

ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BỘ MÔN CÔNG NGHỆ TRI THỨC



Báo cáo Đồ án Thực hành

Đề tài: Image Processing

Môn học: Toán Ứng dụng và Thống kê cho Công nghệ Thông tin

Sinh viên thực hiện:

Trà, Hoàng Anh (21127453)

Giáo viên hướng dẫn:

ThS. Vũ Quốc Hoàng

ThS. Nguyễn Văn Quang Huy

CN. Ngô Dình Hy

ThS. Phan Thị Phương Uyên

Ngày 30 tháng 7 năm 2023

LỜI MỞ ĐẦU

Xử lý ảnh là một lĩnh vực của khoa học máy tính liên quan đến việc xử lý và phân tích các hình ảnh kỹ thuật số. Các kỹ thuật xử lý ảnh được sử dụng để cải thiện chất lượng của hình ảnh, loại bỏ nhiễu, thay đổi độ sáng, độ tương phản, màu sắc, kích thước và các thuộc tính khác của hình ảnh.

Trong báo cáo này, em sẽ trình bày một số kỹ thuật xử lý ảnh cơ bản, bao gồm:

- Thay đổi độ sáng cho ảnh
- Thay đổi độ tương phản cho ảnh
- Lật ảnh (ngang hoặc dọc)
- Chuyển đổi ảnh RGB sang ảnh xám hoặc ảnh sepia
- Làm mờ ảnh
- Làm sắc nét ảnh
- Cắt ảnh ở trung tâm, theo khung tròn, khung elip.

Em sẽ sử dụng các kỹ thuật này để xử lý một số hình ảnh mẫu và thảo luận về kết quả của việc áp dụng các kỹ thuật này.

Do thời gian và trình độ chuyên môn còn hạn chế nên đồ án này không thể tránh khỏi những thiếu sót. Em rất mong nhận được nhiều ý kiến đóng góp từ các thầy, cô giáo và bạn bè để đồ án ngày càng hoàn thiện hơn!

Sinh viên thực hiện:
Trà, Hoàng Anh (21127453)

Mục lục

1 Ý tưởng thực hiện	3
1.1 Thay đổi độ sáng cho ảnh	3
1.2 Thay đổi độ tương phản của ảnh	3
1.3 Lật ảnh (ngang hoặc dọc)	3
1.4 Chuyển đổi ảnh RGB sang ảnh xám hoặc sepia	3
1.5 Làm mờ ảnh hoặc làm sắc nét ảnh	4
1.6 Cắt ảnh theo kích thước ở trung tâm	5
1.7 Cắt ảnh theo khung hình tròn hoặc khung elip	5
2 Mô tả hàm và thuật toán	6
2.1 Các thư viện sử dụng	6
2.2 Các hàm hỗ trợ	6
2.2.1 Hàm đọc ảnh <code>readImg()</code>	6
2.2.2 Hàm biến đổi <code>reshape_img()</code>	6
2.3 Các hàm thực hiện	6
2.3.1 Hàm thay đổi độ sáng <code>adjust_brightness()</code>	6
2.3.2 Hàm thay đổi độ tương phản <code>adjust_contrast()</code>	6
2.3.3 Hàm lật ảnh <code>flip_image()</code>	7
2.3.4 Hàm khởi tạo ma trận Kernel làm mờ ảnh <code>genKernel()</code>	7
2.3.5 Hàm nhân 2 ma trận (Element-wise) <code>convolve()</code>	7
2.3.6 Hàm thực hiện quá trình convolution cho toàn ma trận <code>convolution()</code>	8
2.3.7 Hàm làm mờ ảnh <code>adjust_blur()</code>	8
2.3.8 Hàm khởi tạo ma trận Kernel làm sắc nét ảnh <code>sharpenKernel()</code>	8
2.3.9 Hàm làm sắc nét ảnh <code>adjust_sharpen()</code>	8
2.3.10 Hàm biến đổi ảnh màu thành ảnh xám <code>to_grayscale()</code>	9
2.3.11 Hàm biến đổi ảnh màu thành ảnh sepia <code>to_sepia()</code>	9
2.3.12 Hàm cắt ảnh ở trung tâm <code>center_crop()</code>	9
2.3.13 Hàm cắt ảnh theo khung tròn <code>circle_mask()</code>	10
2.3.14 Hàm cắt ảnh theo khung elip <code>elip_mask()</code>	10
2.4 Hàm vận hành	10
2.4.1 Các hàm kiểm tra khả năng vận hành	10
2.4.2 Hàm vận hành và xuất kết quả <code>showResult()</code>	11
3 Kết quả thực hiện	11
4 Nhận xét	16
4.1 Nhận xét chung	16
4.2 Hạn chế nói chung	16
Tài liệu tham khảo	17

1 Ý tưởng thực hiện

1.1 Thay đổi độ sáng cho ảnh

- Về lý thuyết, khi các giá trị điểm ảnh càng về 255 thì ảnh đó sẽ càng sáng lên. Do đó, ta sử dụng **phép toán cộng ma trận với một scalar** để gia tăng giá trị của điểm ảnh.[8]

Như vậy, với mỗi vector màu $v = [v_1, v_2, v_3]$ thì muốn tăng độ sáng thì phải cộng thêm một lượng α . Khi đó vector màu là $v_1 = [v_1 + \alpha, v_2 + \alpha, v_3 + \alpha]$.

