixel bằng cách sử dụng một khoảng cường độ (intensity) mà ta muốn thay đổi:

- Để tăng độ sáng: cộng ma trận hình ảnh với lượng intensity nói trên.
- Ngược lại, để giảm độ sáng cho hình ảnh thì ta sử dụng phép trừ.

Tuy nhiên, cần lưu ý giá trị sau khi thay đổi phải nằm trong khoảng [0,255].

### 2.2 Hàm chức năng

Dùng np.array và img.shape để lấy ra kích thước của mảng lần lượt là chiều cao, chiều rộng và số kênh(ví dụ: 3 kênh cho hình ảnh màu RGB) của bức ảnh.

```
1 img = np.array(image)
2 height, width, _ = img.shape
```

np.zeros() + intensity để tạo một mảng mới có kích thước với giống với hình ảnh đầu vào, và tất cả các giá trị của mảng này đều bằng intensity. Điều này tạo ra một hình ảnh mới có cùng kích thước và màu sắc với hình ảnh gốc, nhưng màu sắc nó đều như nhau và bằng giá trị intensity. Điều này làm tăng độ sáng của ảnh.

```
1 bright_image=np.zeros((height, width, _)) + intensity
2 bright_image += img
```

Dùng np.where để đảm bảo các giá trị trong mảng bright image không vượt quá khoảng giá trị [0,255]. Các giá trị nhỏ hơn 255 sẽ được giữ nguyên, còn nếu vượt quá 255 sẽ được trả về 255. Tương tự, các giá trị lớn hơn 0 được giữ nguyên, còn nhỏ hơn 0 sẽ được cắt về 0.

```

1 bright_image = np.where(bright_image > 255, bright_image, 255)
2 bright_image = np.where(bright_image < 0, bright_image, 0)

```

Dùng `astype(np.uint8)` để chuyển đổi kiểu dữ liệu của mảng thành kiểu dữ liệu `uint8` phù hợp với một hình ảnh 8-bit unsigned integer.

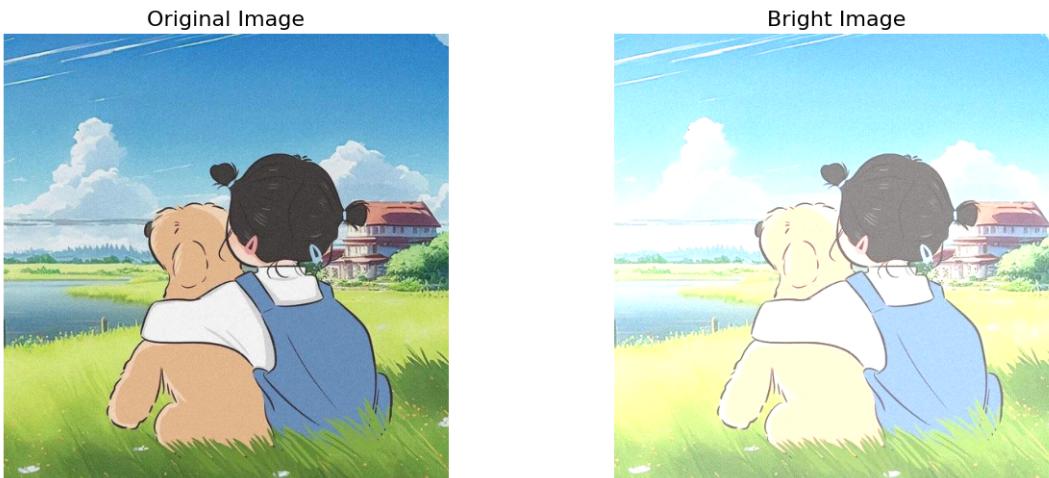
```

1 result = bright_image.astype(np.uint8)
2 result = Image.fromarray(result)

