

ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN



Project 02: Image Processing

Môn học: Toán ứng dụng và thống kê cho CNTT

Sinh viên thực hiện:

Lý Anh Quân (22127344)

Giáo viên hướng dẫn:

ThS. Vũ Quốc Hoàng

ThS. Phan Thị Phương Uyên

ThS. Nguyễn Văn Quang Huy

ThS. Nguyễn Ngọc Toàn

Ngày 31 tháng 7 năm 2024

Mục lục

1	Ý tưởng thực hiện	3
1.1	Thay đổi độ sáng cho ảnh	3
1.2	Thay đổi độ tương phản ảnh	3
1.3	Lật ảnh (ngang - dọc)	3
1.4	Chuyển đổi ảnh RGB thành ảnh xám hoặc sepia	3
1.5	Làm mờ hoặc sắc nét ảnh	5
1.6	Cắt ảnh theo kích thước (cắt ở trung tâm)	7
1.7	Cắt ảnh theo khung tròn hoặc khung elip	7
2	Mô tả hàm và thuật toán	11
2.1	Các thư viện sử dụng	11
2.2	Các hàm hỗ trợ	11
2.2.1	Hàm Đọc ảnh <code>read_Img()</code>	11
2.2.2	Hàm Hiển thị ảnh	11
2.2.3	Hàm Lưu ảnh	11
2.2.4	Hàm Chuyển đổi ảnh từ kích thước 2D sang 1D	12
2.2.5	Hàm vận hành <code>testcases</code>	12
2.3	Các hàm thực hiện	12
2.3.1	Hàm tăng độ sáng ảnh	12
2.3.2	Hàm tăng độ tương phản	12
2.3.3	Hàm lật ảnh ngang - dọc	12
2.3.4	Hàm chuyển đổi thành ảnh xám	13
2.3.5	Hàm chuyển đổi thành ảnh sepia	13
2.3.6	Hàm <code>apply_filter</code>	13
2.3.7	Hàm <code>crop_center</code>	14
2.3.8	Hàm cắt ảnh theo khung tròn <code>circle_mask</code>	14
2.3.9	Hàm <code>ellipse_mask</code>	14
3	Kết quả thực hiện	15

4 Nhận xét	23
4.1 Nhận xét tổng quát	23
4.2 Hạn chế	23
Tài liệu tham Khảo	27

1 Ý tưởng thực hiện

1.1 Thay đổi độ sáng cho ảnh

- Về cơ bản muốn tăng độ sáng cho ảnh thì ta sẽ cần nhiều màu trắng, tức là các giá trị điểm ảnh sẽ gần với 255 thì ảnh sẽ sáng lên. Do đó ta có thể sử dụng **phép cộng ma trận với scalar (broadcasting)** để tăng các giá trị điểm ảnh. [6]
- Khi đó, với mỗi vector màu $v = [v1, v2, v3]$, muốn tăng độ sáng ta sẽ tăng một lượng alpha nên vector màu sau đó là $v = [v1 + \alpha, v2 + \alpha, v3 + \alpha]$,
- Tuy nhiên chúng ta phải chú ý rằng giá trị điểm ảnh không được vượt quá 255 và không nhỏ hơn 0, ta sẽ sử dụng `np.clip()` để đảm bảo chuyện này.

1.2 Thay đổi độ tương phản ảnh

Độ tương phản của ảnh phụ thuộc vào sự chênh lệnh của các giá trị điểm ảnh. Để độ chênh lệnh sáng và tối này càng lớn, chúng ta có thể sử dụng **phép toán nhân ma trận với một scalar > 1** để gia tăng sự chênh lệnh này. [2]

Chúng ta cũng cần phải chú ý rằng giá trị điểm ảnh không được vượt quá 255 và không nhỏ hơn 0, ta sẽ sử dụng `np.clip()` để đảm bảo chuyện này.

1.3 Lật ảnh (ngang - dọc)

Lật ảnh ngang hoặc dọc tức là đảo ngược các vector dòng hoặc cột. Khi đó ta có thể sử dụng hàm `np.fliplr` để lật ảnh ngang (chiều từ trái sang phải) và hàm `np.flipud` để lật ảnh dọc (chiều từ trên xuống). Ví dụ: trên `ma trận gốc => ma trận lật ngang => ma trận lật dọc`

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

1.4 Chuyển đổi ảnh RGB thành ảnh xám hoặc sepia

Có 2 cách chính để có thể chuyển đổi ảnh màu thành ảnh xám.

- Average method
- Weighted method

Ý tưởng của phương pháp trung bình khá đơn giản, lấy giá trị trung bình của ba thành phần màu $(R + G + B)/3$. Phương pháp này giả định rằng mỗi màu có đóng góp ngang nhau (33%) trong ảnh xám. Tuy nhiên, điều này không phản ánh chính xác với cách mà mắt người cảm nhận ánh sáng và màu sắc.

Thực tế, mỗi màu có bước sóng và mức độ ảnh hưởng khác nhau đến cách chúng ta nhìn thấy hình ảnh. Do đó, phương pháp trung bình đơn giản không phù hợp. Thay vào đó, chúng ta có thể sử dụng một cải tiến là phương pháp dựa vào độ sáng (**luminosity method**). Phương pháp này sử dụng các trọng số khác nhau để phản ánh cách mắt người cảm nhận độ sáng của từng màu. Qua nhiều thí nghiệm, các nhà tâm lý học tìm ra cách chúng ta cảm nhận độ sáng của màu đỏ, xanh lá và xanh dương khác nhau như thế nào. Vì vậy, công thức độ chói sử dụng các trọng số cụ thể để tạo ra một ảnh xám chính xác hơn: [4]

$$G = 0.2126R + 0.7152G + 0.0722B$$

Trong đồ án này em sẽ sử dụng **Weighted method** - phương pháp mà thư viện **OpenCV** đã sử dụng thuật toán. Ý tưởng chính là chúng ta sẽ thay đổi sự đóng góp của 3 kênh màu RGB thành một kênh màu (xám) [3]

Màu đỏ có bước sóng lớn nhất trong ba màu, và màu xanh lá là màu không chỉ có bước sóng ngắn hơn màu đỏ mà còn là màu mang lại hiệu ứng dịu mắt hơn. Điều này có nghĩa là chúng ta phải giảm tỷ lệ đóng góp của màu đỏ, tăng tỷ lệ đóng góp của màu xanh lá, và đặt tỷ lệ đóng góp của màu xanh dương nằm giữa hai màu này. Công thức biến đổi sẽ là:

$$G = 0.299R + 0.587G + 0.114B$$

Để chuyển bức ảnh màu thành ảnh sepia, ta cũng sẽ sử dụng công thức biến đổi theo kênh màu tuy nhiên khác biệt là sẽ có 3 kênh màu như RGB và trọng số của 1 kênh màu cũng được tính toán dựa vào các trọng số màu chứ không còn như nhau ở ba kênh màu như ở ảnh xám [5]

