

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN



ĐỒ ÁN 2 - IMAGE PROCESSING

TOÁN ỨNG DỤNG VÀ THỐNG KÊ CHO CÔNG NGHỆ THÔNG TIN

LỚP 21CLC08

GIẢNG VIÊN: PHAN THỊ PHƯƠNG UYÊN

Nguyễn Hồng Hạnh 21127503

Mục lục

I Các chức năng đã hoàn thành	2
II Ý tưởng thực hiện và mô tả hàm	3
1 Thư viện sử dụng	3
2 Thay đổi độ sáng	3
3 Thay đổi độ tương phản	3
4 Lật ảnh ngang - dọc	3
5 Chuyển đổi ảnh RGB thành ảnh xám (grayscale)/ sepia	4
6 Làm mờ/ sắc nét ảnh	5
7 Cắt ảnh theo kích thước (cắt ở trung tâm)	7
8 Cắt ảnh theo khung tròn	7
9 Cắt ảnh theo khung hai hình ellip chéo nhau	8
III Kết quả	10
IV Tài liệu tham khảo	16

I Các chức năng đã hoàn thành

- Thay đổi độ sáng cho ảnh: 100%
- Thay đổi độ tương phản: 100%
- Lật ảnh ngang - dọc: 100%
- Chuyển đổi ảnh RGB thành ảnh xám (grayscale)/ sepia: 100%
- Làm mờ/ sắc nét ảnh: 100%
- Cắt ảnh theo kích thước (cắt ở trung tâm): 100%
- Cắt ảnh theo khung tròn: 100%
- Cắt ảnh theo khung hai hình ellip chéo nhau: 90%

II Ý tưởng thực hiện và mô tả hàm

1 Thư viện sử dụng

- **NumPy (np):** Thực hiện các thao tác trên ma trận
- **PIL:** Đọc và lưu ảnh
- **matplotlib.pyplot (plt):** hiển thị hình ảnh
- **os:** Kiểm tra và tạo đường dẫn lưu file

2 Thay đổi độ sáng

- **Ý tưởng thực hiện:** để thay đổi độ sáng cho ảnh, ta sẽ cùng thay đổi tất cả các kênh màu của tất cả các điểm ảnh với cùng một lượng β

- **Mô tả hàm:**

```
def brightness(img, beta):  
    new_img = np.clip((img / 255) + (beta / 255), 0, 1)  
    return new_img
```

+ Ta dùng toán tử $+$ để thêm lượng β vào tất cả các điểm ảnh của hình ảnh đầu vào

+ Khi dùng kiểu dữ liệu **uint8**, khi giá trị vượt quá giới hạn 255 sẽ xảy ra hiện tượng tràn số và giá trị đó sẽ được reset về 0, dẫn đến kết quả sau khi thêm một lượng β vào các điểm ảnh cho ra giá trị > 255 sẽ dẫn đến kết quả sai. Vì vậy trước khi thêm vào lượng β , ta sẽ chia giá trị các điểm ảnh cho 255 để chuyển sang kiểu dữ liệu **float**

+ Khi đó, giá trị của các kênh màu sẽ nằm trong đoạn $[0, 1]$, nên ta sẽ dùng hàm **np.clip** để giới hạn lại giá trị của các kênh màu sau khi đã được thêm vào một lượng β

3 Thay đổi độ tương phản

- **Ý tưởng thực hiện:** để thay đổi độ sáng cho ảnh, ta sẽ nhân tất cả các kênh màu của tất cả các điểm ảnh với cùng một giá trị $\alpha > 0$ để thay đổi giá trị cường độ sáng của từng kênh ảnh, từ đó thay đổi độ tương phản của ảnh

- **Mô tả hàm:**

```
def contrast(img, alpha):  
    new_img = np.clip(img * alpha / 255, 0, 1)  
    return new_img
```

+ Tương tự như thay đổi độ sáng cho ảnh, trước khi nhân các kênh màu với α , ta sẽ chia giá trị các điểm ảnh cho 255 để chuyển sang kiểu dữ liệu **float** và dùng hàm **np.clip** để giới hạn lại giá trị của các kênh màu trong đoạn $[0, 1]$ sau khi đã được nhân với α

4 Lật ảnh ngang - dọc

- **Ý tưởng thực hiện:** để lật ảnh ngang/ dọc, ta sẽ đảo vị trí của các cột/ dòng trong mảng các điểm ảnh của hình ảnh đầu vào

- **Mô tả hàm:**

```
def flip(img, direction):  
    new_img = np.flip(img, axis=direction)  
    return new_img
```

+ Để lật mảng các điểm ảnh, ta sử dụng hàm **np.flip**: nếu muốn lật ảnh ngang, ta sẽ lật theo axis=1 và nếu muốn lật ảnh dọc, ta sẽ lật theo axis=0

