

**ĐẠI HỌC KHOA HỌC TỰ NHIÊN – ĐHQG HCM
KHOA CÔNG NGHỆ THÔNG TIN**



**Toán Ứng Dụng Và Thống Kê
Cho Công Nghệ Thông Tin**

Báo Cáo

Đồ Án 2: Image Processing

Giảng viên:

Vũ Quốc Hoàng

Trợ giảng:

Phan Thị Phương Uyên

Lê Thanh Tùng

Nguyễn Văn Quang Huy

Sinh viên:

21127657 - Nguyễn Khánh Nhân

TPHCM, tháng 7 năm 2023

Mục Lục

I. Thông tin sinh viên	3
II. Thông tin đề án	3
III. Mức độ hoàn thành đề án	4
IV. Ý tưởng và chi tiết mã nguồn của từng chức năng trong chương trình.....	6
1. Thay đổi độ sáng của ảnh	6
2. Thay đổi độ tương phản của ảnh.....	7
3. Lật ảnh theo chiều ngang-dọc	8
4. Chuyển đổi ảnh màu RGB sang ảnh xám.....	9
5. Chuyển đổi ảnh màu RGB sang ảnh Sepia.....	11
6. Làm mờ/sắc nét ảnh	12
7. Cắt ảnh theo kích thước (cắt ở trung tâm)	17
8. Cắt ảnh theo khung hình tròn	18
9. Cắt ảnh theo khung 2 Elip chéo nhau	20
10.Các hàm khác	24
V. Kết quả từng chức năng	25
VI. Nhận xét và phân tích độ phức tạp thời gian của các chức năng	28
VII. Tài liệu tham khảo	29

I. Thông tin sinh viên

Họ tên: Nguyễn Khánh Nhân

MSSV: 21127657

Lớp: 21CLC02

Email: nknhan21@clc.fitus.edu.vn

II. Thông tin đề án





Trong đề án trước, ta đã biết rằng một bức ảnh thực chất là một ma trận hai chiều các điểm ảnh. Mỗi điểm ảnh có thể là một giá trị (ảnh xám) hoặc một vector (ảnh màu). Vì vậy dựa vào kiến thức về các phép tính trên ma trận, vector đã học thì trong đề án này ta sẽ thực hiện các chức năng xử lý ảnh cơ bản:

- Thay đổi (tăng) độ sáng của ảnh.
- Thay đổi (tăng) độ tương phản của ảnh.
- Lật ảnh theo chiều ngang-dọc.
- Chuyển đổi ảnh màu (RGB) thành ảnh xám/sepia.
- Làm mờ/sắc nét ảnh.
- Cắt ảnh theo kích thước (cắt ở trung tâm ảnh).
- Cắt ảnh theo khung hình tròn (tâm ở chính giữa ảnh).
- Cắt ảnh theo khung hai Elip chéo nhau (mỗi Elip tiếp xúc với cả bốn cạnh của bức ảnh).

III. Mức độ hoàn thành đồ án

STT	Chức năng	Ảnh gốc	Ảnh kết quả
1	Thay đổi (tăng) độ sáng của ảnh		
2	Thay đổi (tăng) độ tương phản của ảnh		
3	Lật ảnh theo chiều ngang		
4	Lật ảnh theo chiều dọc		

5	Chuyển ảnh màu RGB thành ảnh xám		
6	Chuyển ảnh màu RGB thành ảnh Sepia		
7	Làm mờ ảnh		
8	Làm sắc nét ảnh		
9	Cắt ảnh theo kích thước (cắt từ trung tâm)		

10	Cắt ảnh theo khung hình tròn		
11	Cắt ảnh theo khung hai Elip chéo nhau		

IV. Ý tưởng và chi tiết mã nguồn của từng chức năng trong chương trình

Thông tin khái quát về chương trình

- Mã nguồn của chương trình được viết hoàn toàn bằng ngôn ngữ lập trình python.
- Các thư viện sử dụng trong mã nguồn: numpy, PIL, matplotlib.
- Trình bày trên một file duy nhất: "21127657.ipynb".
- Chương trình có tổng cộng 12 hàm phụ và 1 hàm **MAIN**.

1. Thay đổi độ sáng của ảnh

• Ý tưởng:

- Để làm sáng màu 1 bức ảnh, ta chỉ đơn giản giá trị mỗi kênh màu của các điểm ảnh lên một lượng như nhau. Lượng tăng càng lớn thì bức ảnh càng sáng.

- **Cách triển khai:** Thực thi trong hàm `brighten_img()`
 - INPUT: Ảnh gốc (PIL.Image).
 - OUTPUT: Ảnh sau khi được tăng độ sáng (PIL.Image).
 - Chi tiết:

Ảnh gốc được chuyển thành ma trận 2 chiều các điểm ảnh `img_arr`.

Cộng vào 3 kênh màu của tất cả điểm ảnh một lượng bằng nhau để làm tăng độ sáng nhờ phép tính **element-wise**. Lưu ý sử dụng hàm `numpy.clip()` để giới hạn giá trị các kênh màu không vượt quá **[0, 255]**.

Lưu ma trận điểm ảnh mới vào `bright_img_arr` và chuyển ma trận này thành ảnh kết quả nhờ hàm `Image.fromarray()`.

Element-wise là thực hiện phép tính trên từng phần tử của các vector hoặc ma trận. Ví dụ, nếu bạn có hai vector `a` và `b` cùng kích thước, việc tính toán `a + b` **element-wise** nghĩa là thực hiện phép cộng giữa từng cặp phần tử tương ứng trong `a` và `b`, và trả về một vector mới chứa tổng của từng cặp phần tử đó.

2. Thay đổi độ tương phản của ảnh

- Ý tưởng:

Độ tương phản của ảnh là mức độ khác biệt giữa các mức độ sáng và tối trong một bức ảnh. Nó là độ lớn của khoảng cách giữa các điểm ảnh sáng nhất và tối nhất trong ảnh.

Để tăng độ tương phản thì ta cần phải tăng độ lớn khoảng cách giữa các điểm sáng và tối.