```

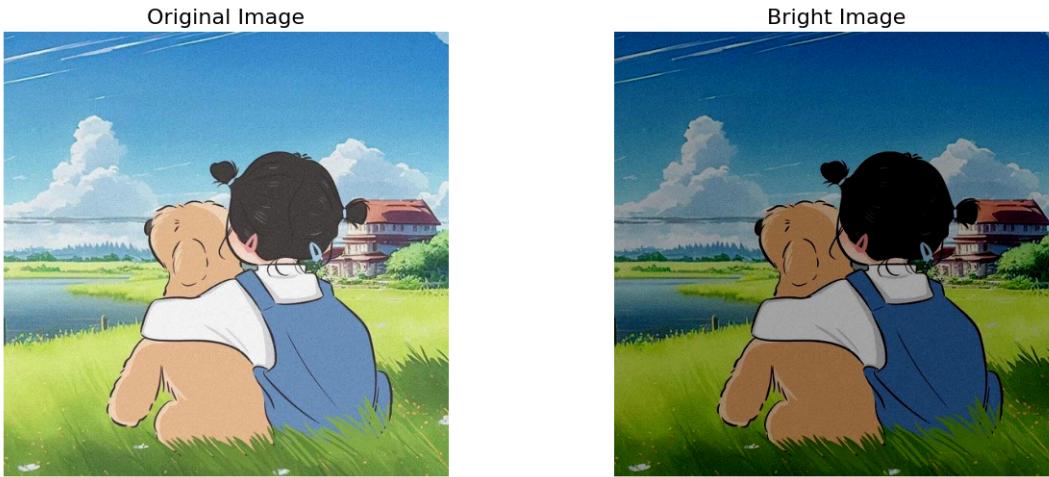
## 2.3 Kết quả

Với giá trị `intensity = 80`, ta được kết quả là hình ảnh đã được tăng độ sáng như sau:



Hình 1: Kết quả tăng độ sáng

Ngược lại, để giảm độ sáng thì giá trị của `intensity < 0`, cụ thể hình ảnh dưới đây được giảm một lượng `intensity = -80`:



Hình 2: Kết quả giảm độ sáng

### 3 Thay đổi độ tương phản

#### 3.1 Ý tưởng thực hiện

Thay đổi độ tương phản là làm thay đổi độ chênh lệch sáng tối. Gọi hai điểm ảnh là 1 và 2 thì ta phải làm sao cho khoảng cách của hai điểm ảnh đó ra xa nhau nhất có thể, bằng cách sử dụng phép nhân, nhân tất cả các pixel trong hình ảnh với một hệ số gamma mong muốn:

- Với  $\gamma = 1.0$  là tương đương hình gốc.
- Với  $\gamma > 1.0$  làm tăng độ tương phản.
- Với  $\gamma < 1.0$  làm giảm độ tương phản.

Cũng giống như thay đổi độ sáng, ta cũng cần chú ý giá trị sau khi thay đổi phải nằm trong khoảng  $[0, 255]$ .

#### 3.2 Hàm chức năng

Dùng `np.array` để chuyển hình ảnh đầu vào thành một mảng numpy, cho phép thực hiện các phép tính toán số học trên hình ảnh.

```
1 img=np.array(image)
```

Thực hiện điều chỉnh độ tương phản bằng cách nhân toàn bộ hình ảnh với giá trị 0 mong muốn.

```
1 contract_image = img*gamma
```

Cũng giống như thay đổi độ sáng, sử dụng `np.where` để đảm bảo rằng các giá trị của mảng hình ảnh sau khi thay đổi nằm trong khoảng  $[0, 255]$ .

```
1 contract_image=np.where(contract_image < 255, contract_image, 255)
2 contract_image=np.where(contract_image > 0, contract_image, 0)
```

Và chuyển đổi kiểu dữ liệu mảng thành kiểu dữ liệu `uint8`. Sử dụng `fromarray` để tạo một đối tượng hình ảnh đã được chuyển đổi. Hình ảnh mới có độ tương phản đã được chỉnh theo gamma.

```
1 result = contract_image.astype(np.uint8)
2 result = Image.fromarray(result)
```

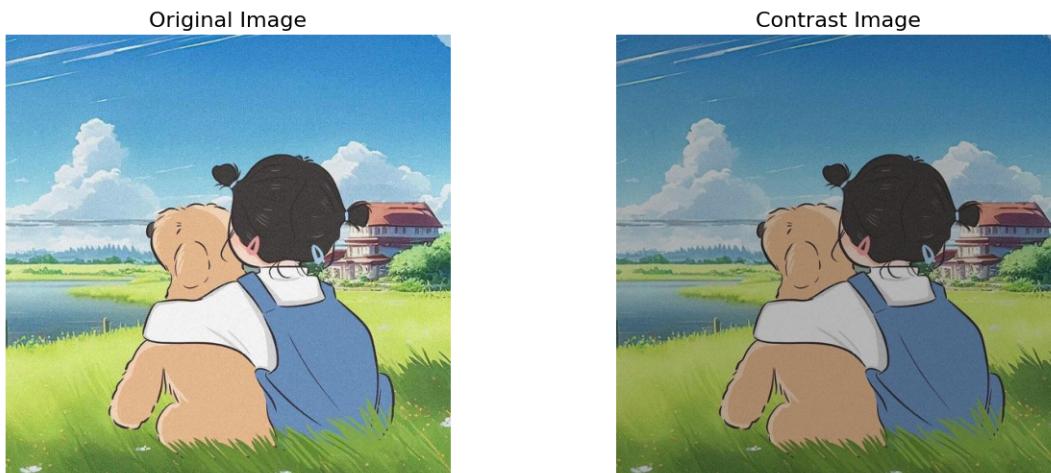
### 3.3 Kết quả

Làm tăng độ tương phản với gamma = 1.5 ta sẽ có kết quả như sau:



Hình 3: Kết quả tăng độ tương phản

Ngược lại, với gamma = 0.5 ta có kết quả:



Hình 4: Kết quả giảm độ tương phản

## 4 Lật ảnh ngang - dọc

### 4.1 Ý tưởng thực hiện

Để lật ngang (hoặc lật dọc) bằng cách đảo ngược thứ tự các cột (hoặc hàng) của mảng hình ảnh.