5 Chuyển đổi ảnh RGB thành ảnh xám (grayscale)/ sepia

Grayscale

- **Ý tưởng thực hiện:** với mỗi điểm ảnh có ba kênh màu red, green, blue, ta sẽ thay thế điểm ảnh đó bằng một giá trị màu xám $\text{gray} = 0.3 \times \text{red} + 0.59 \times \text{green} + 0.11 \times \text{blue}$

- Mô tả hàm:

```
def grayscale(img):
    luma_scalars = np.array([0.3, 0.59, 0.11])
    new_img = np.dot(img / 255, luma_scalars)
    return new_img
```

+ Để nhân từng kênh màu với hệ số tương ứng và lấy tổng là giá trị kênh xám, ta tạo mảng một chiều chứa 3 hệ số ở vị trí tương ứng với vị trí của kênh màu trong điểm ảnh (red, green, blue), sau đó dùng hàm **np.dot** với lần lượt mảng điểm ảnh của hình ảnh đầu vào và mảng hệ số

```
A = [[a00, a01, a02],
     [a10, a11, a12],
     [a20, a21, a22]]

B = [b0, b1, b2]

np.dot(A, B) = [a00 * b0 + a01 * b1 + a02 * b2,
                 a10 * b0 + a11 * b1 + a12 * b2,
                 a20 * b0 + a21 * b1 + a22 * b2]
```

Sepia

- **Ý tưởng thực hiện:** với mỗi điểm ảnh có ba kênh màu red, green, blue, ta sẽ thay thế từng kênh màu bằng các giá trị `new_red`, `new_green`, `new_blue` với:

- $\text{new_red} = 0.393 \times \text{red} + 0.769 \times \text{green} + 0.189 \times \text{blue}$
- $\text{new_green} = 0.349 \times \text{red} + 0.686 \times \text{green} + 0.168 \times \text{blue}$
- $\text{new_blue} = 0.272 \times \text{red} + 0.534 \times \text{green} + 0.131 \times \text{blue}$

- Mô tả hàm:

```
def sepia(img):
    new_rgb_scalars = np.array([[0.393, 0.349, 0.272], [0.769, 0.686, 0.534], [0.189, 0.168, 0.131]])
    new_img = np.clip(np.matmul(img / 255, new_rgb_scalars), 0, 1)
    return new_img
```

+ Để thay thế từng kênh ảnh bằng giá trị mới là tổng của các kênh màu nhân cho giá trị tương ứng, trước hết ta tạo mảng hai chiều với mỗi cột là nhóm các giá trị hệ số cùng tạo nên một giá trị mới của kênh màu và nằm ở vị trí tương ứng với vị trí của kênh màu red, green, blue

```
[[red_new_red, red_new_green, red_new_blue],
 [green_new_red, green_new_green, green_new_blue],
 [blue_new_red, blue_new_green, blue_new_blue]]
```

+ Sau đó ta dùng hàm **np.matmul** nhân ma trận các điểm ảnh của hình ảnh đầu vào và ma trận các hệ số

+ Ở phần này ta sẽ chuyển hệ điểm ảnh sang kiểu dữ liệu **float** và dùng **np.clip** để giới hạn giá trị kênh màu mới nằm trong đoạn $[0, 1]$

6 Làm mờ/ sắc nét ảnh

Làm mờ ảnh (Thuật toán Box Blur)

- **Ý tưởng thực hiện:**

- Với mỗi điểm ảnh, ta sẽ lấy trung bình giá trị của các điểm ảnh trong vùng lân cận để tạo ra giá trị điểm ảnh mới
- Ta sẽ sử dụng một ma trận vuông, gọi là kernel, với kích thước k được người dùng nhập vào ($k = 3, 5, 7, \dots$), cũng chính là kích thước vùng lân cận ta tiến hành lấy giá trị điểm ảnh trung bình. Kích thước kernel lớn sẽ dẫn đến hiệu quả làm mờ mạnh hơn, trong khi kích thước nhỏ sẽ có tác động nhẹ hơn. Ta sẽ sử dụng ma trận vuông có kích thước k và có giá trị các phần tử là $\frac{1}{k \times k}$
- Sau đó, ta sẽ tiến hành tích chập: đặt vị trí trung tâm của kernel lên mỗi điểm ảnh cần cập nhật giá trị, sau đó tính trung bình bằng cách nhân các phần tử kernel tương ứng với các giá trị điểm ảnh và cộng chúng lại, ta sẽ có giá trị điểm ảnh mới cần tìm
- Có thể thấy, ta không thể đặt vị trí trung tâm của kernel lên các điểm ảnh nằm ở ngoài biên, vì vậy trước khi tích chập ta sẽ tiến hành thêm biên ở xung quanh mảng các điểm ảnh với giá trị bằng 0 (zero-padding)