- Tuy nhiên, vẫn sẽ phải đảm bảo sau khi thực hiện phép toán trên, giá trị điểm ảnh không nhỏ hơn 0 và không vượt quá 255.

1.2 Thay đổi độ tương phản của ảnh

- Khi khoảng cách giữa các giá trị điểm ảnh càng về lớn thì ảnh đó sẽ càng tương phản nhau. Do đó, ta sử dụng **phép toán nhân ma trận với một scalar > 1** để gia tăng độ tương phản của ảnh.[8]

Như vậy, với mỗi vector màu $v = [v_1, v_2, v_3]$ thì muốn tăng độ tương phản thì phải nhân thêm một lượng $\alpha > 1$. Khi đó vector màu là $v_1 = [v_1.\alpha, v_2.\alpha, v_3.\alpha]$.

- Tương tự, ta vẫn sẽ phải đảm bảo sau khi thực hiện phép toán trên, giá trị điểm ảnh không nhỏ hơn 0 và không vượt quá 255.

1.3 Lật ảnh (ngang hoặc dọc)

- Ý tưởng của việc lật ảnh khá đơn giản. Đó là đảo ngược (revert) vector dòng hoặc cột tùy theo việc lật ảnh ngang hay dọc. [8]

Ma trận gốc \rightarrow Ma trận đã lật dọc \rightarrow Ma trận đã lật ngang

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 2 & 1 \\ 6 & 5 & 4 \\ 9 & 8 & 7 \end{bmatrix} \rightarrow \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

1.4 Chuyển đổi ảnh RGB sang ảnh xám hoặc sepia

- Có 2 phương pháp biến đổi ảnh màu thành ảnh xám. Ở đây, ta sử dụng phương pháp "Thay đổi sự đóng góp của 3 kênh màu RGB thành một kênh màu (xám)". Công thức biến đổi sẽ là: $Gray = 0.2989.Red + 0.5870.Green + 0.1140.Blue$ [6]

Muốn thể hiện màu xám ở cmap RGB, ta có thể gán cả ba giá trị ở ba kênh màu thành cùng một giá trị Gray được tính theo công thức ở trên.

- Để biến đổi một ảnh màu thành ảnh sepia, ta cũng sử dụng công thức biến đổi theo kênh màu [7]:

- Red = $0.393.Red + 0.769.Green + 0.189.Blue$
- Green = $0.349.Red + 0.686.Green + 0.168.Blue$
- Blue = $0.272.Red + 0.534.Green + 0.131.Blue$