Để đơn giản hóa vấn đề, ta chỉ cần nhân giá màu của các điểm ảnh với 1 con

số thực dương thì độ lớn khoảng cách về "màu sắc" của chúng sẽ được tăng lên và đồng thời ảnh của của ta sẽ có độ tương phản rõ rệt.

- **Cách triển khai:** Thực thi trong hàm `adjust_contrast()`

- INPUT: Ảnh gốc (PIL.Image).
- OUTPUT: Ảnh sau khi được tăng độ sáng (PIL.Image).
- Chi tiết:

Ảnh gốc được chuyển thành ma trận 2 chiều các điểm ảnh `img_arr`.

Nhân ma trận này với 1 con số thực `CONTRAST` để tăng độ chênh lệch "màu sắc" giữa các điểm ảnh để tăng độ tương phản.

Sử dụng hàm `numpy.clip()` để tránh trường hợp màu điểm ảnh vượt quá `[0, 255]`.

Chuyển ma trận kết quả này thành ảnh nhờ hàm `Image.fromarray()`.

3. Lật ảnh theo chiều ngang-dọc

- **Ý tưởng:**

Một bức ảnh thực chất là một ma trận 2 chiều các điểm ảnh, vì vậy để thực hiện lật ngang hoặc dọc một bức ảnh, ta chỉ cần đảo cột hoặc dòng của ma trận điểm ảnh.

Lật ảnh dọc:

$$\begin{bmatrix} i_{11} & i_{12} & \dots & i_{1n} \\ \dots & \dots & \dots & \dots \\ i_{n1} & i_{n2} & \dots & i_{nn} \end{bmatrix} \rightarrow \begin{bmatrix} i_{n1} & i_{n2} & \dots & i_{nn} \\ \dots & \dots & \dots & \dots \\ i_{11} & i_{12} & \dots & i_{1n} \end{bmatrix}$$

Lật ảnh ngang:

$$\begin{bmatrix} i_{11} & i_{12} & \dots & i_{1n} \\ \dots & \dots & \dots & \dots \\ i_{n1} & i_{n2} & \dots & i_{nn} \end{bmatrix} \rightarrow \begin{bmatrix} i_{n1} & i_{n2} & \dots & i_{nn} \\ \dots & \dots & \dots & \dots \\ i_{11} & i_{12} & \dots & i_{1n} \end{bmatrix}$$

- **Cách triển khai:** Thực thi trong hàm `flip_image()`
 - INPUT: Ảnh gốc (PIL.Image).
 - OUTPUT: Ảnh sau khi được lật theo chiều ngang/dọc (PIL.Image).
 - Chi tiết:

Ảnh gốc chuyển thành ma trận hai chiều các điểm ảnh `img_arr`.

Nếu muốn lật ảnh theo chiều ngang/dọc, ta cần đảo cột/dòng của ma trận điểm ảnh nhờ kĩ thuật "**slicing**" trong python.

Nếu muốn lật ảnh theo chiều dọc, ta cần đảo dòng của ma trận điểm ảnh `img_arr[::-1, :]`, còn lật ảnh theo chiều ngang thì ta đảo cột của ma trận điểm ảnh `img_arr[:, ::-1]`.

Chuyển ma trận kết quả thành ảnh nhờ hàm `Image.fromarray()`.

4. Chuyển đổi ảnh màu RGB sang ảnh xám

- **Ý tưởng:**

Chuyển đổi giá trị 3 kênh màu thành cùng 1 giá trị duy nhất. Trọng số màu (đóng góp) của từng màu đã được quy định sẵn lần lượt là `x`, `y`, `z`. Và trong ảnh xám thì bộ trọng số này là như nhau ở cả 3 kênh màu.

Một điểm ảnh trong ảnh xám sẽ có duy nhất 1 kênh màu có giá trị:

$$GrayColor = xRED + yGREEN + zBLUE (x + y + z = 1)$$

Ta lập vector trọng số màu (color weight) để chứa trọng số màu của từng kênh

màu, tham khảo tại [đây](#):

$$ColorWeight = [0.25 \quad 0.6 \quad 0.15]$$

Nếu xem mỗi điểm ảnh RGB là vector chứa ba giá trị kênh màu thì để tính được giá trị màu của điểm ảnh xám (chỉ có 1 kênh màu) theo công thức ở trên, ta sẽ thực hiện tích vô hướng giữa vector điểm ảnh RGB với vector trọng số màu.

$$[R \quad G \quad B] \times [0.25 \quad 0.6 \quad 0.15] = [0.25 * R + 0.6 * G + 0.15 * B]$$

Trong đa số các ảnh màu, màu **xanh lá cây** (Green) có đóng góp cao hơn vào độ sáng của một điểm ảnh so với các kênh màu khác. Lý do là bởi vì mắt người có độ nhạy khác nhau với các kênh màu khác nhau. Theo một nghiên cứu về tâm lý học thị giác, mắt người có độ nhạy cao nhất với màu xanh lá cây, tiếp theo là màu đỏ, và cuối cùng là màu xanh dương. Do đó, màu xanh lá cây thường được sử dụng nhiều hơn trong các ứng dụng liên quan đến thị giác, bao gồm cả trong xử lý ảnh.

Ta thực hiện tính toán như vậy cho tất cả điểm ảnh của ảnh RGB thì sẽ thu được ma trận mới chứa các điểm ảnh xám.

- **Cách triển khai:** Thực thi trong hàm `rgb_to_gray()`
 - INPUT: Ảnh màu RGB (PIL.Image)
 - OUTPUT: Ảnh xám (PIL.Image)
 - Chi tiết:

Chuyển ảnh thành ma trận 2 chiều các điểm ảnh `img_arr`.

Dùng hàm `numpy.matmul()` để thực hiện tích vô hướng tất cả các vector điểm ảnh trong ma trận ảnh gốc với vector trọng số màu `COLOR_WEIGHT_GRAY` nhờ

element wise, sẽ thu được ma trận mới chứa các điểm ảnh chỉ có một kênh màu đại diện cho ảnh xám.