- 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

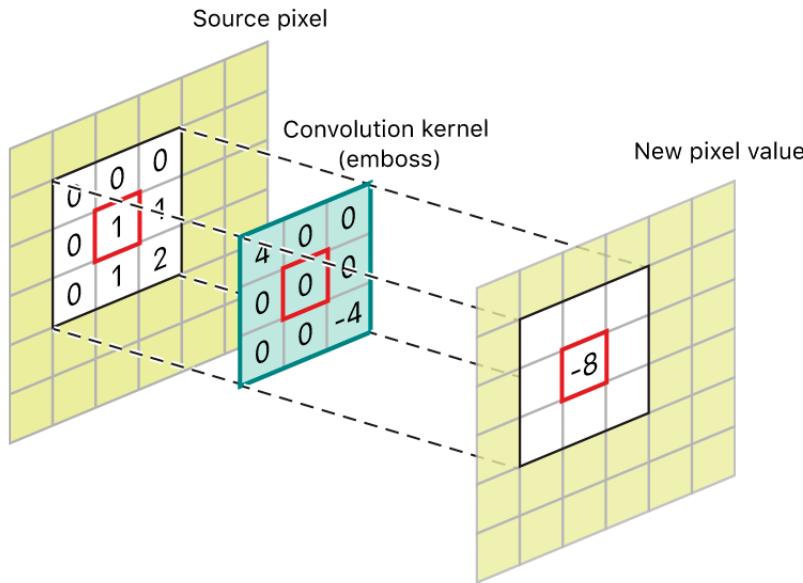
Để thực hiện được phép tính này, cũng như ảnh xám, ta thực hiện nhờ vào phép nhân giữa vector điểm ảnh với ma trận trọng số màu. Thực hiện cho toàn bộ điểm ảnh RGB ta sẽ thu được ma trận mới chứa các điểm ảnh Sepia.

1.5 Làm mờ hoặc sắc nét ảnh

- Ý tưởng cho việc làm mờ hay sắc nét ảnh là ta cần tính toán lại giá trị màu của toàn bộ điểm ảnh bằng cách áp dụng một bộ lọc **Gaussian Kernel** lên ảnh.
- Gaussian Kernel là một ma trận vuông số học với các giá trị có được thông qua phân phối Gauss. Vì ta sẽ cần vị trí trung tâm của ma trận nên kích thước của ma trận là một số lẻ. Khi áp dụng Gaussian Kernel lên ảnh, mỗi điểm ảnh sẽ được thay thế bằng trung bình của các điểm ảnh xung quanh nó (neighbors), với trọng số lớn nhất tại điểm đó. Các điểm ảnh càng xa điểm đang xét, trọng số càng nhỏ. Ta có thể minh họa và thử nghiệm nhiều loại kernel khác qua website này <https://setosa.io/ev/image-kernels/> Với n = 3, kernel phổ biến để làm mờ và sắc nét ảnh là:

$$Blur = \frac{1}{16} \times \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}, Sharpen = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

- Sau đó, ta sẽ thực hiện phép tích chập (convolution) giữa ma trận điểm ảnh gốc và Gaussian Kernel. Ta sẽ lấy ra từng phần tử x_{ij} của ma trận ảnh gốc X sao cho nó là trung tâm của một ma trận con có kích thước bằng với Kernel. Thực hiện Phép tích chập giữa ma trận con và Kernel sẽ thu được giá trị màu của điểm ảnh kết quả. [7]



Hình 1: Minh họa cho quá trình Convolution

- Tuy nhiên với thuật toán trên ta sẽ phải giải quyết vấn đề khi tính toán cho các điểm ảnh ở ngoài rìa mà không thể tìm được ma trận A sao cho x_{ij} làm trung tâm thì cách giải quyết là ta sẽ cần thêm các padding vào 4 cạnh của X. Có nghĩa là ta sẽ thêm các điểm ảnh có giá trị 0 vào các cạnh của ma trận gốc X sao cho luôn đảm bảo sẽ tìm được ma trận con để thực hiện phép tích chập. Chính vì các padding này có giá trị là 0 nên sẽ không làm ảnh hưởng đến chất lượng màu sắc của ảnh sau khi làm mờ/sắc nét.
- Ta có thể lấy ví dụ minh họa thông qua việc làm mờ một bức ảnh 2x2 với Gaussian Kernel 3x3:
 - Bước 1: Thêm các padding vào ma trận ảnh gốc trước khi thực hiện phép tích chập

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \Rightarrow X = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- Bước 2: Xét phần tử x_{11} từ ma trận gốc, lấy ra ma trận con A sao cho x_{11} là trung tâm của ma trận. Sau đó thực hiện tích chập giữa A và bộ lọc Kernel

$$X = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \Rightarrow A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 3 & 4 \end{bmatrix} \times \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \frac{9}{8} \Rightarrow Y = \begin{bmatrix} \frac{9}{8} & \dots \\ \dots & \dots \end{bmatrix}$$

- Bước 3: Lặp lại bước 2 với các phần tử còn lại ta sẽ ra được kết quả ma trận Y đại diện cho ảnh sau khi được làm mờ. Quá trình làm sắc nét ảnh cũng tương tự như vậy.

$$X = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \Rightarrow \dots Y = \begin{bmatrix} \frac{9}{8} & \frac{21}{16} \\ \frac{3}{2} & \frac{7}{4} \end{bmatrix}$$

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

- Vì một bức ảnh là một ma trận các điểm ảnh, vì thế nên khi ta muốn cắt 1 bức ảnh ở trung tâm, ta có thể xóa đi các cột và hàng ở rìa ma trận.
- Thực hiện việc xóa đi số dòng, cột ở ngoài rìa bằng cách chỉ copy các dòng, cột ở trung tâm ma trận cũ sang một ma trận mới. Chuyển ma trận mới này thành ảnh thì sẽ cho ta được một bức ảnh mới có kích thước nhỏ hơn được cắt ở trung tâm ảnh gốc.

1.7 Cắt ảnh theo khung tròn hoặc khung elip

- Đối với khung hình tròn, ý tưởng chính của chúng ta là sử dụng kỹ thuật mask và boolean array kết hợp với chương trình đường tròn để xác định các điểm ảnh nằm bên trong đường tròn, các điểm nằm bên ngoài sẽ thay thế bằng giá trị màu đen. Với R là một nửa kích thước ảnh 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:

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

- Đối với khung hình elip, ta cũng tương tự dùng công thức tổng quát cho hình elip và tính toán giá trị lớn nhất sao cho 2 hình elip nằm xiên 1 góc α trong khung hình vuông: [1]

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

- Tuy nhiên vấn đề ở đây sẽ là làm sao cho Elip của ta có thể tiếp xúc với tất cả các cạnh của bức ảnh. Để giải quyết vấn đề này ta cần giải quyết theo mặt toán học để tìm mối liên hệ giữa trục dài, trục ngắn (a,b) của Elip với kích thước của bức ảnh.
- Xét cụ thể hình Elip được quay góc 45 độ quanh tâm, ta khai triển để tách x, y ra riêng biệt

$$\begin{aligned} & \frac{(x\cos(\frac{\pi}{4}) - y\sin(\frac{\pi}{4}))^2}{a^2} + \frac{(x\sin(\frac{\pi}{4}) + y\cos(\frac{\pi}{4}))^2}{b^2} = 1 \\ \leftrightarrow & x^2(\frac{\cos^2(\frac{\pi}{4})}{a^2} + \frac{\sin^2(\frac{\pi}{4})}{b^2}) - yx(\frac{\sin(\frac{\pi}{2})}{a^2} - \frac{\sin(\frac{\pi}{2})}{b^2}) + y^2(\frac{\cos^2(\frac{\pi}{4})}{a^2} + \frac{\sin^2(\frac{\pi}{4})}{b^2}) = 1 \\ \leftrightarrow & x^2(\frac{1}{2a^2} + \frac{1}{2b^2}) - yx(\frac{1}{a^2} - \frac{1}{b^2}) + y^2(\frac{1}{2a^2} + \frac{1}{2b^2}) - 1 = 0(1) \end{aligned}$$