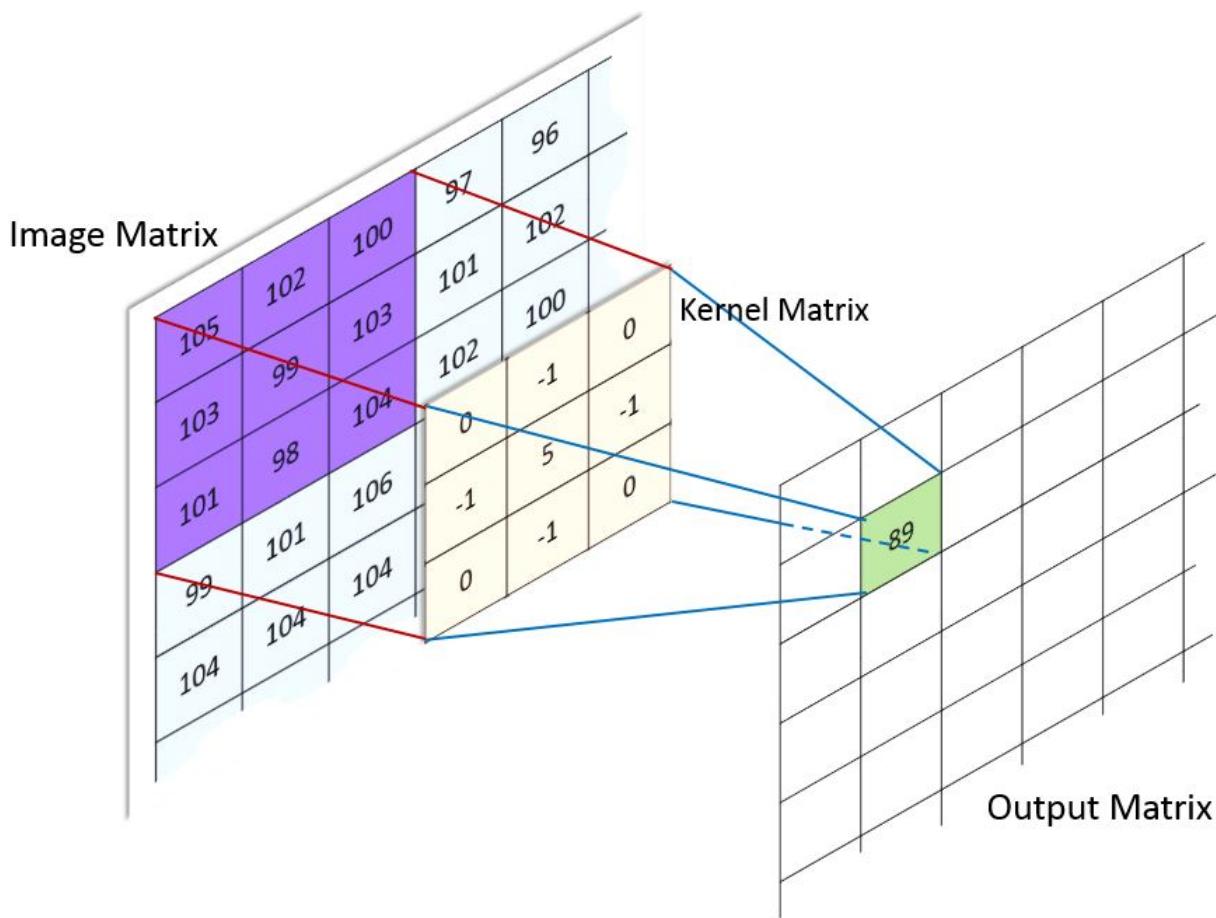
1.5 Làm mờ ảnh hoặc làm sắc nét ảnh

Ý tưởng của việc làm mờ ảnh hay làm sắc nét ảnh là tương đồng với nhau. [1] [8] [9]

- Đầu tiên, tạo ra một ma trận kernel vuông cấp n (với n lẻ). Ma trận này chính là điểm khác biệt giữa việc làm mờ hay làm sắc nét ảnh.

Với $n = 3$, kernel làm mờ ảnh sẽ là: $\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ còn nếu muốn làm sắc nét ảnh thì kernel sẽ là: $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$

- Tiếp theo, xét từng vector màu, cộng thêm từng thành phần của hình ảnh vào các hàng xóm cục bộ của nó, được tính theo trọng số của kernel (convolve).



Hình 1: Minh họa cho quá trình Convolution

- Cuối cùng, sau khi đi qua hết tất cả các vector màu, ta sẽ nhận được một ma trận mới thỏa mãn yêu cầu làm mờ hay làm sắc nét so với ma trận ban đầu.

1.6 Cắt ảnh theo kích thước ở trung tâm

- Ý tưởng chính là sử dụng kỹ thuật indexing.
- Khi nhận được giá trị kích thước sau khi cắt (không lớn hơn kích thước ban đầu), ta sẽ tính toán điểm bắt đầu và điểm kết thúc.
- Lần lượt duyệt qua các vector màu nằm trong khoảng đã tính toán và thêm các giá trị đó vào ma trận mới.
- Ta sẽ nhận được một ma trận kết quả có kích thước theo yêu cầu và vị trí ở trung tâm.

1.7 Cắt ảnh theo khung hình tròn hoặc khung elip

Ý tưởng chính là sử dụng mask và boolean array [3].

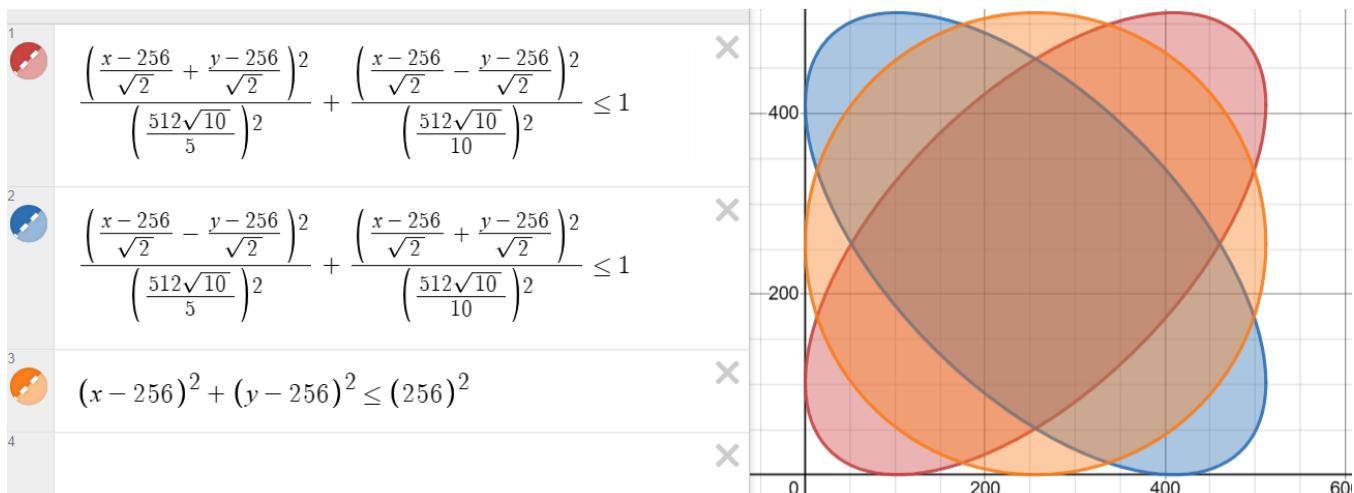
Đối với khung là hình tròn thì với r là một nửa kích thước ảnh (vuông) và (x_0, y_0) là điểm trung tâm của ảnh, ta có thể biểu diễn hình tròn bán kính r , tâm (x_0, y_0) bằng hàm số:

$$(x - x_0)^2 + (y - y_0)^2 \leq r^2$$

Còn đối với khung là 2 hình elip, tương tự ta sử dụng công thức tổng quát cho hình elip và tính toán giá trị lớn nhất sao cho 2 hình elip nằm xiên góc α vừa vặn trong khung hình vuông [4] [5]:

$$\frac{[(x - r) \cdot \cos \alpha + (y - r) \cdot \sin \alpha]^2}{a^2} + \frac{[(x - r) \cdot \sin \alpha - (y - r) \cdot \cos \alpha]^2}{b^2} \leq 1$$

Với a, b thỏa mãn $4.r^2 = a^2 \cdot \sin^2 \alpha + b^2 \cdot \cos^2 \alpha$



Hình 2: Minh họa cho hàm số biểu diễn khung tròn và 2 khung elip (Desmos)

Sử dụng kỹ thuật mask và boolean array, với những điểm nằm ngoài khoảng trên thì gán nó là màu đen.

2 Mô tả hàm và thuật toán

2.1 Các thư viện sử dụng

- PIL: Đọc và ghi ảnh.
- NumPy: Tính toán ma trận.
- matplotlib: Hiển thị hình ảnh.

2.2 Các hàm hỗ trợ

2.2.1 Hàm đọc ảnh readImg()

Input: img_path là đường dẫn tới tập tin ảnh

Sử dụng phương thức `Image.open()` trong thư viện PIL để mở ảnh.

2.2.2 Hàm biến đổi reshape_img()

Input: raw_img là ảnh sau khi mở bằng hàm `readImg()`

Biến đổi bức ảnh thành 1 ma trận mà mỗi hàng là 1 vector màu với 3 giá trị màu bằng `shape()` và `reshape()`

2.3 Các hàm thực hiện

2.3.1 Hàm thay đổi độ sáng adjust_brightness()

Input:

- test_img: ma trận của hình ảnh gốc
- alpha: giá trị thay đổi, mặc định là 50

Output:

- Ma trận kết quả sau khi cộng ma trận gốc với giá trị alpha

Thực hiện:

1. Cộng ma trận `test_img` với một số `alpha`.
2. Sử dụng phương thức `clip()` của thư viện Numpy [2] để giới hạn giá trị trong đoạn từ 0 đến 255.

2.3.2 Hàm thay đổi độ tương phản adjust_contrast()

Input:

- test_img: ma trận của hình ảnh gốc
- alpha: giá trị thay đổi, mặc định là 1.38

Output:

- Ma trận kết quả sau khi nhân ma trận gốc với giá trị alpha

Thực hiện:

1. Nhân ma trận `test_img` với một số alpha.
2. Sử dụng phương thức `clip()` của thư viện Numpy [2] để giới hạn giá trị trong đoạn từ 0 đến 255.

2.3.3 Hàm lật ảnh `flip_image()`

Input:

- `test_img`: ma trận của hình ảnh gốc
- `direction`: chiều lật ảnh

Output:

- Ma trận kết quả sau khi lật

Thực hiện:

1. Nếu chiều lật là vertical, thì sử dụng phương thức `flipud()` trong thư viện Numpy [2] để lật.
2. Nếu chiều lật là horizontal, thì sử dụng phương thức `fliplr()` trong thư viện Numpy [2] để lật.

2.3.4 Hàm khởi tạo ma trận Kernel làm mờ ảnh `genKernel()`

Input:

- `kernel_size`: kích thước của ma trận Kernel.
- `sigma`: hệ số của kernel. Thường là $(\text{kernel_size} - 1)/6$

Output:

- Ma trận Kernel [9] dùng để làm mờ theo yêu cầu có kích thước `kernel_size`

2.3.5 Hàm nhân 2 ma trận (Element-wise) `convolve()`

Input:

- `layer`: Ma trận gồm điểm gốc và các hàng xóm cục bộ của nó.
- `kernel`: Ma trận Kernel.

Output:

- Giá trị kết quả.

Thực hiện:

1. Nhân 2 ma trận `layer` và `kernel` theo element-wise. [8] [9]

2.3.6 Hàm thực hiện quá trình convolution cho toàn ma trận convolution()**Input:**

- `test_img`: ma trận của hình ảnh gốc
- `kernel`: Ma trận Kernel.

Output:

- Ma trận kết quả.

Thực hiện:

1. Duyệt qua 3 kênh màu của tất cả các vector màu trong `test_img` và thực hiện `convolve()` cho chúng gồm gom chúng trở lại thành 1 vector. [9]

2.3.7 Hàm làm mờ ảnh adjust_blur()**Input:**

- `test_img`: ma trận của hình ảnh gốc
- `size`: kích thước của Kernel, mặc định là 7.

Output:

- Ma trận kết quả.