Gán ma trận kết quả này vào `gray_img`.

Chuyển ma trận kết quả thành ảnh nhờ hàm `Image.fromarray()`.

5. Chuyển đổi ảnh màu RGB sang ảnh Sepia

- Ý tưởng:

Tương tự như ảnh xám, một điểm ảnh **Sepia** sẽ có giá trị mới như cách tính ở ảnh xám 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.

Điểm khác biệt to lớn nhất giữa cách tính toán của ảnh **Sepia** so với ảnh **xám** là trọng số màu của một màu xác định là không còn như nhau ở ba kênh màu mà thay vào đó với mỗi kênh màu khác nhau thì trọng số màu của một màu sẽ khác nhau. Ma trận trọng số màu sẽ sử dụng được tham khảo tại [đây](#).

$$Color_Weights = \begin{bmatrix} 0.393 & 0.35 & 0.272 \\ 0.769 & 0.686 & 0.534 \\ 0.189 & 0.168 & 0.13 \end{bmatrix}$$

Trong đó dòng đầu tiên là trọng số màu của màu **RED** đối với ba kênh màu R-G-B, dòng 2 là trọng số màu của màu **GREEN** đối với 3 kênh màu và cuối cùng trọng số của màu **BLUE**. Lí do trọng số màu của **GREEN** luôn cao cũng giống như đã nêu ở ảnh xám.

Điểm ảnh Sepia sẽ có giá trị 3 kênh màu phụ thuộc vào giá trị 3 kênh màu của điểm ảnh RGB theo công thức sau:

$$new_red = 0.393 * R + 0.769 * G + 0.189 * B$$

$$new_green = 0.35 * R + 0.686 * G + 0.168 * B$$

$$new_blue = 0.272 * R + 0.534 * G + 0.13 * B$$

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.

- **Cách triển khai**: Thực hiện trong hàm `rgb_to_sepia()`

- INPUT: Ảnh màu RGB (PIL.Image)
- OUTPUT: Ảnh Sepia (PIL.Image)
- Chi tiết:

Chuyển ảnh thành ma trận điểm ảnh `img_arr`.

Dùng hàm `numpy.matmul()` để thực hiện nhân tất cả vector điểm ảnh RGB với ma trận trọng số màu `COLOR_WEIGHT_SEPIA`.

Ta sẽ thu được ma trận mới chứa các điểm ảnh Sepia `sepia_array`.

Chuyển ma trận điểm ảnh Sepia thành ảnh Sepia nhờ hàm `Image.fromarray()`.

6. Làm mờ/sắc nét ảnh

- **Ý tưởng**

Để làm mờ/sắc nét ảnh, ta cần tính toán lại giá trị màu của toàn bộ điểm ảnh trong ảnh gốc.

Ta sẽ thực hiện phân bố lại màu sắc của một 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**. Kích thước của ma trận là một số lẻ vì với ma trận lẻ ta sẽ luôn có vị trí trung tâm của ma trận. Các giá trị của bộ lọc này biểu thị cho trọng số màu của các điểm ảnh mà giá trị đó được "**đặt lên**". Khi áp dụng **Gaussian Kernel** lên ảnh, mỗi điểm ảnh sẽ được thay thế bằng trung bình của các điểm ảnh xung quanh nó, với trọng số lớn nhất tại điểm đó. Các điểm ảnh càng xa điểm đang xét, trọng số càng nhỏ.

Để tính toán lại giá trị màu cho các điểm ảnh khi làm mờ/sắc nét, ta sẽ thực hiện phép **tích chập (convolution)** giữa ma trận điểm ảnh gốc và **Gaussian Kernel**.

Tích chập hay **convolution** là một phép tính quen thuộc trong **xử lý ảnh**. Với mỗi phần tử x_{ij} trong ma trận ảnh gốc **X** lấy ra một ma trận con (**sub_matrix**) bằng kích thước của Kernel **W** sao cho x_{ij} làm trung tâm, gọi ma trận con này là ma trận **A**. Thực hiện phép tính **tích chập** của ma trận **A** và **W** sẽ thu được giá trị màu của điểm ảnh kết quả. Gán kết quả này vào y_{ij} của ma trận kết quả **Y**. Thực hiện như vậy cho tất cả các điểm ảnh x_{ij} sẽ thu được các điểm ảnh y_{ij} tương ứng.

Tuy nhiên thuật toán sẽ vướng phải vấn đề khi tính toán cho các điểm ảnh x_{ij} ở ngoài rìa khi 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**. Như tên gọi của nó, **padding** là các điểm ảnh có giá trị 0 được thêm vào các cạnh của ma trận gốc **X** sao cho luôn đảm bảo sẽ tìm được ma trận **A** 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.

Tùy vào mục đích làm mờ/sắc nét ảnh nhiều hay ít mà ta sẽ có bộ lọc **Gaussian Kernel** khác nhau về giá trị lẫn kích thước. Kích thước bộ lọc càng lớn thì sẽ hiệu quả hơn nhưng đổi lại mất rất nhiều thời gian cũng như bộ nhớ. Để tính các trị

trọng số trong **Gaussian Kernel** ta sử dụng hàm **Gauss**:

$$G(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-x^2 - y^2}{2\sigma^2}}$$

Với x, y là khoảng cách của vị trí đang xét trong ma trận tới điểm chính giữa của ma trận, σ là độ lệch chuẩn trong phân phối **Gauss**.

Ví dụ về 2 ma trận **Gaussian Kernel** cơ bản thường được dùng để **làm mờ**:

$$Gaussian_Kernel_3 = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$Gaussian_Kernel_5 = \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Ví dụ về ma trận dùng để **làm sắc nét** ảnh:

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

Để hiểu rõ hơn, ta sẽ thực hiện việc **làm mờ** với 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} trong ma trận gốc, lấy ra ma trận con A sao cho x_{11} làm trung tâm ma trận A . Thực hiện tích chập giữa A và bộ lọc Gaussian Kernel `W`.