- Vì elip có tính đối xứng nên ta chỉ cần tìm điều kiện a, b sao cho Elip tiếp xúc một cạnh thì sẽ thỏa tiếp xúc 3 cạnh còn lại.
- Giả sử Elip tiếp xúc với cạnh hình vuông tại điểm có tung độ $y = c$, thì ta sẽ cần tìm a, b sao cho chỉ có duy nhất 1 nghiệm x thỏa phương trình (1) với $y = c$. Thay $y = c$ vào (1) thì (1) sẽ trở thành phương trình biến x với hai tham số a, b:

$$F(x) = (\frac{1}{2a^2} + \frac{1}{2b^2})x^2 - c(\frac{1}{a^2} - \frac{1}{b^2})x + c^2(\frac{1}{2a^2} + \frac{1}{2b^2}) - 1$$

- Vì $F(x)$ là một Parabol nên để $F(x) = 0$ có duy nhất một nghiệm thì đỉnh của Parabol này phải nằm trên trục Ox mà tọa độ đỉnh Parabol là $(-b/2a; F(-b/2a))$ với a là hệ số của x^2 và b là hệ số của x . Suy ra $F(-b/2a)$ phải bằng 0.

$$\begin{aligned} \leftrightarrow & (\frac{1}{2a^2} + \frac{1}{2b^2})(\frac{c(\frac{1}{a^2} + \frac{1}{b^2})}{2(\frac{1}{2a^2} + \frac{1}{2b^2})})^2 - c(\frac{1}{a^2} - \frac{1}{b^2})(\frac{c(\frac{1}{a^2} + \frac{1}{b^2})}{2(\frac{1}{2a^2} + \frac{1}{2b^2})}) + c^2(\frac{1}{2a^2} + \frac{1}{2b^2}) - 1 = 0 \\ \leftrightarrow & (\frac{c^2}{2})[\frac{1}{a^2} + \frac{1}{b^2} - \frac{(\frac{1}{a^2} - \frac{1}{b^2})^2}{\frac{1}{a^2} + \frac{1}{b^2}}] = 1 \end{aligned}$$

$$\leftrightarrow \frac{\frac{4}{a^2 b^2}}{\frac{1}{a^2} + \frac{1}{b^2}} = \frac{2}{c^2}$$

$$\leftrightarrow a^2 + b^2 = 2c^2$$

- Nếu ta xem tâm bức ảnh trùng với gốc tọa O thì c sẽ bằng một nửa kích thước cạnh của bức ảnh. Vì thế để Elip của ta tiếp xúc được với cả bốn cạnh bức ảnh và hai trục Elip trùng với hai đường chéo bức ảnh thì góc quay phải là 45 độ và -45 độ, đồng thời độ dài hai trục a, b ($a > b$) của Elip phải luôn thỏa:

$$a^2 + b^2 = \frac{canh^2}{2}$$

- Ta có thể chọn a, b tùy ý dựa vào phương trình tổng quát và điều kiện trên:
- Giả sử ta có bức ảnh vuông $512 \times 512 \Rightarrow c = 256$, a và b theo tỷ lệ: $a = kb(2)$

$$\leftrightarrow a^2 + b^2 = 2 \times 256^2 = 131072$$

$$\leftrightarrow (kb)^2 + b^2 = 131072$$

$$\leftrightarrow b^2(k^2 + 1) = 131072$$

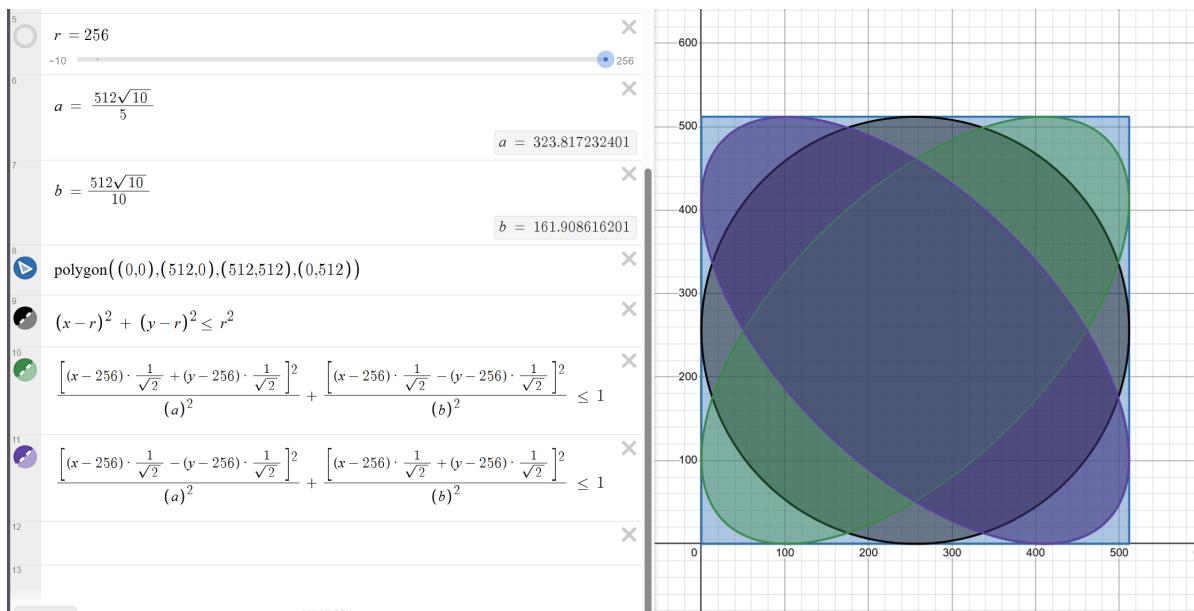
$$b = \sqrt{\frac{131072}{k^2 + 1}}$$

- Thay vào (2): $a = k\sqrt{\frac{131072}{k^2 + 1}}$

- Ta có thể thay $k = 2$ và được:

$$a = \frac{512\sqrt{10}}{5}; b = \frac{512\sqrt{10}}{10}$$

- Ta có $\sin(45) = \cos(45) = \frac{\sqrt{2}}{2}$ thay vào phương trình (I), để được phương trình Elip thứ nhất và $\sin(-45) = \frac{\sqrt{2}}{2}$ để được phương trình elip thứ 2.
- Các điểm ảnh nằm bên ngoài cả Elip nghiêng 45 độ và Elip nghiêng -45 độ thì sẽ bị thay đổi giá trị màu thành màu đen (0, 0, 0). Như vậy ảnh của ta sẽ được cắt theo khung hai hình Elip nghiêng chồng lên nhau.



Hình 2: Minh họa cho hàm số biểu diễn khung tròn và 2 khung elip chéo nhau

2 Mô tả hàm và thuật toán

2.1 Các thư viện sử dụng

- NumPy: Tính toán ma trận
- PIL: Đọc, ghi ảnh
- matplotlib: Hiển thị ảnh

2.2 Các hàm hỗ trợ

2.2.1 Hàm Đọc ảnh read_Img()

Input: img_path là đường dẫn tới tập tin ảnh (ảnh phải nằm chung folder với file .ipynb).

Sử dụng phương thức `Image.Open()` của thư viện PIL để mở ảnh.