- **Mô tả hàm:**

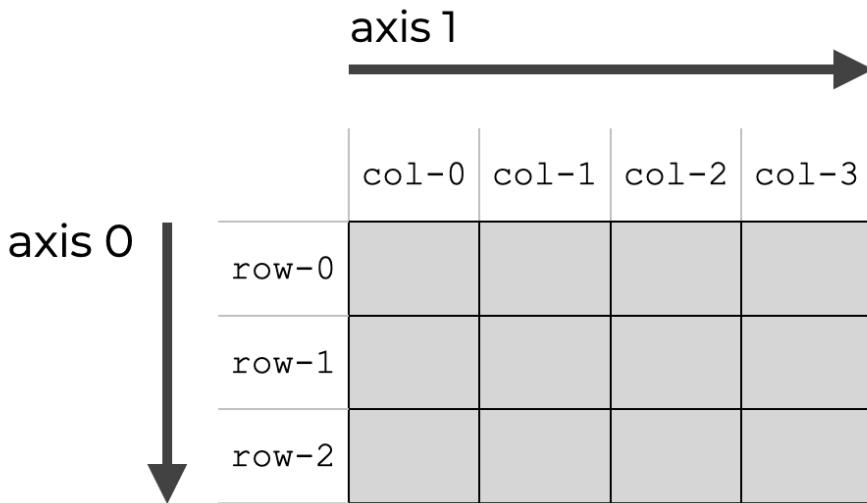
```
def padding(img, size):  
    height, width, channels = img.shape  
    img_padding = np.zeros((height + 2 * size, width + 2 * size, channels))  
    img_padding[size : height + size, size : width + size] = img  
    return img_padding  
  
def blur(img, k):  
    img_padding = padding(img, k // 2)  
    row, col = img_padding.shape[:2]  
    new_img = []  
    for i in range(row - k + 1):  
        for j in range(col - k + 1):  
            new_img.append(np.sum(img_padding[i:i+k, j:j+k], axis=(0, 1)) / (k * k * 1.0))  
    new_img = np.array(new_img, dtype=np.uint8)  
    new_img = np.reshape(new_img, img.shape)  
    return new_img
```

+ Ta tiến hành thêm biên cho hình ảnh bằng hàm **padding**. Hàm sẽ tạo một mảng số 0 có kích thước là chiều cao, chiều rộng và cộng thêm các biên ở xung quanh, sau đó thêm dữ liệu từ mảng hình ảnh đầu vào vào vị trí tương ứng.

+ Sau khi đã thêm biên cho ảnh, ta tiến hành duyệt từng điểm ảnh và tính trung bình các giá trị trong vùng $k \times k$. Ví dụ, ta có một mảng ba chiều gồm các điểm ảnh như sau:

```
[[[1, 2, 3], [4, 5, 6], [7, 8, 9]],  
 [[10, 11, 12], [13, 14, 15], [16, 17, 18]],  
 [[19, 20, 21], [22, 23, 24], [25, 26, 27]]]
```

+ Ta sẽ dùng hàm **np.sum** với $axis = 0$ và $axis = 1$ để lần lượt cộng các giá trị theo trực tương ứng và trả về mảng một chiều có ba phần tử là tổng của các phần tử nằm ở vị trí tương ứng



```

$$\begin{aligned} & [(1 + 10 + 19) + (4 + 13 + 22) + (7 + 16 + 25) \\ & (2 + 11 + 20) + (5 + 14 + 23) + (8 + 17 + 26) \\ & (3 + 12 + 21) + (6 + 15 + 24) + (9 + 18 + 27)] \\ = & [117 \ 126 \ 135] \end{aligned}$$

```

- + Sau đó ta sẽ chia tổng cho $k \times k$ để lấy giá trị trung bình và lưu vào list new_img
- + Sau khi hoàn thành tích chập, ta chuyển danh sách new_img thành một mảng NumPy với kiểu dữ liệu **uint8** và định hình lại mảng cùng hình dạng với hình ảnh đầu vào bằng hàm **np.reshape**

Làm sắc nét ảnh

- **Ý tưởng thực hiện:** để làm sắc nét ảnh, ta tăng độ tương phản giữa các điểm ảnh lân cận để làm nổi bật các biên và chi tiết, cụ thể hơn ta sẽ dùng kernel $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ để tích chập với mảng các điểm ảnh của hình ảnh đầu vào