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

Bước 3: Lập lại bước 2 cho tất cả các phần tử trong ma trận X ban đầu.

$$X \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow A \begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & 0 \\ 3 & 4 & 0 \end{bmatrix} \times W \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \frac{21}{16} \rightarrow Y \begin{bmatrix} \frac{9}{8} & \frac{21}{16} \\ \dots & \dots \end{bmatrix}$$

$$X \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow A \begin{bmatrix} 0 & 1 & 2 \\ 0 & 3 & 4 \\ 0 & 0 & 0 \end{bmatrix} \times W \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \frac{3}{2} \rightarrow Y \begin{bmatrix} \frac{9}{8} & \frac{21}{16} \\ \frac{3}{2} & \dots \end{bmatrix}$$

$$X \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow A \begin{bmatrix} 1 & 2 & 0 \\ 3 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix} \times W \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \frac{7}{4} \rightarrow Y \begin{bmatrix} \frac{9}{8} & \frac{21}{16} \\ \frac{3}{2} & \frac{7}{4} \end{bmatrix}$$

Bước 4: Kết thúc và Y là ma trận đại diện cho ảnh sau khi được làm mờ. Quá trình làm sắc nét ảnh cũng diễn ra tương tự như vậy.

- **Cách triển khai:** Thực hiện trong hàm `apply_filter()`

- INPUT: Ảnh gốc (PIL.Image)
- OUTPUT: Ảnh sau khi được làm mờ/sắc nét (PIL.Image)
- Chi tiết:

Dựa vào biến `type` để xác định mục đích là làm mờ hay làm sắc nét để lấy bộ lọc `Kernel` cho phù hợp thông qua hàm `get_kernel()`.

Chuyển ảnh thành ma trận 2 chiều các điểm ảnh `img_arr`.

Xác định kích thước của ảnh cũng như của mảng lọc `kernel`.

Xác định kích thước (số lượng) `padding` cần thêm vào ma trận ảnh gốc.

Tạo ma trận `b` có kích thước đủ để chứa tất cả các `padding` cũng như các điểm ảnh trong ảnh gốc, 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.

Sử dụng 2 vòng lặp `for` để duyệt hết tất cả các điểm ảnh gốc. Với điểm ảnh x_{ij} , nhờ kĩ 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 này.

Trong phép tính toán **tích chập**, `kernel` có kích thước là `(kernel_height x kernel_width)`, trong khi ma trận ảnh có 3 chiều (`height, width, channels`). Do đó, khi ta muốn nhân hai ma trận này với nhau, chúng ta cần phải thêm một chiều mới vào mảng `kernel` để nó có kích thước là `(kernel_height, kernel_width, 1)`. Điều này có thể thực hiện bằng cách sử dụng `numpy.newaxis` trong NumPy. Chiều mới này sẽ cho phép NumPy broadcast ma trận `kernel` để có cùng kích thước với ma trận ảnh, để có thể thực hiện phép nhân ma trận giữa chúng.

Kết quả của phép nhân trên sẽ tạo ra một ma trận mới với kích thước `(kernel_height, kernel_width, 1)`, bước cuối cùng của 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. Để thực hiện ta sử dụng hàm `numpy.sum()`. **Tuy nhiên**, cần lưu ý là ta thực hiện tính tổng tất cả phần tử (điểm ảnh) của ma trận (bao gồm tất cả các cột và hàng) nên cần gán tham số `axis =`

(0, 1), vì **axis (chiều)** = 0 và 1 là nơi chứa các điểm ảnh, còn **axis = 2** là chiều chứa giá trị 3 kênh màu của từng điểm ảnh (như đã nêu ở **Đồ án 1**).

Đối với việc làm sắc nét, để tránh giá trị màu vượt quá 255 thì ta sử dụng **numpy.clip()** để đảm bảo giá trị màu của điểm ảnh sẽ không vượt quá 255.

Ta gán kết quả của phép **tích chập** là điểm ảnh mới vào ma trận kết quả **result** tại vị trí giống với vị trí điểm ảnh gốc mà ta đang xét.

Chuyển ma trận kết quả **result** thành ảnh nhờ hàm **Image.fromarray()**, ảnh này chính là ảnh sau khi được làm mờ/sắc nét.

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

- **Ý tưởng:**

Như đã biết, 1 tấm ảnh là 1 ma trận 2 chiều các điểm ảnh, vì vậy để có thể cắt được 1 bức ảnh ở trung tâm ta chỉ đơn giản xóa đi các dòng, cột ở ngoài rìa của ma trận.

Tùy vào kích thước ảnh sau khi cắt mà số dòng, cột bị xóa đi sẽ khác nhau.

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.

- **Cách triển khai:** Thực hiện trong hàm **crop_image()**

- **INPUT:** Ảnh gốc (PIL.Image).
- **OUTPUT:** Ảnh có được từ việc cắt ảnh gốc (từ trung tâm) (PIL.Image).
- **Chi tiết:**

Chuyển ảnh thành ma trận `numpy.array()`.

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`.

Việc thực hiện được như vậy sẽ nhờ kĩ thuật `slicing` trong python.

`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 $\frac{size}{4}$ đến $\frac{3size}{4}$.

Chuyển ma trận kết quả thành ảnh nhờ hàm `Image.fromarray()`.

8. Cắt ảnh theo khung hình tròn

- Ý tưởng:

Mỗi điểm ảnh có cho mình một tọa độ (i, j) vì vậy ta có thể xác định được khoảng cách giữa chúng.

Để thực hiện cắt ảnh theo khung hình tròn bán kính R với tâm ở chính giữa bức ảnh. Ta sẽ sử dụng phương trình đường tròn đã học ở Trung học Phổ Thông để xác định các điểm ảnh nào nằm bên trong hình tròn, các điểm ảnh nằm bên ngoài đường tròn sẽ được thay đổi giá trị màu thành màu đen $(0, 0, 0)$.