### 4.2 Hàm chức năng

Cũng giống như các chức năng khác, vẫn sẽ dùng `np.array` để chuyển hình ảnh thành mảng numpy. `img[:, :, :]` là cách lấy toàn bộ mảng `img`.

Lật ngang ảnh:

```
1 img=img[:, ::-1, :]
```

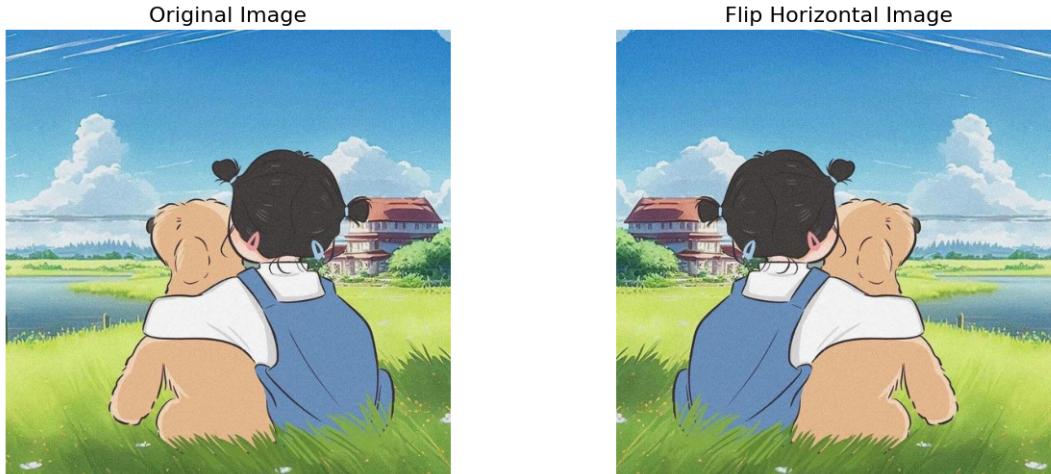
Khi sử dụng `::-1` ở phần cắt của mảng, nó đảo ngược thứ tự của các phần tử trong chiều tương ứng. Trong trường hợp này, sử dụng `::-1` để lật ngang (đảo ngược) toàn bộ các cột trong mảng hình ảnh, tạo thành hiệu ứng lật ngang của hình ảnh ban đầu.

Lật dọc ảnh:

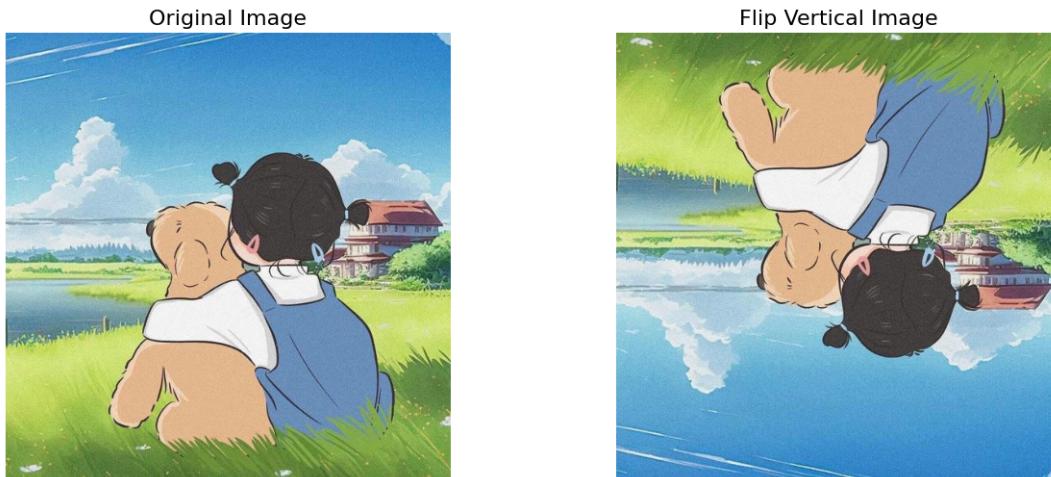
```
1 img=img[::-1, :, :]
```

Tương tự với lật ngang, thì lật dọc hình ảnh cũng sẽ sử dụng `::-1` để lật dọc toàn bộ các cột trong mảng hình ảnh, tạo hiệu ứng lật dọc của hình ảnh ban đầu.