Mô tả hàm:

```
def sharpen(img):
    kernel = np.array([[0, -1, 0],
                      [-1, 5, -1],
                      [0, -1, 0]])
    k = 3
    img_padding = padding(img, k // 2)
    row, col = img_padding.shape[:2]
    new_img = []
    for i in range(row - k + 1):
        for j in range(col - k + 1):
            new_img.append(
                np.clip(np.sum(
                    (np.reshape(img_padding[i:i+k, j:j+k], (k * k, img_padding.shape[2])) *
                     np.reshape(kernel, (k * k, 1))), axis=0), 0, 255))
    new_img = np.array(new_img, dtype=np.uint8)
    new_img = np.reshape(new_img, img.shape)
    return new_img
```

- + Sau khi đã thêm biên cho hình ảnh, ta tiến hành duyệt từng điểm ảnh và tính trung bình các giá trị trong vùng $k \times k$
- + Ta dùng hàm **np.shape** định dạng lại vùng điểm ảnh thành một dãy các điểm ảnh và kernel thành một dãy các hệ số, sau đó tiến hành nhân từng phần tử rồi dùng hàm **np.sum** để tính tổng và chia tổng

cho $k \times k$ để lấy giá trị trung bình các điểm ảnh của vùng đó. Ta kết hợp thêm hàm `np.clip` để giới hạn các giá trị điểm ảnh trong đoạn [0, 1]

+ Sau khi tất cả giá trị điểm ảnh trung bình đã được lưu vào list `new_img`, ta sẽ chuyển danh sách `new_img` thành một mảng NumPy với kiểu dữ liệu `uint8` và định hình lại mảng cùng hình dạng với hình ảnh đầu vào bằng hàm `np.reshape`

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

- **Ý tưởng thực hiện:** ta sẽ lấy những điểm ảnh nằm trong kích thước được truyền vào tính từ trung tâm

- Mô tả hàm:

```
def center_crop(img, crop_height, crop_width):
    height, width = img.shape[:2]
    center = [height // 2, width // 2]
    new_img = img[center[0] - (crop_height//2):center[0] + (crop_height // 2), center[1] - (crop_width//2):center[1] + (crop_width // 2)]
    return new_img
```

+ Để lấy những điểm ảnh nằm trong vùng kích thước được cho, ta sẽ dùng index slicing và xác định điểm ảnh bắt đầu và kết thúc từ vị trí trung tâm ảnh

8 Cắt ảnh theo khung tròn

- **Ý tưởng thực hiện:** ta sẽ tạo một mặt nạ hình tròn và áp mặt nạ này lên mảng ảnh gốc để cắt ảnh theo hình tròn

- Mô tả hàm:

```
def generate_circle_array(array_size):
    circle_array = np.zeros((array_size, array_size), dtype=np.uint8)

    center = array_size // 2
    radius = center

    y, x = np.ogrid[-center:array_size-center, -center:array_size-center]
    mask = x**2 + y**2 <= radius**2
    circle_array[mask] = 1

    return circle_array

def circle_drop(img):
    size = img.shape[0] if img.shape[0] <= img.shape[1] else img.shape[1]
    circle_array = generate_circle_array(img.shape[0])
    circle_mask = circle_array.astype(bool)
    red_channel, green_channel, blue_channel = img[:, :, 0], img[:, :, 1], img[:, :, 2]

    crop_red_channel = np.zeros_like(red_channel)
    crop_red_channel[circle_mask] = red_channel[circle_mask]

    crop_green_channel = np.zeros_like(green_channel)
    crop_green_channel[circle_mask] = green_channel[circle_mask]

    crop_blue_channel = np.zeros_like(blue_channel)
    crop_blue_channel[circle_mask] = blue_channel[circle_mask]

    new_img = np.stack([crop_red_channel, crop_green_channel, crop_blue_channel], axis=-1)
    return new_img
```

+ Ta tạo một mảng hai chiều đại diện cho hình tròn bằng cách tạo một mảng trống với kích thước được truyền vào và đặt giá trị của các điểm ảnh nằm trong hình tròn thành 1. Các điểm nằm trong hình tròn sẽ được xác định bằng công thức: $x^2 + y^2 \leq r^2$

+ Từ mảng đại diện cho hình tròn, ta chuyển đổi mảng thành một mảng boolean, trong đó các giá trị khác 0 được chuyển thành True, còn các giá trị bằng 0 được chuyển thành False, tạo thành một mặt nạ hình tròn, với các điểm ảnh nằm trong hình tròn được đánh dấu là True, các điểm ảnh nằm ngoài hình tròn được đánh dấu là False

+ Sau đó ta sẽ sao chép từng kênh màu của hình ảnh đầu và chỉ giữ lại các điểm ảnh nằm trong vùng hình tròn theo mặt nạ hình tròn. Các điểm ảnh nằm ngoài hình tròn được đặt thành 0

+ Cuối cùng, ta kết hợp các kênh màu mới để tạo ra hình ảnh giống hình ảnh ban đầu nhưng chỉ giữ lại các điểm ảnh nằm trong hình tròn