Sau đó ta sẽ biến đổi ảnh thành 1 ma trận với giá trị mỗi hàng là 1 vector màu có 3 giá trị màu RGB bằng `np.array()` của thư viện Numpy

Output: Trả về kết quả ma trận điểm ảnh

2.2.2 Hàm Hiển thị ảnh

Input:

- src_img là ảnh ban đầu khi đã dùng hàm `read_Img()`
- img_path là đường dẫn ảnh
- result_img là ảnh sau khi xử lý
- func là chuỗi để biểu diễn cho chức năng thực thi

Sử dụng `plt.imshow()` để hiển thị ảnh gốc ban đầu và ảnh sau xử lý

2.2.3 Hàm Lưu ảnh

Input: result_img, img_path và func là ảnh sau xử lý, đường dẫn để lưu ảnh và chức năng đã xử lý

Sử dụng `split()` để xử lý chuỗi cho tên của ảnh và lưu lại.

2.2.4 Hàm Chuyển đổi ảnh từ kích thước 2D sang 1D

Input: img_2d là ảnh ban đầu khi đã dùng hàm `read_Img()`

Sử dụng `shape()` để lấy ra 3 giá trị tương ứng của mỗi hàng (height, width, channel). Và dùng `reshape()` để giảm số chiều của ma trận xuống thành (height * width, channel) Output: Trả về ảnh 1D

2.2.5 Hàm vận hành testcases

Input: Choice là lựa chọn của người dùng và img_path là đường dẫn của ảnh

Hàm sẽ đọc đường dẫn ảnh và trả về ảnh gốc. Sau đó nhận vào các lựa chọn tương ứng với các chức năng của chương trình. Nếu chọn 0 thì sẽ thực hiện tất cả chức năng. Khi lựa chọn một chức năng bất kỳ, ảnh gốc src_img sẽ được đưa vào các hàm test rồi thực hiện thuật toán. Sau đó sẽ được xuất ra màn hình và lưu lại kèm đường dẫn ứng với tên hàm chức năng.

2.3 Các hàm thực hiện

2.3.1 Hàm tăng độ sáng ảnh

Input: img và brightness là ảnh ban đầu và độ sáng muốn tăng (ở đây mặc định là 50)

Thực hiện phép cộng ma trận với một scalar (broadcast) để các giá trị điểm màu của ảnh được tăng lên gần với 255. Sử dụng `np.clip()` để đảm bảo giá trị điểm ảnh nằm trong khoảng [0,255]

Output: Trả ra ảnh sau xử lý

2.3.2 Hàm tăng độ tương phản

Input: img và contrast là ảnh ban đầu và độ tương phản muốn tăng (ở đây mặc định là 1.28)

Thực hiện phép nhân ma trận với một scalar (broadcast) để các giá trị điểm màu có sự chênh lệch.

Sử dụng `np.clip()` để đảm bảo giá trị điểm ảnh nằm trong khoảng [0,255]

Output: Trả ra ảnh sau xử lý

2.3.3 Hàm lật ảnh ngang - dọc

Input: img và direction là ảnh ban đầu và lựa chọn hướng muốn xoay

Sử dụng hàm `np.fliplr()` để revert ma trận từ dưới lên trên và sử dụng `np.flipud()` để revert ma trận từ trái sang phải.

Output: Trả ra ảnh sau xử lý

2.3.4 Hàm chuyển đổi thành ảnh xám

Input: img ảnh ban đầu chưa xử lý

Output: Trả ra ảnh sau xử lý

Thực hiện phương thức `dot()` để tính tích vô hướng giữa các vector màu với trọng số của ảnh xám. Cần đảm bảo ảnh đầu ra vẫn giữ nguyên định dạng 3D bằng cách gán 3 kênh màu đều có giá trị của ảnh xám.

2.3.5 Hàm chuyển đổi thành ảnh sepia

Input: img ảnh ban đầu chưa xử lý

Output: Trả ra ảnh sau xử lý

Thực hiện phương thức `matmul()` của Numpy để nhân tất cả vector điểm ảnh màu với ma trận trọng số màu. Sau đó trả về ảnh Sepia từ ma trận điểm ảnh Sepia

2.3.6 Hàm apply_filter

Input: img ảnh ban đầu chưa xử lý

Output: Trả ra ảnh sau xử lý

- Xác định kích thước của ảnh cũng như của màng lọc ‘kernel’.
- Xác định kích thước (số lượng) padding cần thêm vào ma trận ảnh gốc.
- Tạo ma trận **b** sau đó thực hiện sao chép các điểm ảnh của ảnh gốc vào trung tâm của ma trận này sao cho các padding sẽ nằm ở rìa các cạnh của ma trận.
- Thực hiện phép tích chập nhờ kỹ thuật slicing `b[i : i + kernel_height, j : j + kernel_width, :]` sẽ giúp ta lấy ra ma trận con của ma trận **b** sao cho điểm ảnh đang xét nằm ở trung tâm ma trận.
- Vì kernel chỉ có kích thước là `height x width` so với ma trận ảnh 3 chiều nên ta cần phải thêm một chiều mới vào mảng kernel bằng cách sử dụng `np.newaxis()` trong Numpy.

- Bước cuối cùng trong phép tích chập là tính tổng tất cả các phần tử (điểm ảnh) trong ma trận này để cho ra một điểm ảnh mới có 3 kênh màu thay thế cho điểm ảnh cũ. Ta thực hiện bằng cách dùng `np.sum()`.
- Khi làm sắc nét ảnh ta cũng cần phải đảm bảo giá trị màu không vượt quá 255 bằng cách sử dụng `np.clip()`
- Sau đó ta chuyển ma trận ảnh thành ảnh nhờ vào `Image.fromarray`

2.3.7 Hàm crop_center

Input: img là ảnh ban đầu chưa xử lý

Output: Trả ra ảnh sau xử lý

- Xác định kích thước ảnh gốc size và kích thước ảnh kết quả bằng một nửa ảnh gốc.
- Do ảnh kết quả bằng một nửa ảnh gốc và cắt từ trung tâm nên các điểm ảnh ở các dòng, cột thuộc đoạn $[\frac{size}{4}, \frac{3size}{4}]$ sẽ được lưu vào ma trận điểm ảnh kết quả crop_arr.
- `img_arr[start_point:(3*size//4)-1, start_point:(3*size//4)-1]` sẽ thực hiện lấy các điểm ảnh thuộc hàng, cột từ khoảng trên

2.3.8 Hàm cắt ảnh theo khung tròn circle_mask

Input: img là ảnh ban đầu chưa xử lý, size_img kích thước của ma trận (**vuông**)