Thực hiện:

1. Thực hiện `convolution()` cho `test_img` với Kernel là `genKernel()` kích thước `size`.

2.3.8 Hàm khởi tạo ma trận Kernel làm sắc nét ảnh sharpenKernel()**Output:**

- Ma trận làm sắc nét, kích thước 3x3

Thực hiện:

1. Trả về ma trận $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$

2.3.9 Hàm làm sắc nét ảnh adjust_sharpen()**Input:**

- `test_img`: ma trận của hình ảnh gốc

Output:

- Ma trận kết quả sau khi làm sắc nét.

Thực hiện:

1. Thực hiện `convolution()` cho `test_img` với Kernel là `sharpenKernel()`.

2.3.10 Hàm biến đổi ảnh màu thành ảnh xám to_grayscale()

Input:

- `test_img`: ma trận của hình ảnh gốc

Output:

- Ma trận kết quả sau khi biến đổi.

Thực hiện:

1. Dùng phương thức `dot` trong Numpy [2] để tính tích vô hướng giữa các vector màu với $[0.2989, 0.5870, 0.1140]$.
2. Biến đổi ma trận từ 1 chiều thành 3 chiều.

2.3.11 Hàm biến đổi ảnh màu thành ảnh sepia to_sepia()

Input:

- `test_img`: ma trận của hình ảnh gốc

Output:

- Ma trận kết quả sau khi biến đổi.

Thực hiện:

1. Tính tỉ lệ phần trăm phân phối RGB trong ảnh.
2. Sử dụng công thức để tính ra giá trị RGB mới cho từng vector màu.
3. Dưa các giá trị về thang màu 0 đến 255.

2.3.12 Hàm cắt ảnh ở trung tâm center_crop()

Input:

- `test_img`: ma trận của hình ảnh gốc
- `size_img`: kích thước của ma trận (vuông)
- `crop_size`: kích thước sau khi cắt.

Output:

- Ma trận kết quả

Thực hiện:

1. Duyệt x trong nửa khoảng $[(size_img - crop_size) \text{ div } 2, (size_img - crop_size) \text{ div } 2 + crop_size)$
2. Với các vector màu nằm trong vị trí đã duyệt (2 chiều) thì thêm vào ma trận kết quả.

2.3.13 Hàm cắt ảnh theo khung tròn circle_mask()

Input:

- **test_img**: ma trận của hình ảnh gốc
- **size_img**: kích thước của ma trận (vuông)
- **num**: số chiều của vector màu. (Phòng trường hợp RGBA)

Output:

- Ma trận kết quả

Thực hiện:

1. Dùng phương thức **ogrid()** trong thư viện Numpy [2] và bắt phương trình hình tròn để duyệt qua các điểm (vector màu) nằm trong ma trận vuông ban đầu.
2. Nếu điểm đó nằm ngoài hình tròn thì gán nó là vector màu đen.

2.3.14 Hàm cắt ảnh theo khung elip elip_mask()

Input:

- **test_img**: ma trận của hình ảnh gốc
- **size_img**: kích thước của ma trận (vuông)
- **num**: số chiều của vector màu. (Phòng trường hợp RGBA)

Output:

- Ma trận kết quả

Thực hiện:

1. Dùng phương thức **ogrid()** trong thư viện Numpy [2] và 2 bất phương trình hình elip nằm xiên góc 45 và -45 để duyệt qua các điểm (vector màu) nằm trong ma trận vuông ban đầu.
2. Nếu điểm đó nằm ngoài cả 2 hình elip thì gán nó là vector màu đen.

2.4 Hàm vận hành

2.4.1 Các hàm kiểm tra khả năng vận hành

Input:

- **img_name**: tên hình ảnh gốc.
- **raw_img**: ảnh sau khi đọc.
- **output_img**: mảng các ảnh sau khi thực hiện theo yêu cầu

- **choice:** Đối với những yêu cầu đặc biệt cần test cả ảnh màu lần ảnh xám khi choice = 0 như làm mờ ảnh, làm sắc nét ảnh thì cần thêm tham số choice.

Output:

- Title là tiêu đề tên ảnh sau khi chỉnh sửa.
- Ảnh đã chỉnh sửa được thêm vào mảng output_img.

Thực hiện:

1. Biến đổi bức ảnh thành 1 ma trận mà mỗi hàng là 1 vector màu với 3 giá trị màu bằng hàm reshape_img(). Nếu không, chỉ cần đưa raw_img về dạng np.array
2. Thực hiện yêu cầu bằng các hàm thực hiện.
3. Đưa ma trận kết quả về kiểu "uint8" và kích thước ban đầu (nếu cần).
4. Thêm ma trận sau khi định dạng vào mảng output_img.
5. Tạo tên theo đúng định dạng cho ảnh sau khi biến đổi.
6. Lưu ảnh với tên đã tạo và trả về tên ảnh.
7. Lưu ý: Đối với các yêu cầu cần input thêm như lật ảnh, làm mờ, làm sắc nét, cắt ảnh ở trung tâm thì cần nhập input trước khi thực hiện yêu cầu.