### 4.3 Kết quả



Hình 5: Kết quả lật ngang ảnh



Hình 6: Kết quả lật dọc ảnh

## 5 Chuyển đổi ảnh RGB thành ảnh xám/sepia

### 5.1 Ý tưởng thực hiện

**Chuyển đổi ảnh RGB sang ảnh xám:**

- Phương pháp đơn giản là lấy giá trị trung bình của ba thành phần (đỏ, lục và lam) làm giá trị thang độ xám:

$$\text{grayscale} = \frac{\mathbf{R} + \mathbf{G} + \mathbf{B}}{3}$$

- Tuy nhiên, theo cách này vì nó gán trọng số như nhau cho mỗi thành phần nên kết quả sẽ không được hiệu quả như mong muốn. Do đó, ta sử dụng công thức sau:

$$\text{grayscale} = 0.3 * \mathbf{R} + 0.59 * \mathbf{G} + 0.11 * \mathbf{B} [1]$$

**Chuyển đổi ảnh RGB sang ảnh sepia:**

- Để chuyển đổi ảnh RGB sang ảnh sepia, sử dụng công thức [2]:

$$\text{newRed} = 0.393 * \mathbf{R} + 0.769 * \mathbf{G} + 0.189 * \mathbf{B}$$

$$\text{newGreen} = 0.349 * \mathbf{R} + 0.686 * \mathbf{G} + 0.168 * \mathbf{B}$$

$$\text{newBlue} = 0.272 * \mathbf{R} + 0.534 * \mathbf{G} + 0.131 * \mathbf{B}$$

### 5.2 Hàm chức năng

**Chuyển đổi ảnh RGB thành ảnh xám:**

Trích xuất các kênh màu đỏ, xanh lá và xanh dương từ mảng hình ảnh. Tính toán giá trị xám tại mỗi điểm ảnh bằng cách lấy trung bình có trọng số của các kênh màu theo tỉ lệ 0.3(đỏ), 0.59(xanh lá) và 0.11(xanh dương).

```

1 R=img[:, :, 0]
2 G=img[:, :, 1]
3 B=img[:, :, 2]
4 gray=(R*0.3+G*0.59+B*0.11)

```

Cũng như các chức năng khác thì sử dụng astype(np.uint8) để chuyển đổi mảng giá trị xám thành kiểu dữ liệu unsigned 8-bit integer. Và sử dụng np.dstack() chuyển đổi mảng giá trị xám thành hình ảnh RGB với cùng giá trị xám cho tất cả các kênh màu.

```
1 rgb_image= np.dstack((result, result, result))
```

### Chuyển đổi ảnh RGB thành ảnh sepia:

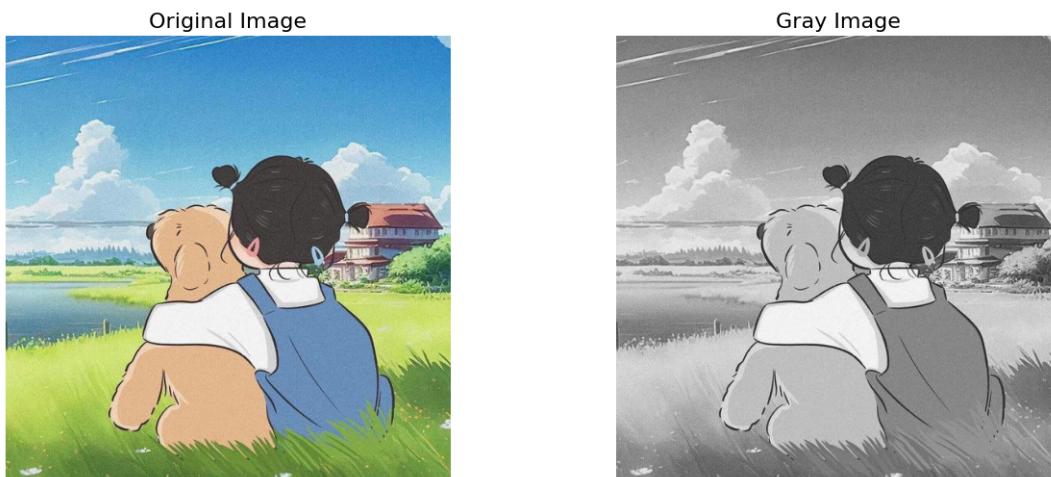
Thực hiện chuyển đổi hình ảnh RGB thành hình ảnh sepia bằng cách tính toán giá trị mới cho từng kênh màu với công thức ở phần ý tưởng.

```
1 img=np.dstack((R*0.393 + G*0.769 + B*0.189 ,  
2 R*0.349 + G*0.686 + B*0.168 ,  
3 R*0.272 + G*0.534 + B*0.131))
```