Phương trình đường tròn có tâm tại (a, b) , bán kính R :

$$(C): (x - a)^2 + (y - b)^2 = R^2$$

Điều kiện để một điểm ảnh có tọa độ (i, j) nằm bên trong đường tròn:

$$(i - a)^2 + (j - b)^2 \leq R^2$$

Để đảm bảo hình tròn tiếp xúc với 4 cạnh bức ảnh thì đường kính **2R** sẽ bằng với kích thước một cạnh của ảnh.

Như vậy, sau khi thực hiện thì ảnh của ta sẽ được cắt theo khung hình tròn có bán kính R và tâm tại chính giữa ảnh.

- **Cách triển khai:**

- INPUT: Ảnh gốc (PIL.Image).
- OUTPUT: Ảnh sau khi được cắt theo khung tròn (PIL.Image).
- Chi tiết:

Chuyển ảnh thành ma trận `numpy.array()`.

Xác định kích thước ảnh để từ đó tính được bán kính đường tròn sao cho đảm bảo đường tròn sẽ tiếp xúc với 4 cạnh của ảnh.

Xác định tọa độ tâm của đường tròn là `(center_index, center_index)`.

Mỗi điểm đều có cho mình 1 tọa độ hàng và 1 tọa độ cột. Vì vậy ta sẽ lập hai ma trận dùng để chứa các tọa độ này. Trong đó, `matrix_index_i` là ma trận dùng để lưu trữ tọa độ hàng của tất cả các điểm ảnh vì vậy các phần tử trên cùng hàng `i` của ma trận này sẽ có cùng giá trị là `i`. Còn `matrix_index_j` là ma trận dùng để lưu trữ tọa độ cột của tất cả điểm ảnh vì vậy các phần tử trên cùng cột `j` của ma trận này sẽ có cùng giá trị là `j`.

Ma trận `matrix_index_j` được tạo ra nhờ hàm `numpy.tile()`. Hàm này được sử dụng để sao chép và ghép nối các ma trận lại với nhau theo tham số `reps`. Trong trường hợp này, `np.arange(size)` là một ma trận 1 chiều chứa các giá trị từ 0 tới `size - 1`, và `reps=(size, 1)` cho biết rằng ma trận này sẽ được sao chép `size` lần theo chiều 0 (theo hàng) và 1 lần theo chiều 1 (theo cột)

Ma trận `matrix_index_i` chính là ma trận chuyển vị của `matrix_index_j` nên việc tạo ra ma trận này thì chỉ cần thực hiện phép chuyển vị lên `matrix_index_j`.

Sử dụng mảng `boolean (mask)` trong `numpy` có cùng kích thước với ma trận điểm ảnh để đánh dấu các điểm ảnh nằm bên ngoài đường tròn. Cụ thể, điểm ảnh nằm bên ngoài đường tròn nếu bình phương khoảng cách từ điểm ảnh đó tới tâm của đường tròn lớn hơn bình phương bán kính đường tròn $((matrix_index_i - center_index)^2 + (matrix_index_j - center_index)^2 \leq (radius^2))$.

Sau đó, `img[masks] = [0, 0, 0]` sẽ gán màu đen cho các điểm ảnh nằm bên ngoài đường tròn. Các điểm ảnh này được xác định bởi mảng `boolean masks`.

Chuyển ma trận kết quả thành ảnh nhờ hàm `Image.fromarray()`.

9. Cắt ảnh theo khung 2 Elip chéo nhau

- Ý tưởng:

Để cắt ảnh theo khung 2 hình Elip nghiêng chéo lên nhau thì cũng giống như việc cắt ảnh theo khung tròn. Tuy nhiên điểm khác biệt ở đây là phương trình của 2 đường Elip này khác và phức tạp hơn nhiều so với đường tròn.

Như đã học ở Trung học Phổ Thông, một elip sẽ có phương trình:

$$(E): \frac{(x - m)^2}{a^2} + \frac{(y - n)^2}{b^2} = 1$$

Với `(m, n)` là tọa độ tâm Elip; `(a, b)` lần lượt là bán trục lớn, bán trục nhỏ của Elip.

Ở kiến thức toán lớp 10, ta có học đến **phép quay**, vì thế để elip có 2 trục nằm trên 2 đường chéo của bức ảnh (hình vuông), ta cần thực hiện quay Elip (E) một góc 45 độ hoặc 135 độ. Như vậy hai hình Elip của chúng ta, một hình sẽ quay góc 45 độ quanh tâm và hình còn lại quay góc 135 độ quanh tâm.

Công thức tổng quát của Elip khi quay góc α quanh tâm:

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

Tuy nhiên vấn đề khó nhất 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 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:

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

Do Elip có tính chất đối xứng nên chỉ cần tìm điều kiện a, b để Elip tiếp xúc với một cạnh thì cũng 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 mục tiêu của ta bây giờ sẽ tìm a, b sao cho chỉ có duy nhất **1 nghiệm x** thỏa **phương trình (I)** với $y = c$. Thay $y = c$ vào (I) thì (I) sẽ trở thành phương trình biến x với hai tham số a, b :

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

Ta thấy $F(x)$ là một **Parabol**, vì vậy 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.

$$\begin{aligned}
&\Leftrightarrow F\left(\frac{c(\frac{1}{a^2} + \frac{1}{b^2})}{2(\frac{1}{2a^2} + \frac{1}{2b^2})}\right) = 0 \\
&\Leftrightarrow \left(\frac{1}{2a^2} + \frac{1}{2b^2}\right)\left(\frac{c(\frac{1}{a^2} + \frac{1}{b^2})}{2(\frac{1}{2a^2} + \frac{1}{2b^2})}\right)^2 - c\left(\frac{1}{a^2} - \frac{1}{b^2}\right)\left(\frac{c(\frac{1}{a^2} + \frac{1}{b^2})}{2(\frac{1}{2a^2} + \frac{1}{2b^2})}\right) + c^2\left(\frac{1}{2a^2} + \frac{1}{2b^2}\right) - 1 = 0 \\
&\Leftrightarrow \left(\frac{-c^2}{2}\right)\frac{(\frac{1}{a^2} - \frac{1}{b^2})^2}{\frac{1}{a^2} + \frac{1}{b^2}} + \left(\frac{c^2}{2}\right)\left(\frac{1}{a^2} + \frac{1}{b^2}\right) - 1 = 0 \\
&\Leftrightarrow \left(\frac{c^2}{2}\right)\left[\frac{1}{a^2} + \frac{1}{b^2} - \frac{(\frac{1}{a^2} - \frac{1}{b^2})^2}{\frac{1}{a^2} + \frac{1}{b^2}}\right] = 1 \\
&\Leftrightarrow \frac{\frac{4}{a^2b^2}}{\frac{1}{a^2} + \frac{1}{b^2}} = \frac{2}{c^2} \\
&\Leftrightarrow \frac{4}{a^2 + b^2} = \frac{2}{c^2} \\
&\Leftrightarrow a^2 + b^2 = 2c^2
\end{aligned}$$