2.4.2 Hàm vận hành và xuất kết quả showResult()

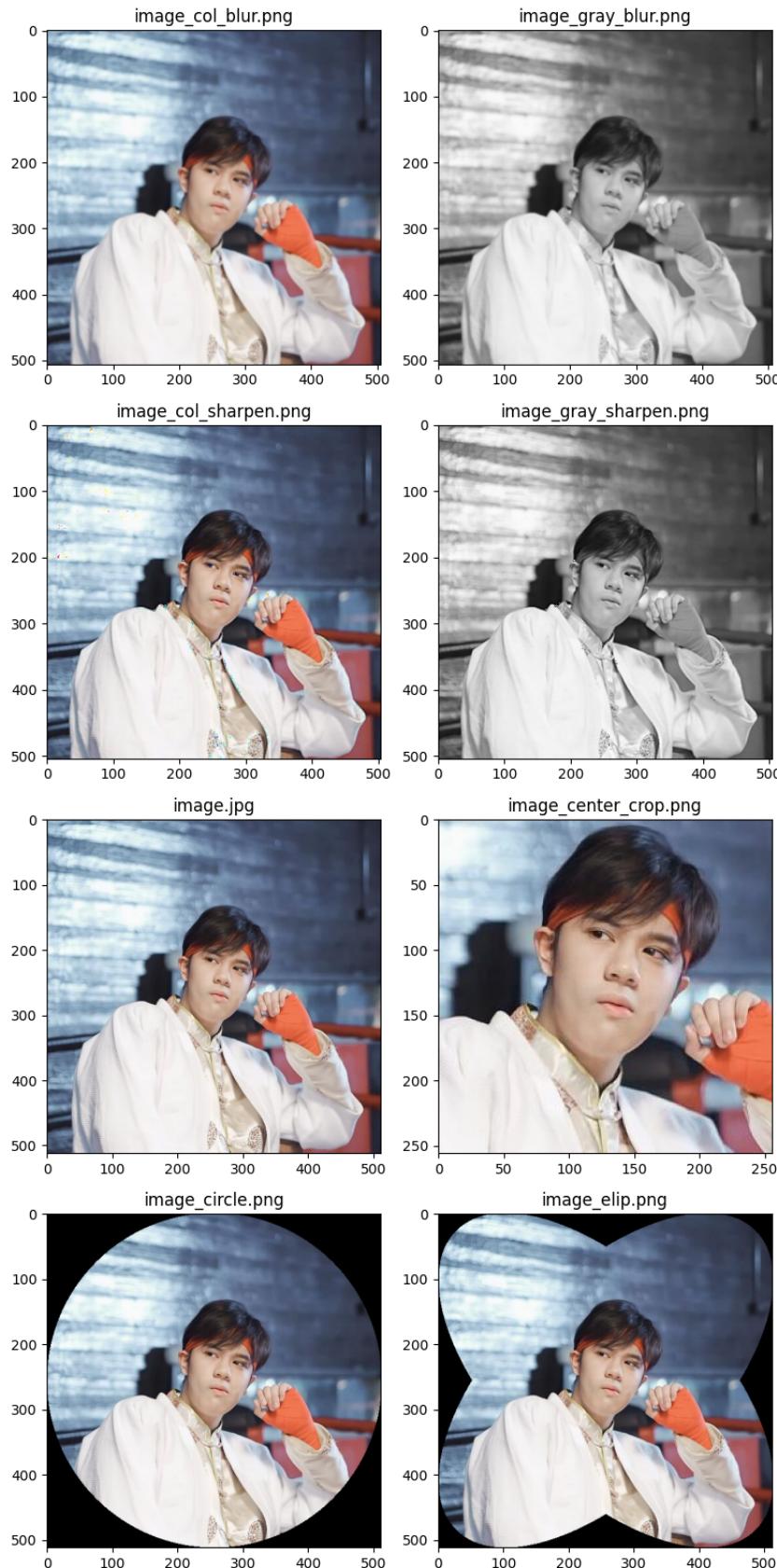
Thực hiện các thao tác để xuất ra kết quả cuối cùng:

- Đọc ảnh bằng readImg().
- Thêm ảnh đã đọc vào mảng output_img.
- Tùy thuộc vào giá trị choice đã nhập từ hàm main mà test các chức năng cần thiết.
- Xây dựng bối cảnh cho các hình ảnh kết quả bằng phương thức subplots() trong Matplotlib.
- Hiển thị kết quả bằng phương thức tight_layout()