Output: Trả ra ảnh sau xử lý

1. Dùng phương thức `ogrid()` trong thư viện Numpy 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.

2.3.9 Hàm elipse_mask

Input: img là ảnh ban đầu chưa xử lý

Output: Trả ra ảnh sau xử lý

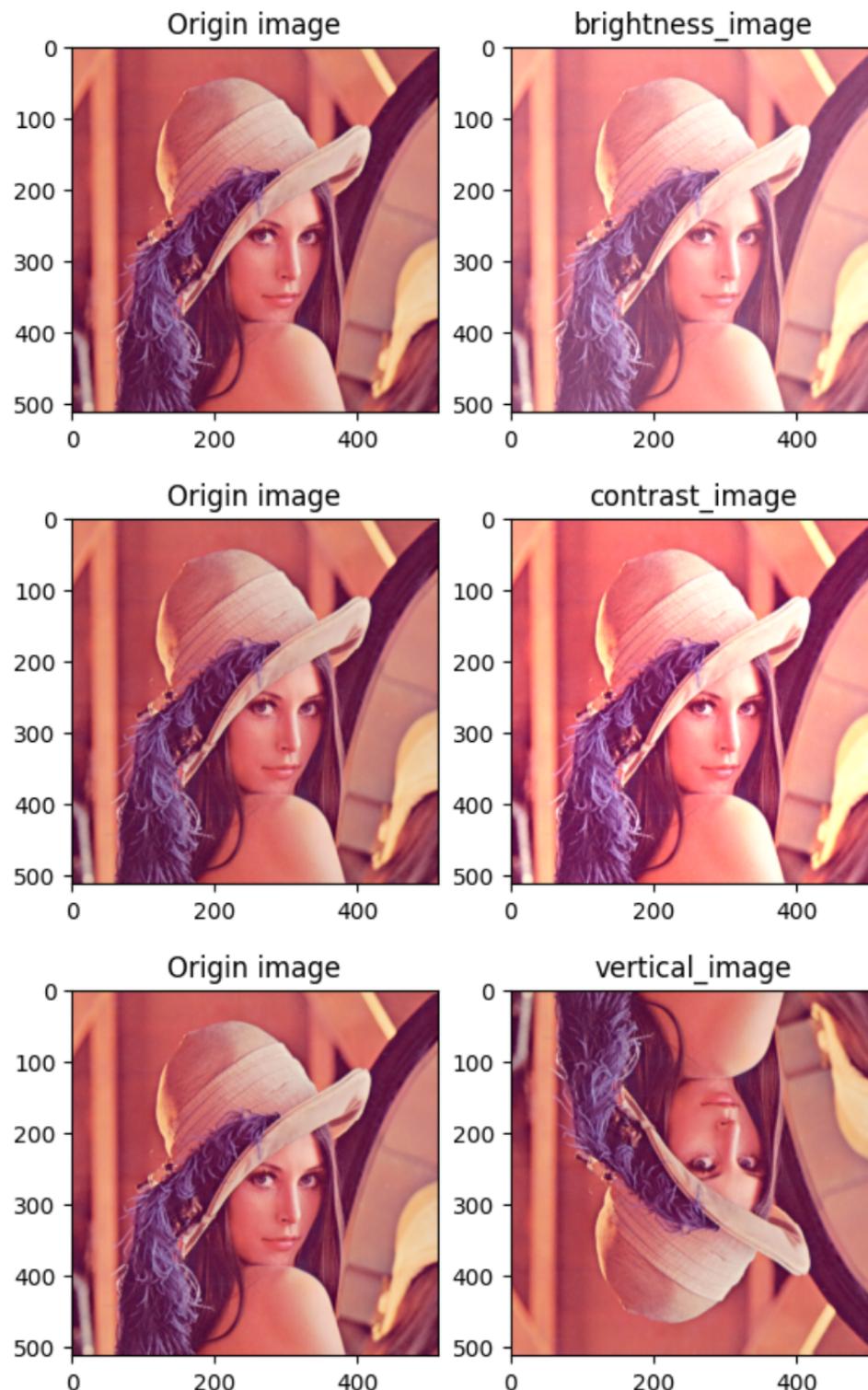
1. Dùng phương thức ogrid() trong thư viện Numpy và bắt phương trình hình elip nằm xiên 2 góc 45 độ và -45 độ để duyệt qua các điểm (vector màu) nằm trong ma trận vuông ban đầu
2. Nếu điểm đó nằm ngoài hình tròn thì gán nó là vector màu đen.