Nếu ta xem tâm bức ảnh trùng với gốc tọa O thì c sẽ bằng một nửa kích thước cạnh của bức ảnh, vì vậy ta sẽ đi đến kết luận rằng: **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 hoặc 135 độ, đồng thời độ dài hai trục a, b (a > b) của Elip phải luôn thỏa:**

$$a^2 + b^2 = \frac{cạnh^2}{2}$$

Ta có thể chọn a, b tùy ý nhưng vẫn phải đảm bảo a lớn hơn b đồng thời thỏa tính chất trên. Cụ thể ở đây, ta sẽ chọn:

$$a^2 = \frac{3}{4}cạnh^2, b^2 = \frac{1}{4}cạnh^2$$

Cuối cùng, Elip tâm **(m, n)** nghiêng góc **alpha** chúng ta đang tìm sẽ có phương trình:

$$(E): \frac{\left((x-m)\cos(\alpha) - (y-n)\sin(\alpha)\right)^2}{\frac{3}{4}cạnh^2} + \frac{\left((x-m)\sin(\alpha) + (y-n)\cos(\alpha)\right)^2}{\frac{1}{4}cạnh^2} = 1$$

Khi đã có được phương trình Elip, ta sẽ tìm được các điểm ảnh nào nằm bên trong, bên ngoài Elip. Với một điểm ảnh có tọa độ (i, j) thì điều kiện để điểm ảnh này nằm bên trong Elip là:

$$\frac{\left((i-m)\cos(\alpha) - (j-n)\sin(\alpha)\right)^2}{\frac{3}{4}cạnh^2} + \frac{\left((i-m)\sin(\alpha) + (j-n)\cos(\alpha)\right)^2}{\frac{1}{4}cạnh^2} \leq 1$$

Các điểm ảnh nằm bên ngoài cả Elip nghiêng 45 độ và Elip nghiêng 135 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.

- **Cách triển khai:**

- INPUT: Ảnh gốc (PIL.Image).
- OUTPUT: Ảnh sau khi cắt theo khung 2 Elip chéo nhau (PIL.Image).
- Chi tiết:

Chuyển ảnh thành ma trận `numpy.array()` và lấy kích thước ảnh `size`.

Lưu tọa độ tâm Elip vào `center_index`.

Như đã chứng minh ở trên thì ta sẽ chọn trục nhỏ, bé của ma trận sao cho vẫn thỏa điều kiện đã chứng minh.

Giống y như việc **cắt ảnh theo khung tròn**, ta vẫn sẽ cần hai ma trận lưu trữ tọa độ dòng, cột của tất cả các điểm ảnh `matrix_index_i` và `matrix_index_j`. Cách tạo ra và ý nghĩa vẫn giống y hệt ở phần **cắt ảnh theo khung tròn**.

Hàm `ellipse_alpha_degrees()` sẽ nhận các tham số: hai trục Elip, góc quay, ma trận chứa các tọa độ dòng, cột. Hàm này sẽ dựa vào công thức Elip mà ta đã lập ra ở phần **Ý tưởng** để thực hiện tính toán và trả về mảng đánh dấu `boolean (mask)` các điểm ảnh nào nằm bên trong, bên ngoài Elip. Nếu nằm trong Elip sẽ được

đánh dấu là **False**, ngược lại là **True**.

`masks1` sẽ là ma trận **boolean** dùng để đánh dấu các điểm ảnh nằm bên ngoài Elip nghiêng góc 45 độ theo chiều kim đồng hồ và `masks2` là ma trận dùng để đánh dấu các điểm ảnh nằm bên ngoài Elip nghiêng 135 độ theo chiều kim đồng hồ.

Cuối cùng, ta cần phải xác định những điểm ảnh nào nằm ở ngoài cả 2 Elip này và tạo ra ma trận đánh dấu các điểm ảnh ấy. Để thực hiện được ta sẽ sử dụng hàm `numpy.logical_and()` để thực hiện toán tử logic **AND** giữa các phần tử tương ứng của hai ma trận `masks1` và `masks2`.

`masks` sẽ là ma trận đánh dấu những điểm ảnh nào nằm ngoài cả hai Elip, chúng sẽ được đánh dấu là **TRUE**, còn nếu điểm ảnh nào nằm trong cả hai Elip hoặc một Elip bất kì sẽ được đánh dấu là **FALSE**.

`img_arr[masks] = [0, 0, 0]` sẽ thay đổi giá trị màu của các điểm ảnh nằm ngoài hai Elip thành màu đen.

Chuyển ma trận kết quả về lại thành ảnh nhờ hàm `Image.fromarray()`.

10. Các hàm khác

- Hàm `show_save_img()`

INPUT: `input_file` (str): tên file ảnh gốc, `src_img` (PIL.Image): ảnh gốc, `result_img` (PIL.Image): ảnh kết quả, `func` (str): chức năng thực hiện, `run_time` (float): thời gian thực thi chức năng.