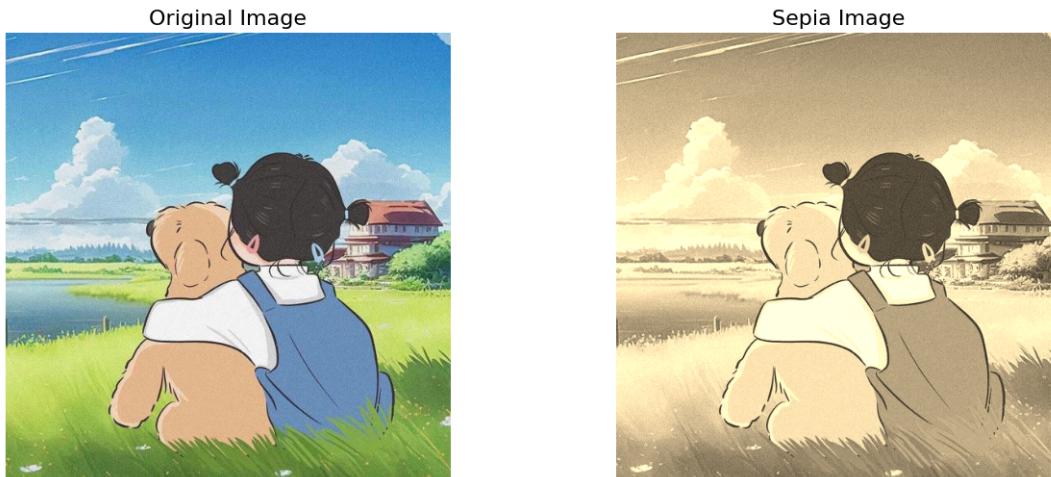
Ở cả hai phần chuyển đổi RGB thành ảnh xám và thành ảnh sepia, vẫn sẽ sử dụng np.where() để đảm bảo rằng các giá trị của mỗi pixel trong hình ảnh luôn nằm trong khoảng [0,255].

```
1 img=np.where(img < 255, img, 255)  
2 img=np.where(img > 0, img, 0)
```

## 5.3 Kết quả



Hình 7: Kết quả chuyển RGB thành ảnh xám



Hình 8: Kết quả chuyển RGB thành ảnh sepia

## 6 Làm mờ/sắc nét ảnh

### 6.1 Ý tưởng thực hiện

Làm mờ ảnh bằng cách sử dụng công thức Gaussian Blur 3x3 [3]:

$$1/16 * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Làm sắc nét ảnh bằng cách sử dụng công thức Sharpen [3]:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

### 6.2 Hàm chức năng

#### Làm mờ ảnh:

Sử dụng công thức ở phần ý tưởng để làm mờ ảnh.

```
1 kernel = np.array([[1.0, 2.0, 1.0], [2.0, 4.0, 2.0], [1.0, 2.0, 1.0]])
2 kernel = kernel/16
```

Tạo một mảng mới có kích thước tương tự với hình ảnh ban đầu để lưu trữ kết quả làm mờ. Là một mảng dạng số thực để tránh việc tràn số trong quá trình tính toán. Thực hiện 3 vòng lặp với vòng lặp đầu tiên lên chiều cao, vòng lặp thứ hai lên chiều rộng, và cuối cùng là vòng lặp lên các kênh màu (R, G, B) của hình ảnh. Tính toán giá trị mới cho từng kênh màu tại mỗi điểm ảnh bằng cách thực hiện phép tính chập giữa ma trận kernel và các vùng lân cận 3x3 tương ứng của hình ảnh. Kết quả là giá trị làm mờ tại điểm này cho từng kênh màu.

```
1 blur_image = np.zeros((height, width, _), float)
2 for h in range(1, height - 1):
3     for w in range(1, width - 1):
4         for k in range(3):
5             blur_image[h, w, k] = float(np.sum(img[h-1:h+2, w-1:w+2, k]*kernel))
```