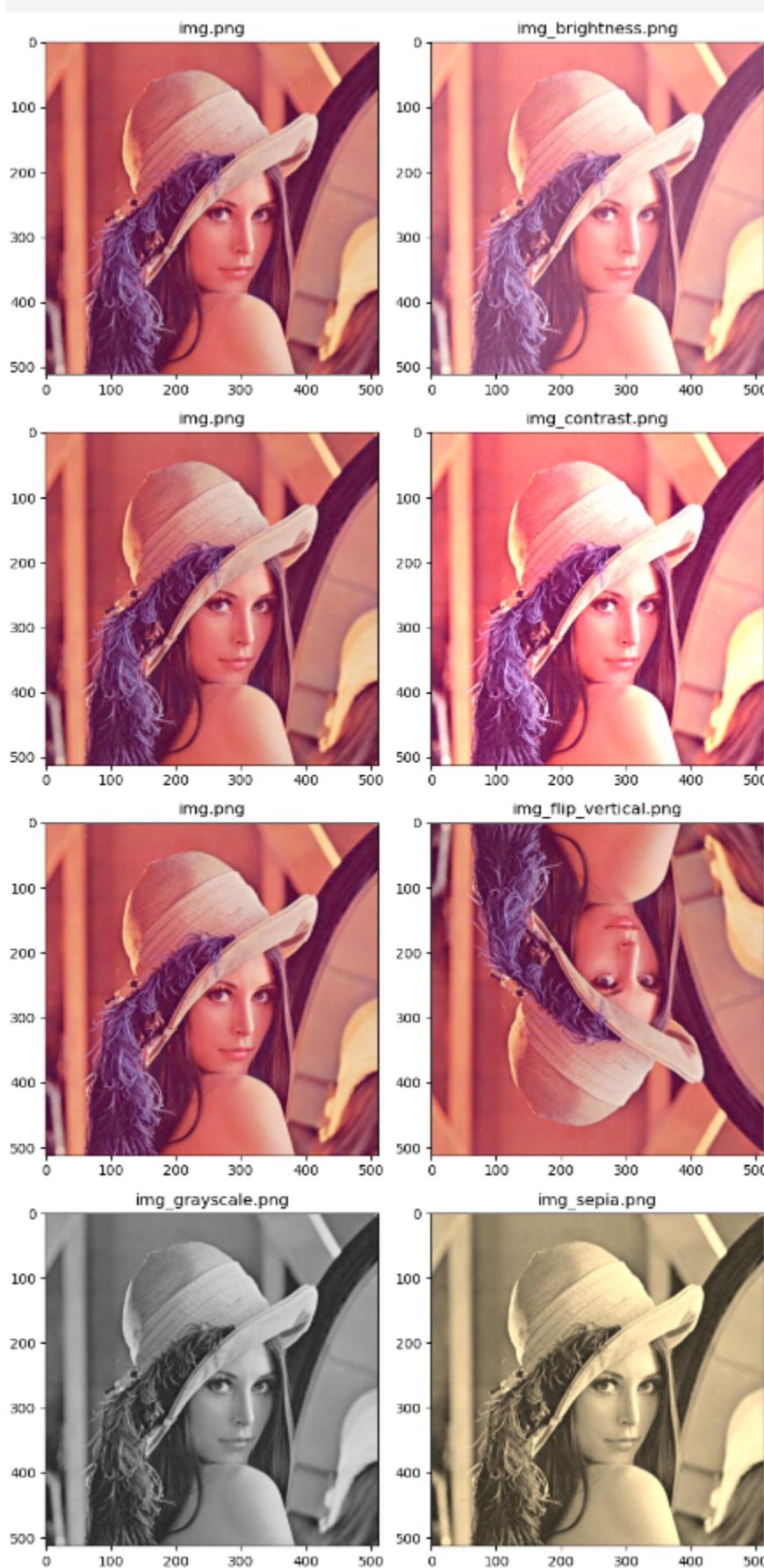
3 Kết quả thực hiện

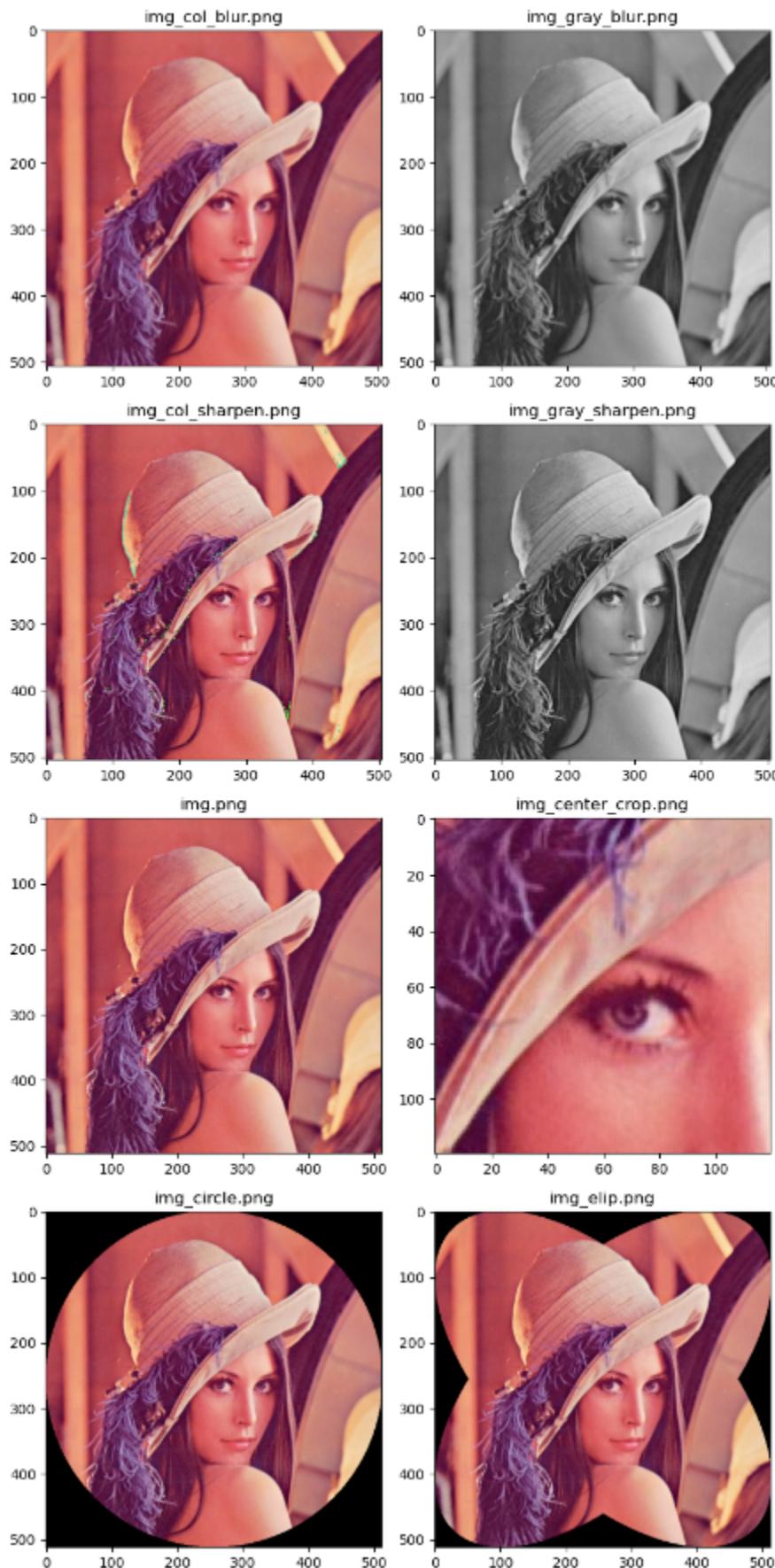
Kết quả thực hiện với hình đầu tiên.





Kết quả thực hiện với hình thứ 2.





4 Nhận xét

4.1 Nhận xét chung

Nhìn chung, kết quả thu được sau 2 lần thử khá tốt, hiệu quả xấp xỉ với yêu cầu ban đầu của đồ án [8]. Đối với yêu cầu làm mờ và làm sắc nét có thể thực hiện trên cả ảnh xám và ảnh màu. Đối với các yêu cầu cắt thì cần sử dụng ảnh có kích thước vuông để đạt được hiệu quả tối đa.

Thời gian thực hiện các yêu cầu cùng lúc tối đa cho ảnh kích thước 512x512 là không quá 15 giây.

Dánh giá mức độ hoàn thành của đồ án là 100%.

Yêu cầu	Mức độ hoàn thành	Ghi chú
Thay đổi độ sáng cho ảnh	100%	Tự động tăng độ sáng lên 50
Thay đổi độ tương phản	100%	Tự động tăng độ tương phản lên 1.38 lần
Lật ảnh	100%	Có thể lật ngang hoặc dọc
Chuyển đổi ảnh RGB thành ảnh xám/sepia	100%	
Làm mờ/sắc nét ảnh	100%	Thực hiện trên cả ảnh màu và ảnh xám
Cắt ảnh theo kích thước ở trung tâm	100%	
Cắt ảnh theo khung tròn	100%	
Cắt ảnh theo khung elip chéo nhau	100%	Nâng cao

4.2 Hạn chế nói chung

- Đối với các yêu cầu cắt ảnh thì chỉ thực hiện trên ảnh vuông, không tối ưu trên các ảnh có hình dạng khác.