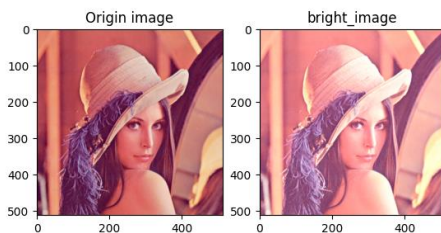
Chi tiết: Hàm này sẽ thực hiện hiển thị ảnh kết quả của chức năng tương ứng bằng hàm `im_show()` của thư viện `matplotlib.pyplot` đồng thời lưu ảnh kết quả dưới định dạng **png**.


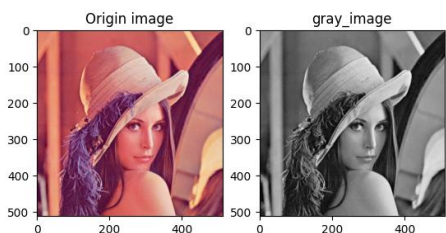
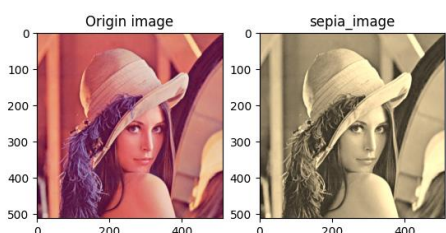
Tuy nhiên, khi hiển thị ảnh **xám (grayscale image)**, các điểm ảnh trên ảnh chỉ có giá trị độ **xám (grayscale value)** từ **0** đến **255**, trong đó giá trị **0** tương ứng với màu **đen** và giá trị **255** tương ứng với màu **trắng**. Vì vậy, khi hiển thị ảnh xám, chúng ta muốn hiển thị các điểm ảnh với các mức xám khác nhau phù hợp với giá trị độ xám của chúng. Trong thư viện **matplotlib**, để hiển thị ảnh xám dưới dạng mức xám, ta có thể sử dụng **bản đồ màu (colormap) 'gray'** bằng cách truyền giá trị **'gray'** vào tham số **cmap** của hàm **imshow()**. Bản đồ màu **'gray'** được thiết kế để có các mức xám tương ứng với các giá trị độ xám từ 0 đến 255, giúp hiển thị ảnh xám một cách chính xác và dễ nhìn.

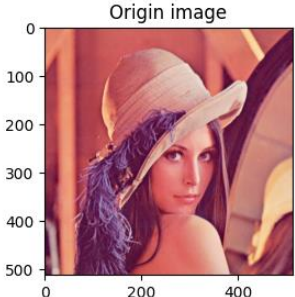

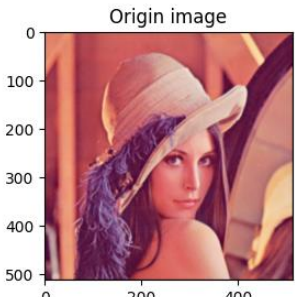
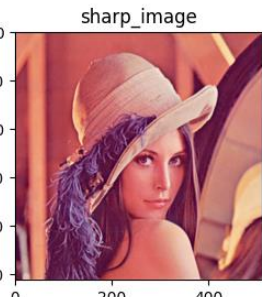
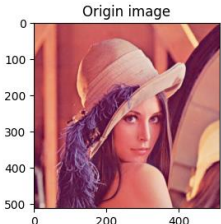

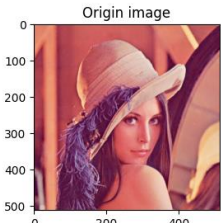

- Hàm **main()**

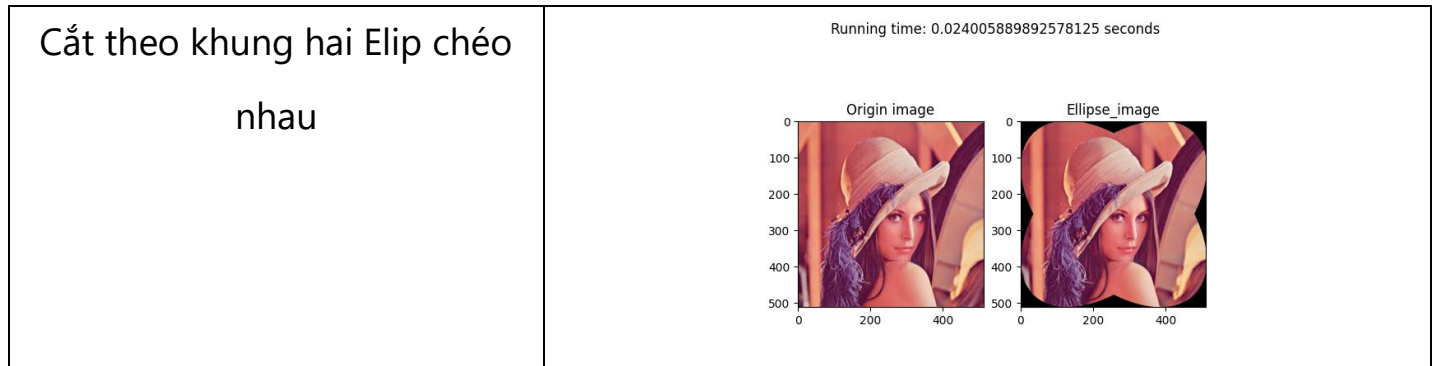
Thực hiện nhận vào các input: tên file ảnh cần xử lí, chức năng cần xử lí (1->8), nếu nhập **0** thì sẽ thực hiện **tất cả chức năng**. Ứng với mỗi chức năng sẽ thực hiện xử lí cho ra ảnh kết quả rồi hiển thị lên màn hình và lưu ảnh dưới định dạng **png**, bên cạnh đó cũng thực hiện đo thời gian thực thi của chứng năng và hiển thị thông số này cùng với ảnh kết quả.