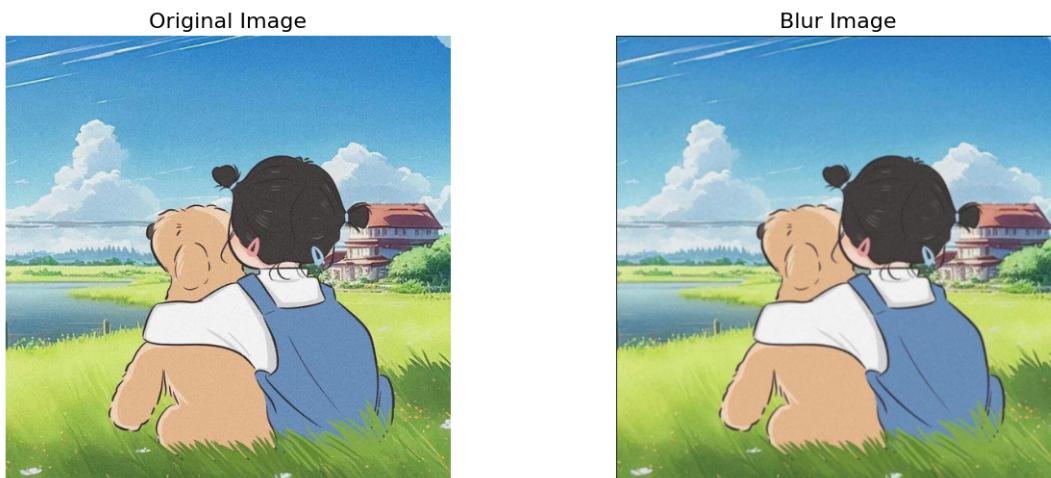
**Làm sắc nét ảnh:**

Làm sắc nét ảnh cũng tương tự như làm mờ ảnh. Tuy nhiên, mảng kernel sẽ tương ứng với công thức ở phần ý tưởng như sau:

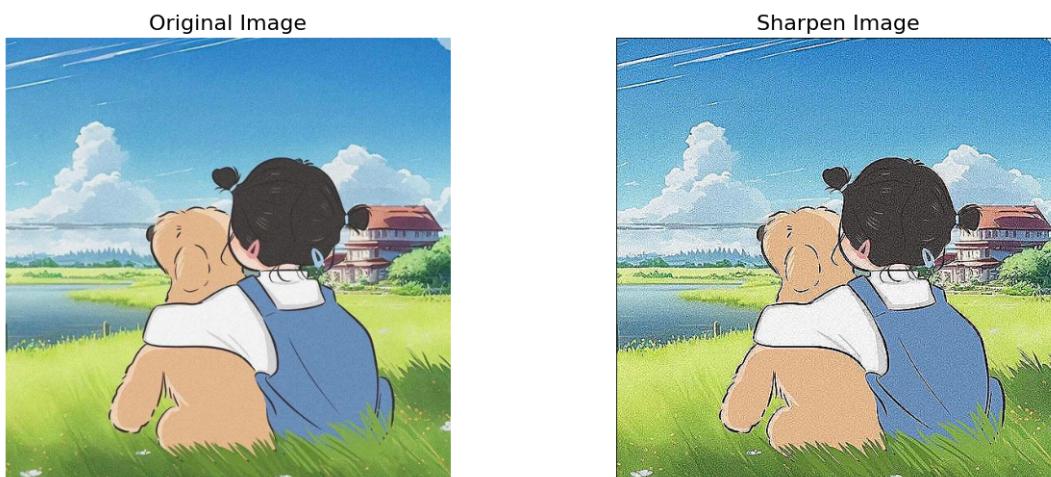
```
1 kernel = np.array([[0,-1,0], [-1,5,-1], [0,-1,0]])
```

Cũng sẽ sử dụng các hàm như các chức năng khác để đảm bảo các giá trị của mỗi pixel trong hình ảnh nằm trong khoảng giá trị của 8-bit unsigned integer.

### 6.3 Kết quả



Hình 9: Kết quả làm mờ ảnh



Hình 10: Kết quả làm sắc nét ảnh

## 7 Cắt ảnh theo kích thước (cắt ở trung tâm)

### 7.1 Ý tưởng thực hiện

Xác định kích thước cắt (chiều rộng và chiều cao) muốn cắt từ hình ảnh ban đầu. Xác định tọa độ x, y để cắt hình ảnh theo kích thước cắt. Thực hiện việc cắt hình ảnh bằng cách sử dụng tọa độ x, y đã tính để lấy phần hình ảnh trung tâm có kích thước cắt.

### 7.2 Hàm chức năng

Chuyển đổi hình ảnh đầu vào thành mảng numpy. Trích xuất chiều cao, chiều rộng và số kênh màu của hình ảnh.

```
1 img=np.array(image)
2 row, col, _ = img.shape
```

Xác định tọa độ hàng và cột để chọn vùng hình ảnh có kích thước như mong muốn tại vị trí trung tâm của hình ảnh.

```
1 [language=Python]
2 row = row//2 - 95
3 col = col//2 - 95
```

Thực hiện việc cắt hình ảnh bằng cách chọn vùng hình ảnh có kích thước mong muốn. Ở dưới đây là kích thước 225x225 pixels.

```
1 img=img[row:row + 225, col:col +225, :]
```

### 7.3 Kết quả



Hình 11: Kết quả cắt ảnh theo kích thước

## 8 Cắt ảnh theo khung hình tròn

### 8.1 Ý tưởng thực hiện

Viết phương trình đường tròn. Những điểm nằm trong và trên đường tròn đó sẽ giữ nguyên màu. Còn ngược lại, những điểm còn lại sẽ chuyển đổi nó thành màu đen.