9 Cắt ảnh theo khung hai hình ellip chéo nhau

- **Ý tưởng thực hiện:** tương tự như cắt hình ảnh theo hình tròn, ta sẽ tạo một hình ellip chéo, kết hợp với hình ảnh lật theo chiều ngang của nó để tạo thành mặt nạ hình 2 ellip chéo nhau, và áp mặt nạ này lên mảng ảnh gốc để cắt ảnh theo hai hình ellip chéo nhau

- Mô tả hàm:

```
def generate_ellipse_array(array_size):
    size = int(np.ceil(array_size * 5/4))
    ellipse_array = np.zeros((size, size), dtype=np.uint8)

    center = size // 2
    radius = center

    a = (size / 2)**2
    b = int(np.ceil(array_size * np.sqrt(2) / 4))**2

    y, x = np.ogrid[-center:size-center, -center:size-center]
    mask = (x**2/a) + (y**2/b) <= 1
    ellipse_array[mask] = 1

    height, width = ellipse_array.shape
    diagonal_length = int(np.ceil(np.sqrt(height**2 + width**2)))

    rotated_arr = np.zeros((height, width), dtype=ellipse_array.dtype)

    center_x, center_y = width // 2, height // 2
    cos45, sin45 = np.cos(np.pi/4), np.sin(np.pi/4)

    for y in range(height):
        for x in range(width):
            x_rot = int((x - center_x) * cos45 + (y - center_y) * sin45 + center_x)
            y_rot = int((y - center_y) * cos45 - (x - center_x) * sin45 + center_y)
            if 0 <= x_rot < width and 0 <= y_rot < height:
                rotated_arr[y_rot, x_rot] = ellipse_array[y, x]

    return center_crop(rotated_arr, array_size, array_size)

def ellipse_drop(img):
    ellipse_array = generate_ellipse_array(img.shape[0]) + flip(generate_ellipse_array(img.
        shape[0]), 0)
    ellipse_mask = ellipse_array.astype(bool)
    red_channel, green_channel, blue_channel = img[:, :, 0], img[:, :, 1], img[:, :, 2]

    crop_red_channel = np.zeros_like(red_channel)
    crop_red_channel[ellipse_mask] = red_channel[ellipse_mask]

    crop_green_channel = np.zeros_like(green_channel)
    crop_green_channel[ellipse_mask] = green_channel[ellipse_mask]

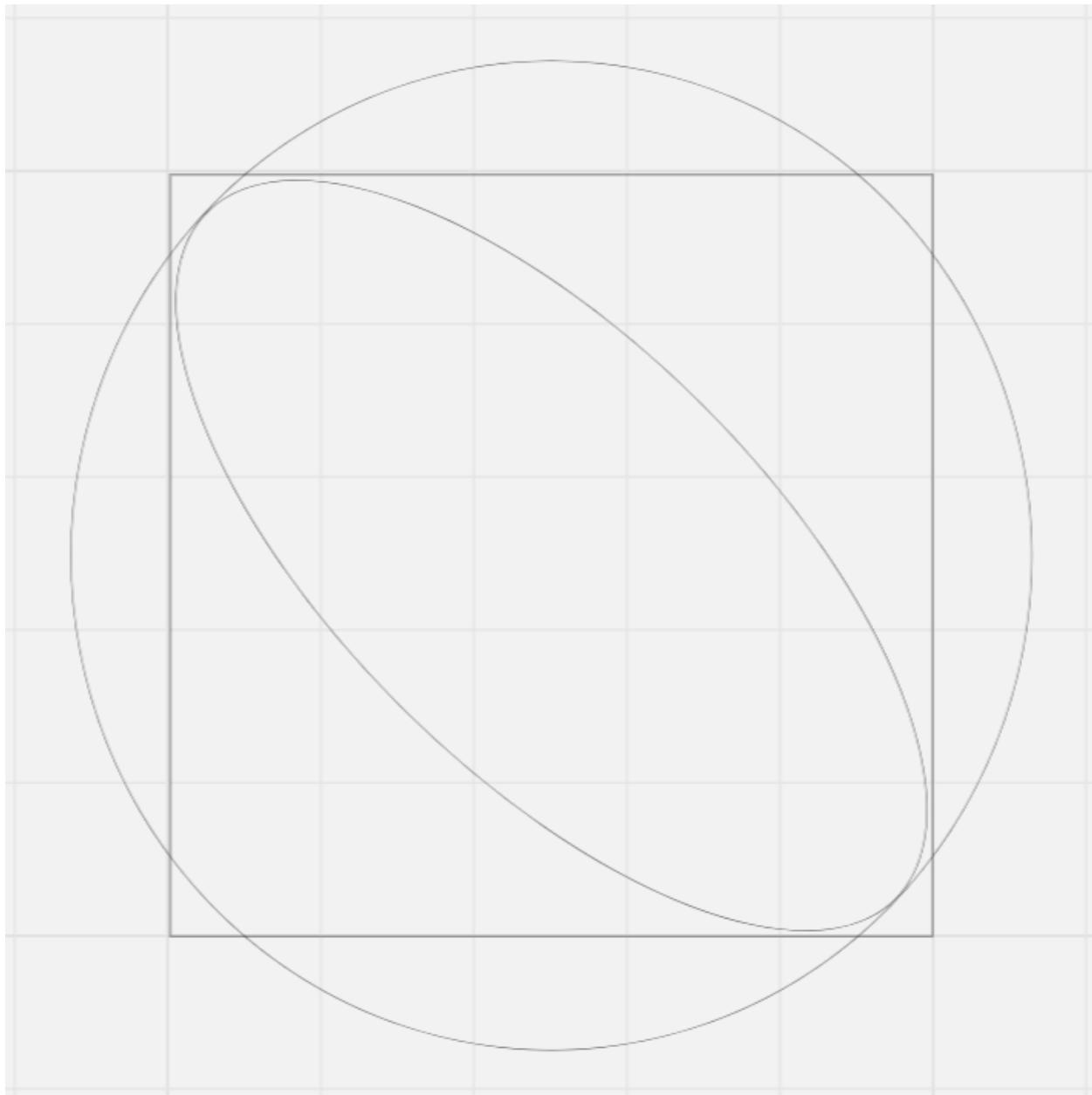
    crop_blue_channel = np.zeros_like(blue_channel)
    crop_blue_channel[ellipse_mask] = blue_channel[ellipse_mask]

    new_img = np.stack([crop_red_channel, crop_green_channel, crop_blue_channel], axis=-1)
    return new_img
```