V. Kết quả từng chức năng

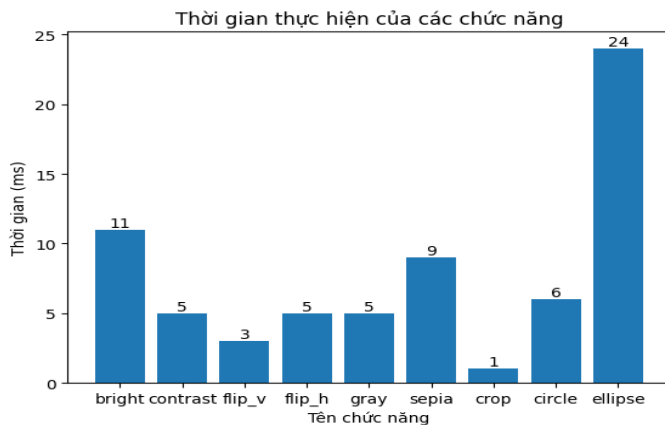
Chức năng	Kết quả
Tăng độ sáng ảnh	<p>Running time: 0.011003494262695312 seconds</p> 

<p>Tăng độ tương phản</p>	<p>Running time: 0.00500178337097168 seconds</p> <div data-bbox="857 210 1300 441">  </div>
<p>Lật ảnh ngang</p>	<p>Running time: 0.0050008296966552734 seconds</p> <div data-bbox="857 567 1300 798">  </div>
<p>Lật ảnh dọc</p>	<p>Running time: 0.0030002593994140625 seconds</p> <div data-bbox="857 924 1300 1155">  </div>
<p>Chuyển ảnh màu sang xám</p>	<p>Running time: 0.005000591278076172 seconds</p> <div data-bbox="857 1281 1300 1512">  </div>
<p>Chuyển ảnh màu sang Sepia</p>	<p>Running time: 0.009002208709716797 seconds</p> <div data-bbox="857 1638 1300 1869">  </div>

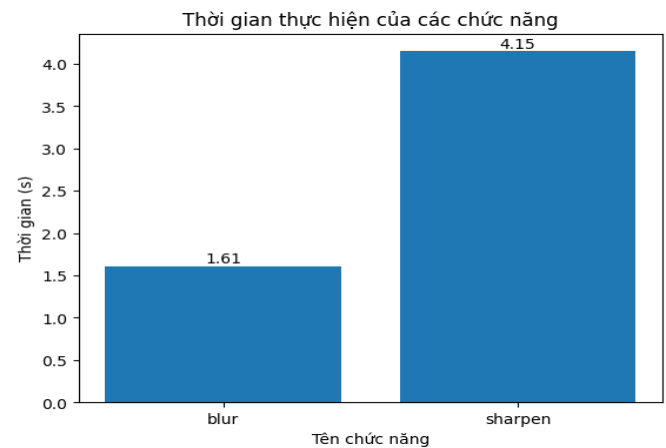
Làm mờ ảnh	<p>Running time: 1.610363483428955 seconds</p> <div>   </div>
Làm sắc nét ảnh	<p>Running time: 4.300969362258911 seconds</p> <div>   </div>
Cắt theo kích thước (từ trung tâm)	<p>Running time: 0.0010013580322265625 seconds</p> <div>   </div>
Cắt theo khung hình tròn	<p>Running time: 0.006001949310302734 seconds</p> <div>   </div>



VI. Nhận xét về sự chênh lệch thời gian của các chức năng



Biểu đồ thể hiện running time của các chức năng theo milli giây (ms)



Biểu đồ thể hiện running time của các chức năng theo giây (s)

- Từ 2 biểu đồ cột trên ta thấy được việc làm mờ/sắc nét ảnh tốn thời gian hơn rất nhiều so với các chức năng còn lại. Tuy nhiên, ta đã thấy các hàm thực chức năng tuy có độ phức tạp thời gian θ giống nhau nhưng khi thực tế thời gian thực thi lại có sự chênh lệch, điều này là dễ hiểu vì một số chức năng như **lật ảnh, cắt ảnh** không cần quá nhiều phép tính toán mà chỉ sử dụng những kĩ thuật được cung cấp sẵn trong python như là **slicing**,... còn các chức 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.
- Ba chức năng cắt ảnh có **sự chênh lệch thời gian thực thi** là vì việc cắt ảnh theo

khung hình chữ nhật cần phải tính toán quá phức tạp mà chỉ dùng kĩ thuật **slicing** là đủ. Còn **cắt theo khung hình tròn**, ta phải tính toán dựa theo công thức hình tròn đã học để xác định các điểm ảnh bên ngoài và bên trong hình tròn để cắt. Và tất nhiên **cắt theo 2 hình Elip chéo nhau** còn lâu hơn nhiều vì đơn giản là công thức của Elip nghiêng đòi hỏi rất nhiều phép tính và vô cùng phức tạp hơn so với đường tròn.

- Cuối cùng là việc **làm mờ** và **làm sắc nét** tại sao có sự chênh lệch thời gian như vậy mặc dù cách thức thực hiện cũng như độ phức tạp thời gian là như nhau? Vì trong quá trình **làm mờ**, ta thực hiện "**chia sẻ**" giá trị màu của một điểm ảnh cho các điểm ảnh lân cận nên không lo việc giá trị màu vượt quá 255, tuy nhiên việc **làm sắc nét** thì ngược lại nên phải sử dụng hàm **numpy.clip()** để đảm bảo giá trị màu của điểm ảnh sau khi thực hiện phép tích chập để **làm sắc nét** ảnh bị vượt quá 255. Mỗi lần thực hiện phép chập cho một điểm ảnh thì ta phải xài **numpy.clip()** là nguyên nhân dẫn đến việc làm sắc nét **lâu gần gấp 3 lần** so với làm mờ.

VII. Tài liệu tham khảo

- Cách không để giá trị điểm ảnh vượt quá [0,255]
- Cách chuyển ảnh màu sang xám/sepia
- Làm mờ ảnh sử dụng màng lọc Gaussian Kernel
- Wiki cho việc làm mờ ảnh sử dụng Gaussian Kernel
- Nguyên lí hoạt động của việc làm mờ bằng màng lọc
- Sử dụng tích chập (convolution) trong xử lí ảnh
- Công thức quay một điểm quanh tâm một góc bất kì

HẾT