### 8.2 Hàm chức năng

Xác định chiều cao và chiều rộng của hình ảnh.

```
1 img=np.array(image)
2 row = img.shape[0]
3 col = img.shape[1]
```

Sử dụng np.ogrid để tạo mảng 2D, chứa các giá trị từ 0 đến 'row-1' cho x, từ 0 đến 'col-1' cho y. Các mảng này sẽ được sử dụng để tạo tọa độ cho từng điểm ảnh trong hình ảnh.

```
1 y, x = np.ogrid[:col, :row]
2 center_x = col//2
3 center_y = row//2
```

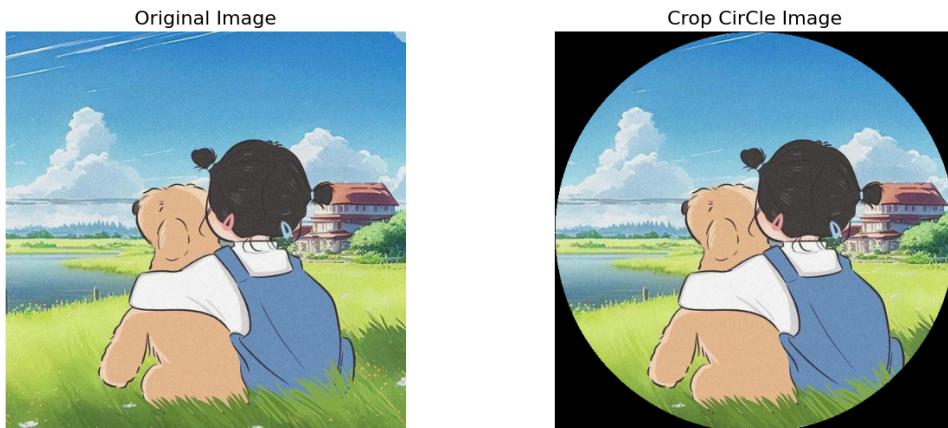
Tạo 'mask' dạng hình tròn bằng cách tính khoảng cách từ mỗi điểm ảnh đến trung tâm hình ảnh và kiểm tra xem khoảng cách đó có nhỏ hơn hoặc bằng bán kính của hình tròn hay không. Kết quả là một mảng boolean có kích thước giống với kích thước hình ảnh, mà giá trị True tương ứng với các điểm ảnh nằm trong vùng tròn và False tương ứng với các điểm ảnh nằm ngoài vùng tròn.

```
1 mask=(x-center_x)**2 + (y-center_y)**2 <= (row//2)**2
```

Áp dụng 'mask' lên hình ảnh để chỉ giữ lại các điểm ảnh nằm trong vùng tròn (có giá trị True trong mảng mask). Các điểm nằm ngoài vùng tròn sẽ có giá trị 0, làm cho chúng trở thành điểm đen trong kết quả. Tạo thành bức ảnh đã được cắt theo khung hình tròn.

```
1 crop=img*mask[:, :, np.newaxis]
```

### 8.3 Kết quả



Hình 12: Kết quả cắt ảnh theo khung hình tròn

## 9 Cắt ảnh có khung là 2 hình ellip chéo nhau

### 9.1 Ý tưởng thực hiện

Cũng tương tự như cắt ảnh theo khung hình tròn, để cắt ảnh theo khung là 2 hình ellip chéo nhau, bước đầu là viết phương trình 2 ellip với trục x, y là 2 chiều cao và rộng của hình ảnh. Ban đầu 2 hình ellip sẽ nằm ngang và dọc theo chiều của x, y. Để nó nằm chéo nhau theo yêu cầu của đề tài, thực hiện phép quay 45 độ với hình ellip. Sau đó thực hiện phép or để lấy những điểm đều nằm trong cả 2 hình ellip. Những điểm đó sẽ được giữ nguyên màu, còn những điểm không thuộc sẽ đổi nó thành màu đen.