+ Nhìn hình ta thấy kích thước hai trục của hình ellip nằm chéo trong hình vuông có cạnh là a có thể tính như sau:

- Trục chính $A = \sqrt{(\frac{a}{2})^2 + (\frac{3a}{8})^2} = \frac{5a}{8}$

- Trục phụ $B = \frac{a\sqrt{2}}{4}$



+ Các điểm nằm trong hình ellip chéo sẽ được xác định bằng công thức: $(\frac{x}{A})^2 + (\frac{y}{B})^2 \leq 1$

+ Sau khi tạo được một hình ellip nằm chéo, ta dùng hàm **flip** để lật hình lại theo chiều ngang và kết hợp 2 hình để có được khung hình như yêu cầu, sau đó, tương tự như cắt hình theo khung hình tròn, ta sao chép từng kênh màu của hình ảnh đầu và chỉ giữ lại các điểm ảnh nằm trong vùng hình hai ellip chéo nhau theo mặt nạ và kết hợp các kênh màu mới lại

- Tuy nhiên, khi tạo khung hình hai ellip chéo nhau, vẫn còn bị lỗi ở phần đặt màu cho khung hình nên hình ảnh ra bị hiện các đường chéo và không giữ được màu gốc

III Kết quả

Hình 1

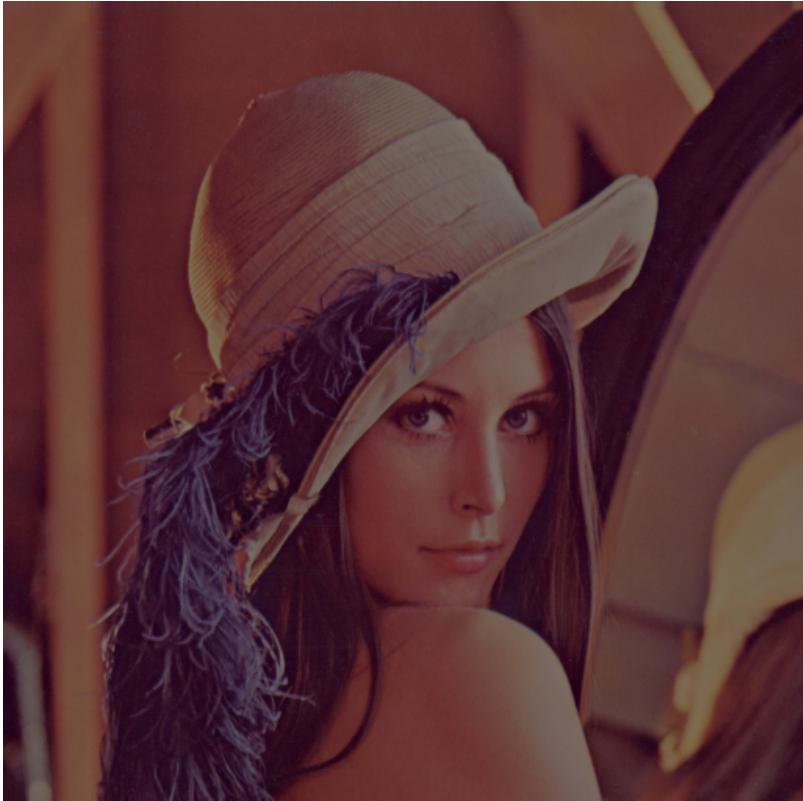
Hình gốc



Thay đổi độ sáng



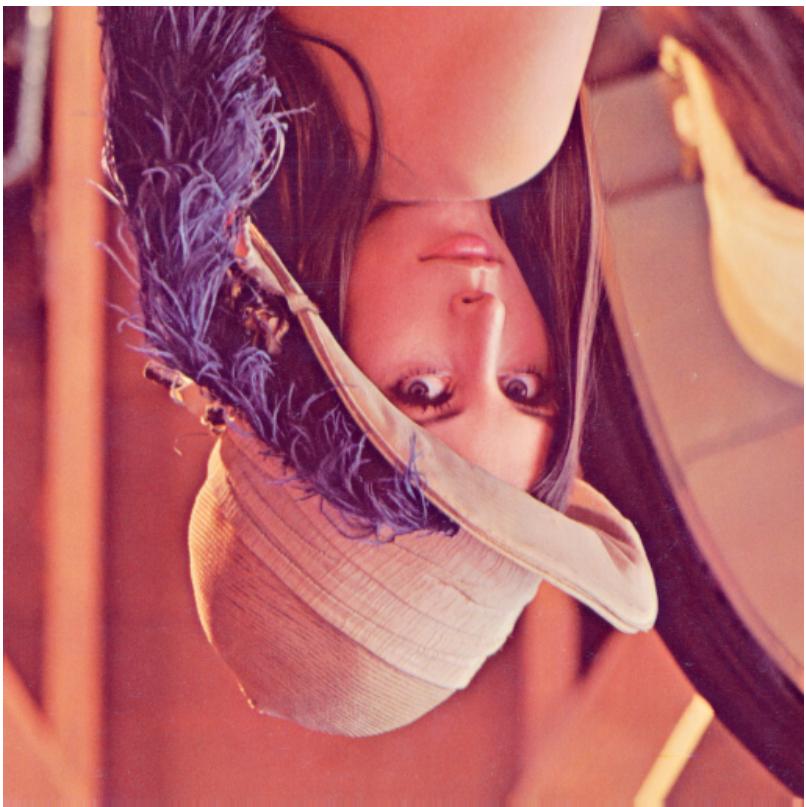
Thay độ đậm tương phản