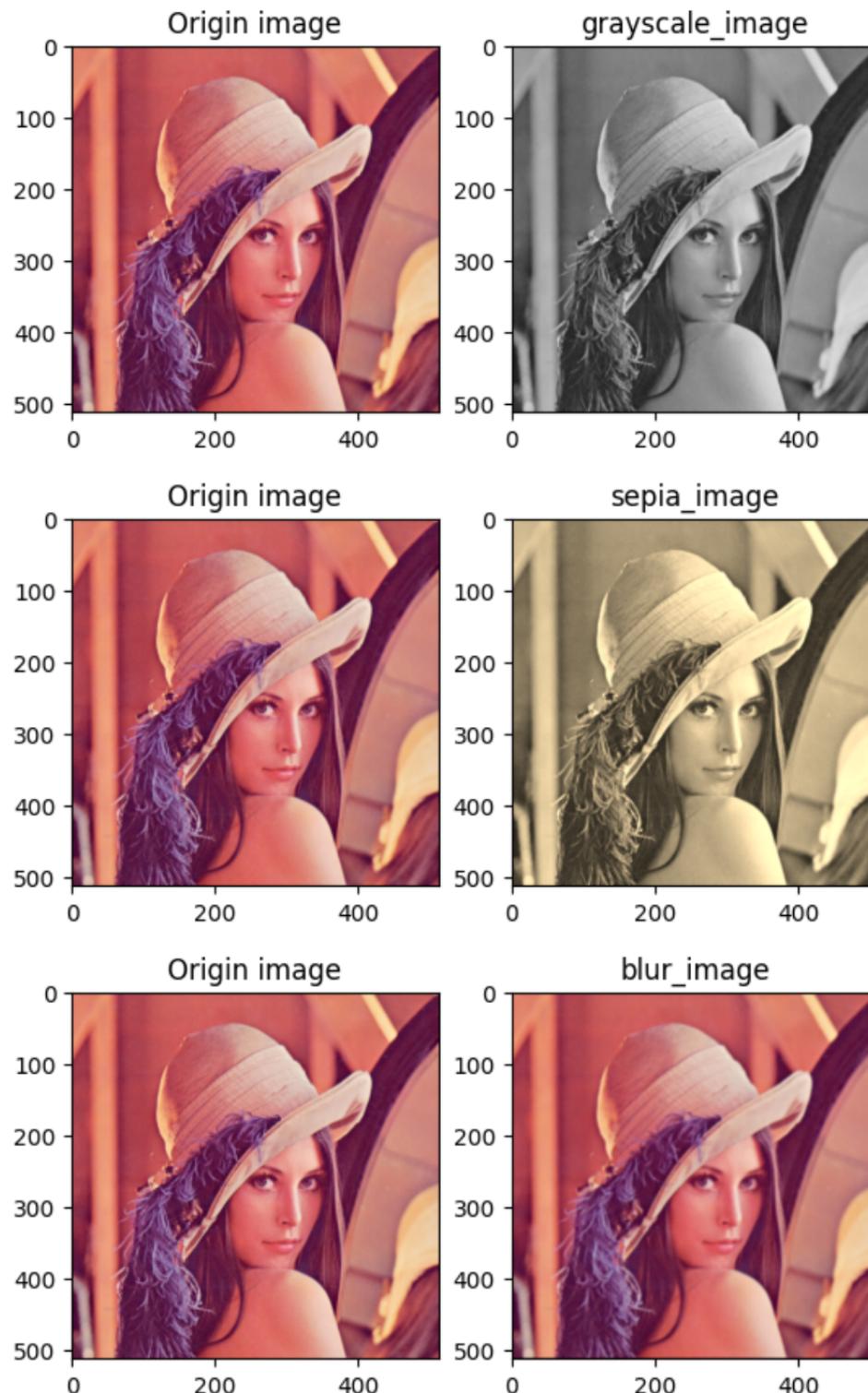
3 Kết quả thực hiện

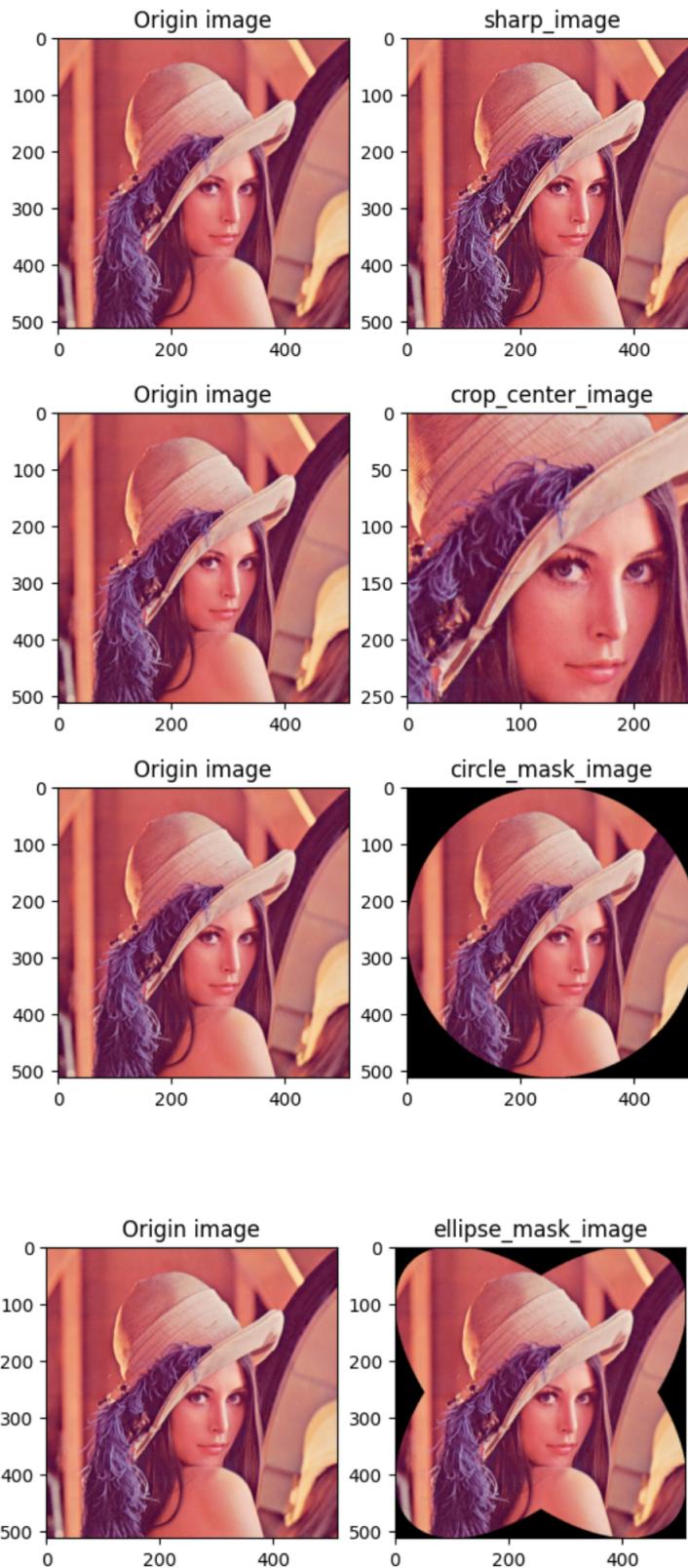


Hình 3: Hình ví dụ (512x512)