### 9.2 Hàm chức năng

```
1 height = img.shape[0]
2 width = img.shape[1]
3 y, x = np.ogrid[:height, :width]
```

Xác định các tham số cho hai hình ellip và góc quay 45 độ của chúng. Trong quá trình làm bài, em đã tìm ra các tham số theo điều kiện để phù hợp với phương trình ellip của mình như sau:

```
1 rotation_angle=45
2 ellipse1_center=(width/2, height/2)
3 ellipse2_center=(width/2, height/2)
4 ellipse1_radius_x=width//3.25
5 ellipse1_radius_y=height//3.25
6 ellipse2_radius_x=width//3.25
7 ellipse2_radius_y=height//3.25
```

Áp dụng phép quay tọa độ:

```
1 x_rot = (x - width / 2) * np.cos(np.radians(rotation_angle)) - (y - height / 2)
   * np.sin(np.radians(rotation_angle)) + width / 2
2 y_rot = (x - width / 2) * np.sin(np.radians(rotation_angle)) + (y - height / 2)
   * np.cos(np.radians(rotation_angle)) + height / 2
```

Phương trình của hai hình ellip như sau:

```
1 ellipse1 = (3/4) * ((x_rot - ellipse1_center[0]) / ellipse1_radius_x) ** 2 +
   (1/4) * ((y_rot - ellipse1_center[1]) / ellipse1_radius_y) ** 2 <= 1
2 ellipse2 = (1/4) * ((x_rot - ellipse2_center[0]) / ellipse2_radius_x) ** 2 +
   (3/4) * ((y_rot - ellipse2_center[1]) / ellipse2_radius_y) ** 2 <= 1
```

Sử dụng np.logicalor để thực hiện phép or giữa hai hình ellip:

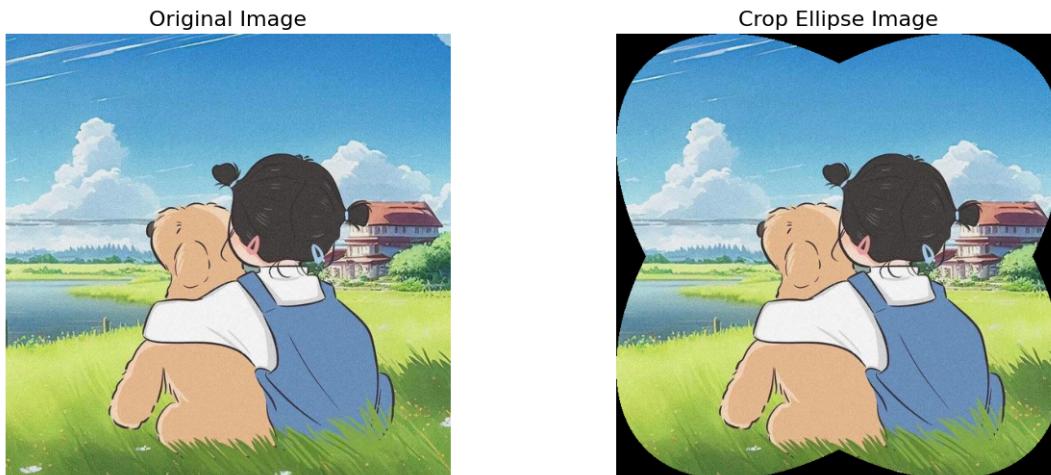
```
1 ellipse = np.logical_or(ellipse1, ellipse2)
```

Cũng giống như cắt ảnh theo khung hình tròn, 'mask' ở đây có hình dạng là 2 hình ellip chéo nhau. Sử dụng mask[:, :, np.newaxis] để giữ lại các điểm nằm trong vùng 2 hình ellip, còn các điểm nằm ngoài sẽ có giá trị 0, làm cho chúng trở thành điểm đen. Kết quả là hình ảnh được cắt theo khung hình tròn.

```
1 mask=ellipse
2 crop=img*mask[:, :, np.newaxis]
3 result=crop.astype(np.uint8)
4 result = Image.fromarray(result)
```

### 9.3 Kết quả

Mặc dù đã thực hiện được kết quả, tuy nhiên bài làm còn hạn chế ở phần các tham số của hình ellip.



Hình 13: Kết quả cắt ảnh với khung là 2 hình ellip chéo nhau

## 10 Hàm main xử lý

Cho phép người dùng nhập vào tên tập tin ảnh mỗi khi hàm main thực hiện thực thi.

Cho phép người dùng lựa chọn chức năng xử lý ảnh (từ 1 đến 7, lựa chọn 0 cho phép thực hiện tất cả chức năng).

- Sử dụng PIL(open(), save()) từ Image để đọc và ghi hình.
- Sử dụng Matplotlib(imshow() từ pyplot) để hiển thị ảnh.

## 11 Tài liệu tham khảo

- [0] Tài liệu thực hành
- [1] [https://www.tutorialspoint.com/dip/grayscale\\_to\\_rgb\\_conversion.htm/](https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.htm/)
- [2] <https://dyclassroom.com/image-processing-project/how-to-convert-a-color-image-into-sepia/>
- [3] [https://en.wikipedia.org/wiki/Kernel\\_\(image\\_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))
- [4] <https://www.codespeedy.com/how-to-crop-an-image-with-rounded-circle-shape-in-python/>
- [5] <https://koodibar.com/posts/xu-ly-hinh-anh-voi-python#3-thay-/>