Lật ảnh ngang



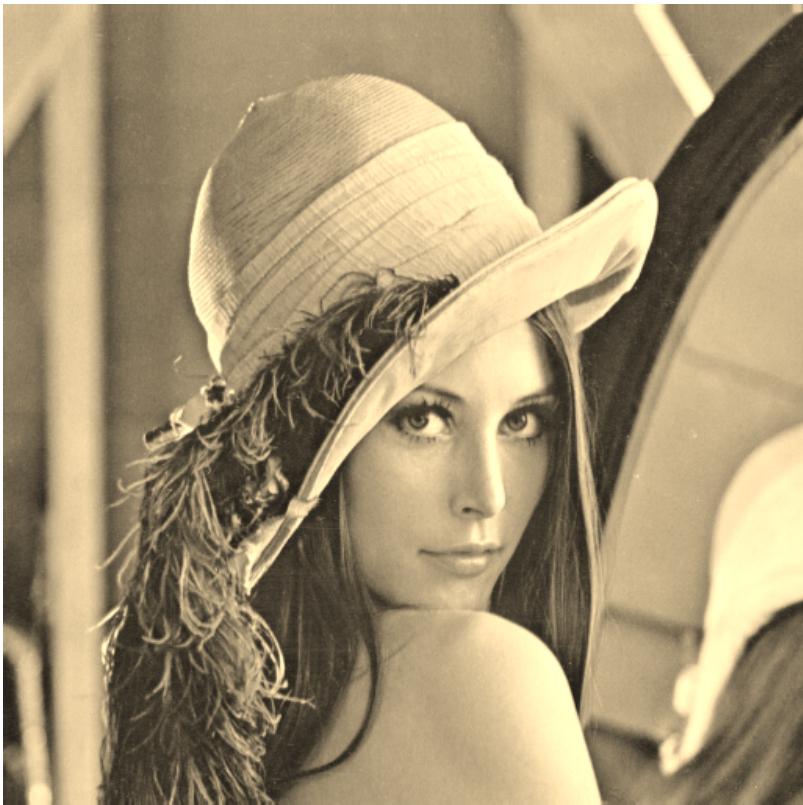
Lật ảnh dọc



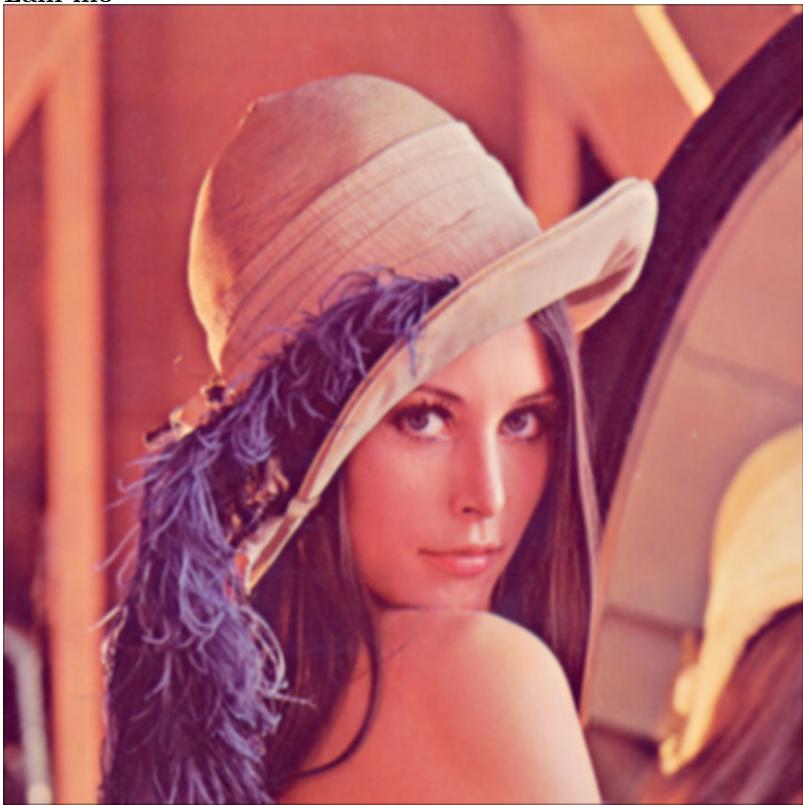
Grayscale



Sepia



Làm mờ



Làm sắc nét



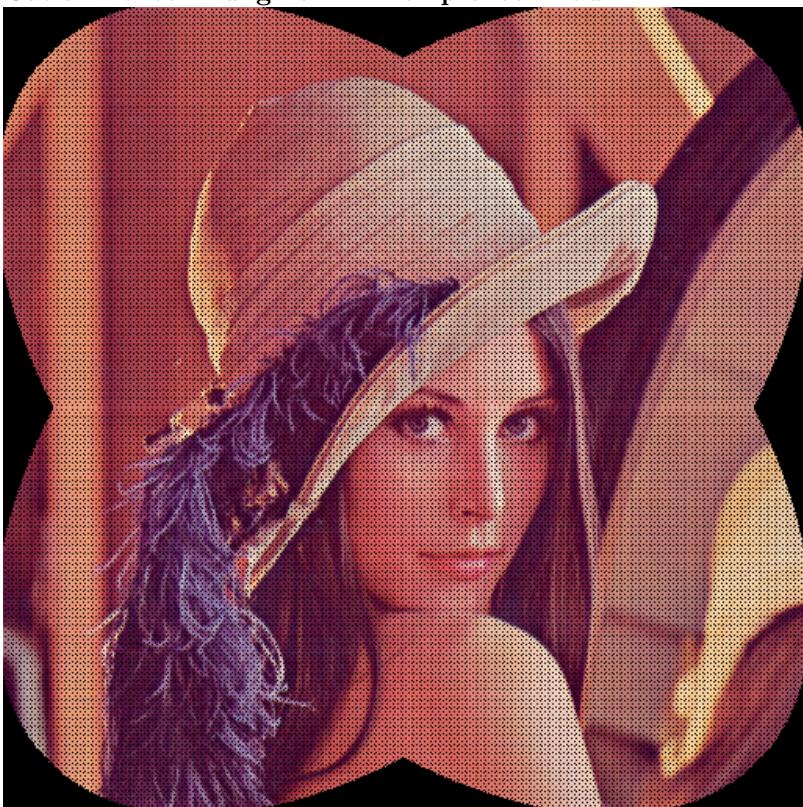
Cắt ảnh theo kích thước (cắt ở trung tâm)



Cắt ảnh theo khung tròn



Cắt ảnh theo khung hai hình ellip chéo nhau



IV Tài liệu tham khảo

1. [https://en.wikipedia.org/wiki/Kernel_\(image_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing)) - Box Blur and Sharpen Kernel
2. <https://numpy.org/> - Các hàm của NumPy