Kết quả thực hiện với hình 3



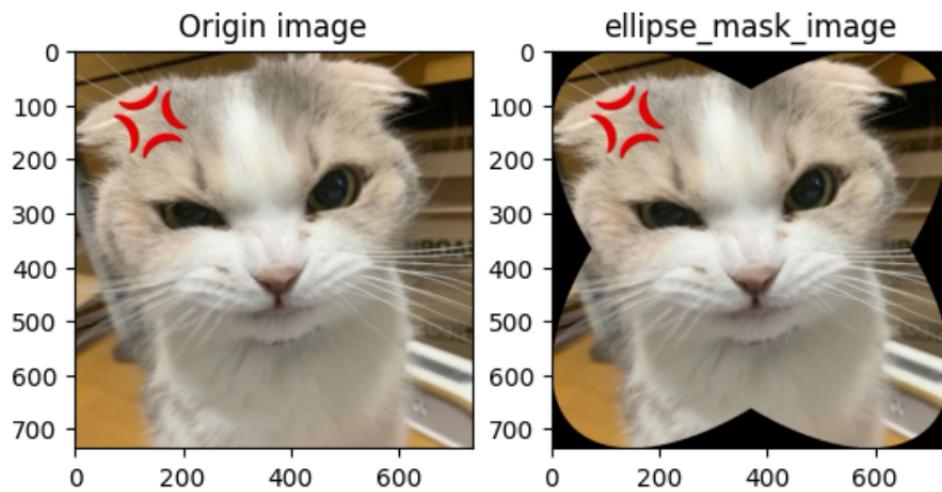


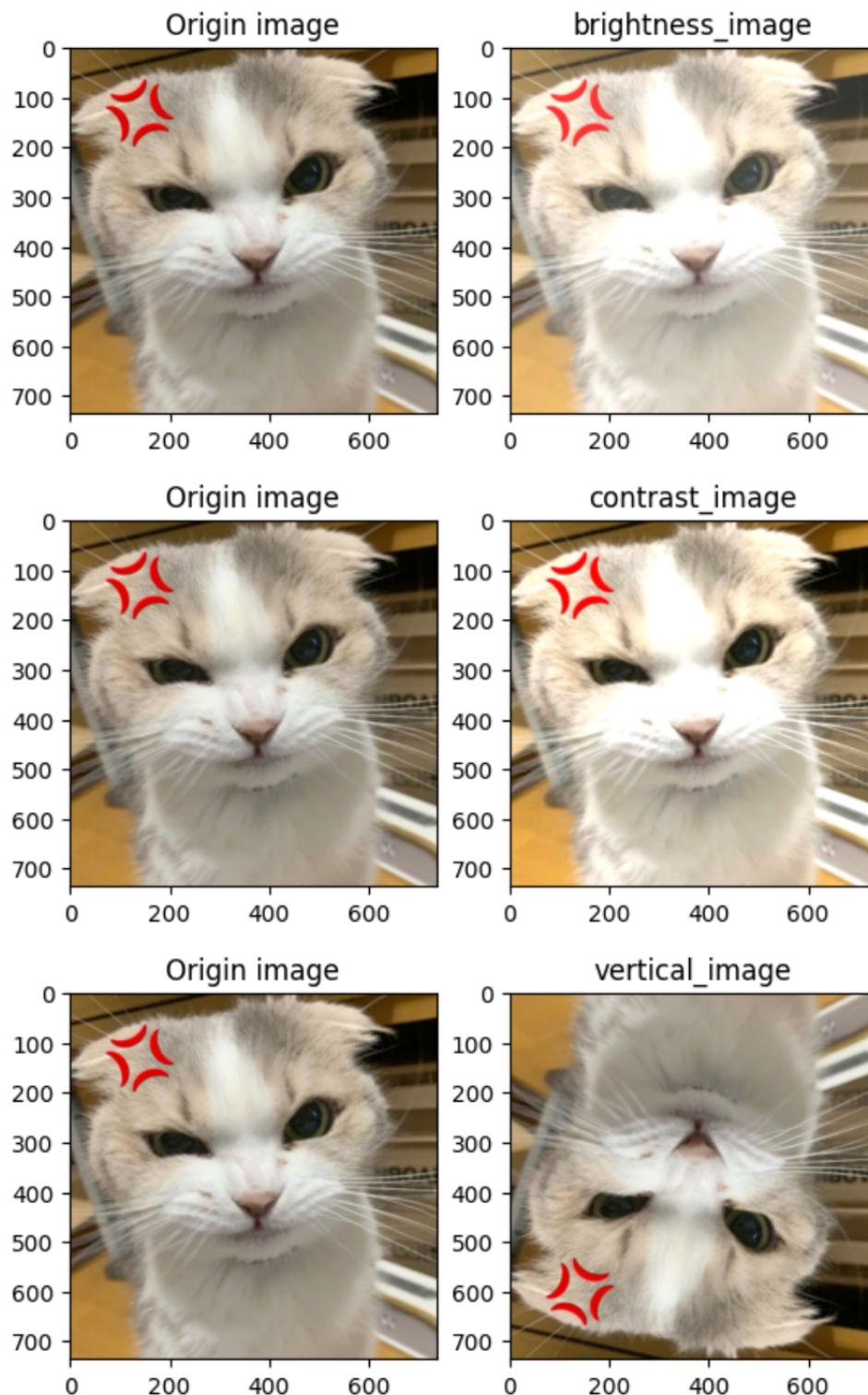


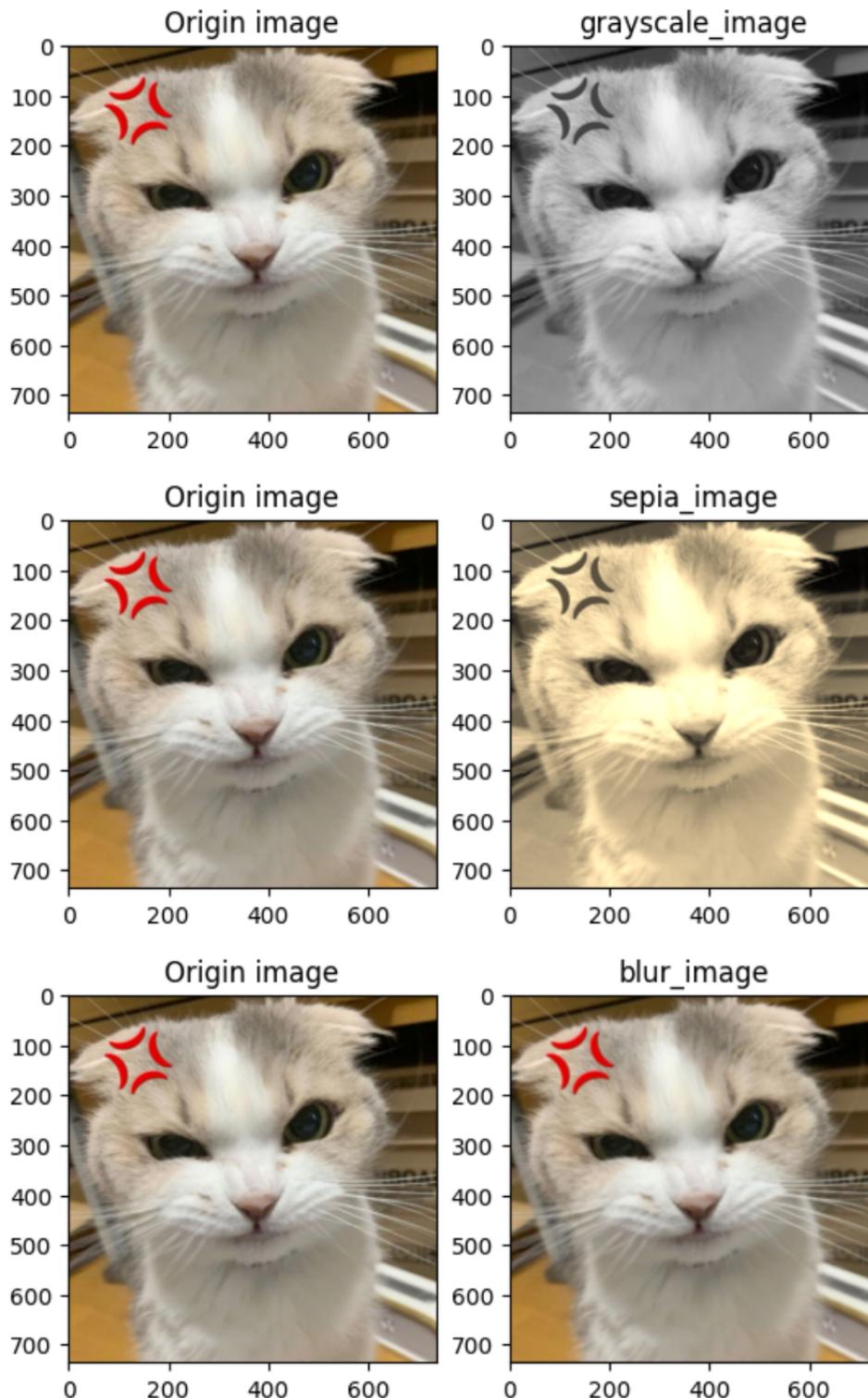


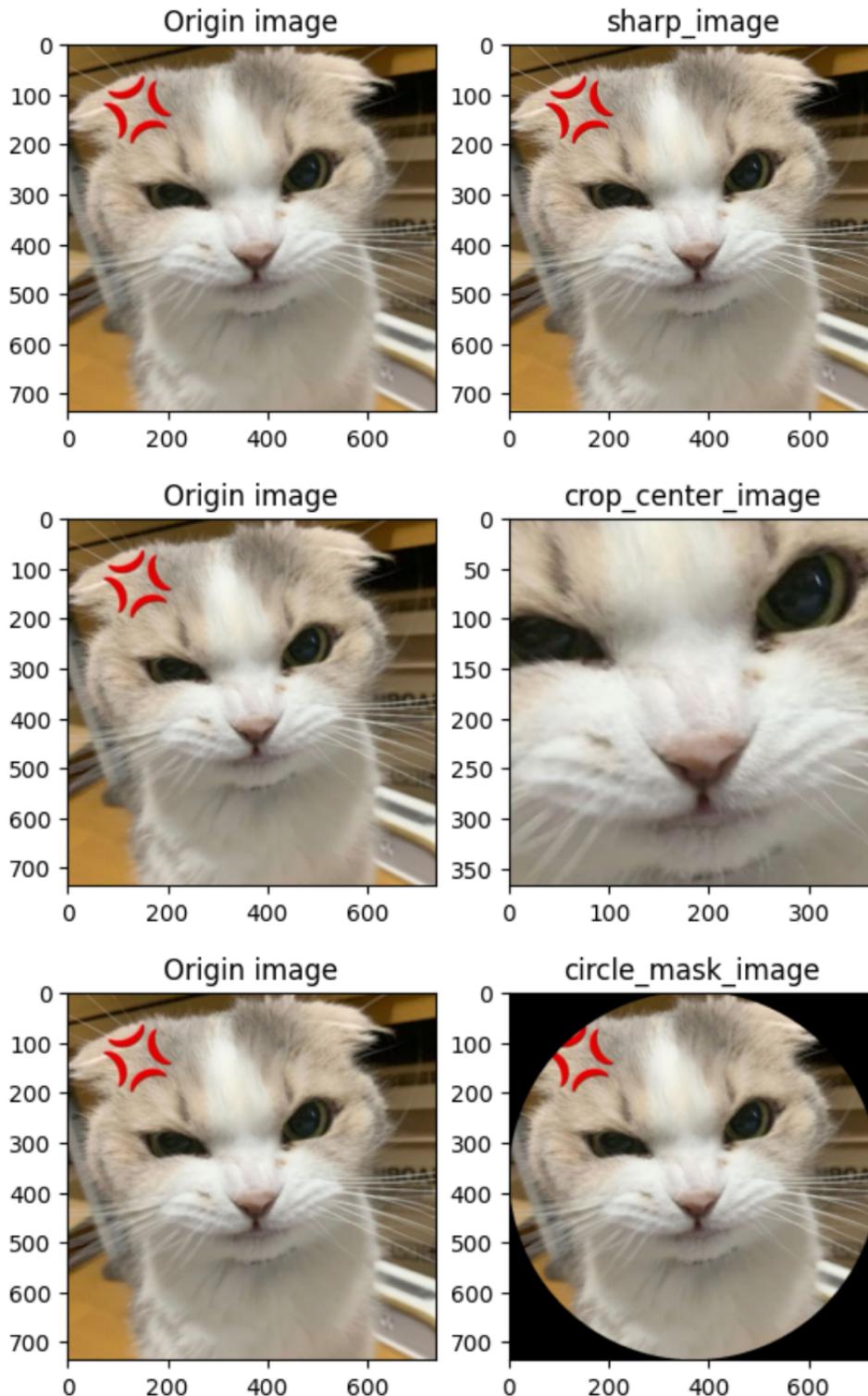
Hình 4: Hình ví dụ (736x736)