- Về yêu cầu làm sắc nét vẫn xảy ra trường hợp bị lỗi màu ở một số điểm ảnh có màu tương tự nhau và chỉ thực hiện trên các ảnh đã làm mờ trước đó.

Tài liệu

- [1] Wikipedia contributors. Kernel (image processing). *Wikipedia, The Free Encyclopedia*, 2023.
- [2] NumPy Developers. Numpy reference. 2008.
- [3] FreeTuts. Masks và boolean arrays trong numpy. *freetuts.net*.
- [4] Dan Kalman. General equation of an ellipse. *The Journal of Online Mathematics and Its Applications*, 8, 2008.
- [5] StackExchange. How to get the limits of rotated ellipse? [math.stackexchange.com](https://math.stackexchange.com/questions/91132), Questions/91132.
- [6] StackOverflow. How can i convert an rgb image into grayscale in python? [stackoverflow.com](https://stackoverflow.com/questions/12201577), Questions/12201577.
- [7] StackOverflow. Processing an image to sepia tone in python. [stackoverflow.com](https://stackoverflow.com/questions/36434905), Questions/36434905.
- [8] ThS. Phan Thị Phương Uyên. Hướng dẫn thực hiện Đồ án 2: Image processing. 2023.
- [9] Đặng Ngọc Tiến. Project 2: Image processing. 2022.