Kết quả thực hiện với hình 4









4 Nhận xét

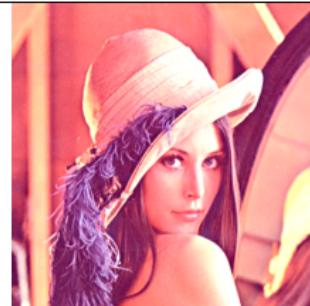
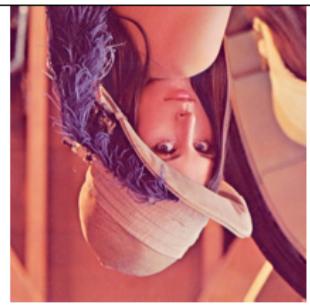
4.1 Nhận xét tổng quát

- Về tổng thể, các kết quả sau 2 lần thử thể hiện đúng với yêu cầu. Đối với yêu cầu cắt ảnh sang khung tròn, khung elip và trung tâm ảnh đầu vào cần là ảnh vuông thì mới thực thi tốt được.
- Thời gian thực hiện các yêu cầu cùng lúc tối đa cho ảnh kích thước 512x512 là không quá 15 giây.
- Việc làm mờ/sắc nét ảnh tốn thời gian cùng tài nguyên hơn rất nhiều so với các chức năng còn lại (vào khoảng 4 giây). Điều này là dễ hiểu vì một số chức năng như lật ảnh, cắt chỉ sử dụng những kĩ thuật được cung cấp sẵn trong python như là slicing,... còn các chứng năng tăng độ sáng, chuyển ảnh màu sang Sepia, xám thì cần nhiều phép tính toán: cộng, nhân ma trận, vector,... nên thời gian sẽ lâu hơn một ít.

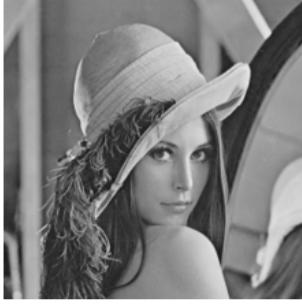
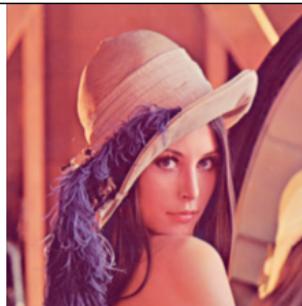
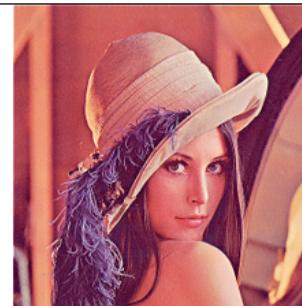
4.2 Hạn chế

Đối với các yêu cầu cắt ảnh thì chỉ thực hiện trên ảnh vuông, không thực thi tối ưu trên các ảnh có hình dạng khác.

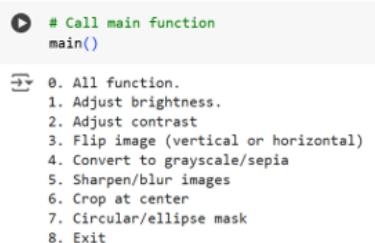
Chưa tối ưu trong code xử lý làm mờ ảnh, có thể nghiên cứu thêm một thuật toán khác để chạy nhanh hơn.

STT	Chức năng hàm	Mức độ hoàn thành	Ghi Chú	Ảnh kết quả
1	Thay đổi độ sáng	100%		
2	Thay đổi độ tương phản	100%		
3.1	Lật ảnh ngang	100%		
3.2	Lật ảnh dọc	100%		

Hình 5: Bảng đánh giá mức độ hoàn thành

4.1	RGB thành ảnh xám	100%	
4.2	RGB thành ảnh sepia	100%	
5.1	Làm mờ ảnh	100%	
5.2	Làm sắc nét ảnh	100%	

Hình 6: Bảng đánh giá mức độ hoàn thành

6	Cắt ảnh theo kích thước	90%	Với các kích thước lớn thì cắt sẽ cho ra không chính xác	
7.1	Cắt ảnh theo khung tròn	80%	Chỉ ảnh vuông	
7.2	Cắt ảnh theo khung elip	80%	Chỉ ảnh vuông	
8	Hàm main	100%		 <pre># Call main function main() → 0. All function. 1. Adjust brightness. 2. Adjust contrast. 3. Flip image (vertical or horizontal) 4. Convert to grayscale/sepia 5. Sharpen/blur images 6. Crop at center 7. Circular/ellipse mask 8. Exit</pre> <p>Origin image</p>

Hình 7: Bảng đánh giá mức độ hoàn thành

Tài liệu

- [1] Rotated ellipse. [https://math.stackexchange.com/questions/91132/how-to-get-the-limits-of-rotated-ellipse.](https://math.stackexchange.com/questions/91132/how-to-get-the-limits-of-rotated-ellipse)
- [2] Numpy document. 2008.
- [3] mmuratarat. Rgb to grayscale conversion. https://mmuratarat.github.io/2020-05-13-rgb_to_grayscale_formulas#:~:text=Average%20method&text=You%20just%20have%20to%20take, get%20your%20desired%20grayscale%20image.
- [4] OpenCV. Image transformations. https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html#void%20cvtColor%28InputArray%20src,%20OutputArray%20dst,%20int%20code,%20int%20dstCn%29.
- [5] StackOverflow. convert-image-to-sepia. <https://stackoverflow.com/questions/67587008/python-convert-image-to-sepia.>
- [6] Phan Thị Phương uyên. Lab 02 - image processing.
- [7] Đặng Ngọc Tiên. Project 02: Image processing. 2022.