ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Khoa Công Nghệ Thông Tin

----------***----------

BÁO CÁO

Project 2 – Image Processing

Tác giả
Nguyễn Thế Hiển - 22127107
Lớp
22CLC08

Môn học
Toán ứng dụng và thống kê
Công Nghệ Thông Tin

Giảng viên hướng dẫn
Vũ Quốc Hoàng
Nguyễn Văn Quang Huy
Nguyễn Ngọc Toàn
Phan Thị Phương Uyên

Thành phố Hồ Chí Minh – 2024

Mục Lục

1. Đánh giá độ hoàn thiện

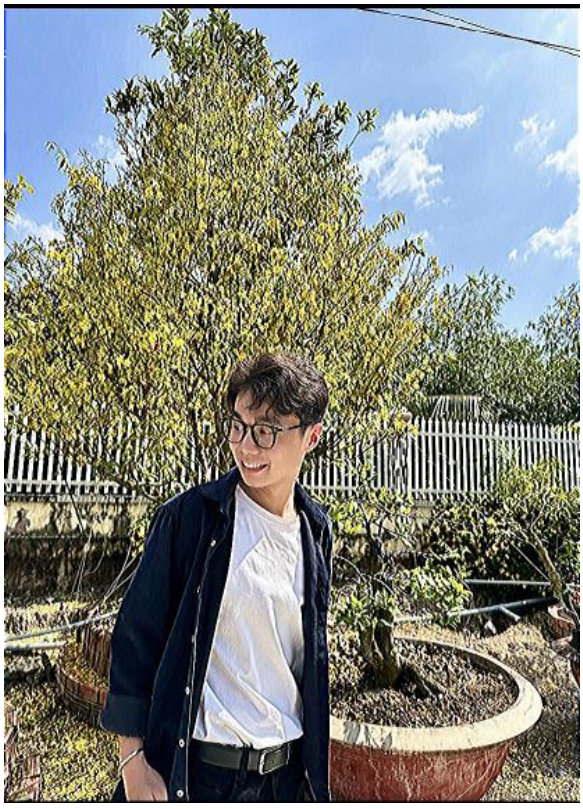
Ảnh test

STT	Chức năng/ Hàm	Mức độ hoàn thành	Ảnh kết quả
1	Thay đổi độ sáng	100%	

2	Thay đổi độ tương phản	100%	
3.1	Lật ảnh ngang	100%	

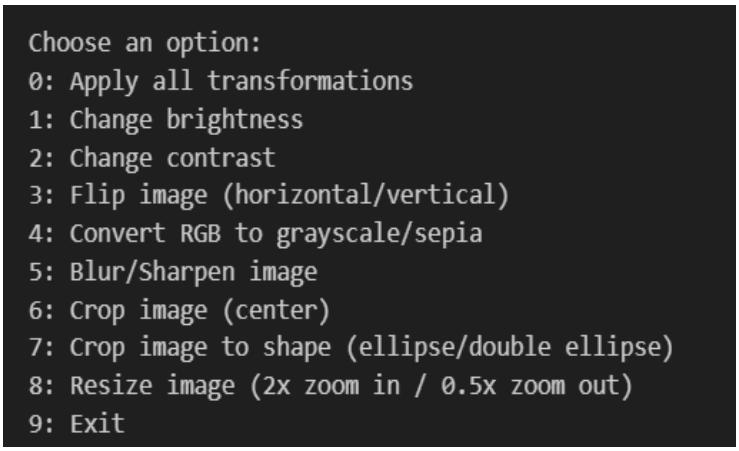
3.2	Lật ảnh dọc	100%	
4.1	RGB thành ảnh xám	100%	

4.2	RGB thành ảnh sepia	90%	
5.1	Làm mờ ảnh (màu, xám)	100%	Làm mờ ảnh màu 

			Làm mờ ảnh xám 
5.2	Làm sắc nét ảnh (màu, xám)	100%	Làm sắc nét ảnh màu 

			Làm sắc nét ảnh xám 
6	Cắt ảnh theo kích thước	100%	

7.1	Cắt ảnh theo khung tròn	100%	
7.2	Cắt ảnh theo khung elip	70%	

8	Hàm main	100%	
9	Phóng to/ Thu nhỏ 2x	90%	Phóng to 2x 

			Thu nhỏ 0.5x 

2. Ý tưởng thực hiện

Trong đồ án này chúng ta sử dụng với sự trợ giúp của các thư viện: PIL, NumPy, Matplotlib để thực hiện các chức năng xử lí ảnh trong đồ án này.

PIL (Python Imaging Library):

+ Mục đích: Mở, thao tác và lưu trữ nhiều định dạng ảnh khác nhau.
+ Các hàm sử dụng: Image.open(), Image.save(), ImageDraw.Draw(), Image.tranpose(), Image.crop(), Image.resize().

NumPy:

+ Mục đích: Thực hiện các thao tác trên mảng đa chiều và các hàm toán học hiệu quả.
+ Các hàm sử dụng: np.array(), np.clip(), np.mean(), np.tile(), np.logical_or(), np.cos(), np.sin().

Matplotlib:

+ Mục đích: Vẽ đồ thị và trực quan hóa dữ liệu.
+ Các hàm sử dụng: plt.imshow(), plt.axis(), plt.show()

- **change_brigthness:** Chức năng thay đổi độ sáng cho ảnh. Đối với mỗi điểm ảnh (pixel), ta nhân giá trị của các kênh màu (red, green, blue) với một hệ số factor. Sau đó, sử dụng hàm np.clip để giới hạn giá trị nằm trong khoảng 0-255.
- **change_contrast:** Chức năng thay đổi độ tương phản cho ảnh. Thì ta tăng độ chênh lệch giữa các giá trị pixel so với giá trị trung bình của chúng. Đối với mỗi điểm ảnh, ta thực hiện phép tính (pixel - mean) * factor + mean, sau đó dùng np.clip để giới hạn giá trị nằm trong khoảng 0-255.
- **flip_image**: Đối với chức năng lật ảnh theo chiều ngang, dọc. Ta sử dụng các phương thức transpose của PIL để lật ảnh. Nếu direction là 'horizontal', ta sử dụng Image.FLIP_LEFT_RIGHT, nếu là 'vertical', ta sử dụng Image.FLIP_TOP_BOTTOM.
- **convert_RGB_to_grayscale:** Đối với chức năng chuyển đổi màu RGB thành ảnh xám. Ta sẽ sử dụng công thức tính giá trị xám từ các giá trị màu RGB: gray_scale = 0.299 * red + 0.587 * green + 0.114 * blue. Đối với mỗi điểm ảnh, ta tính toán giá trị xám và gán lại giá trị này cho cả ba kênh màu (red, green, blue).
- **Convert_RGB_to_sepia:** Đối với chức năng chuyển đổi màu sepia thì ta sử dụng ma trận chuyển đổi màu sepia để tính toán lại giá trị của các kênh màu (red, green, blue) cho từng điểm ảnh. Các công thức tính toán giá trị mới: new_red = 0.393 * red + 0.769 * green + 0.189 * blue, new_green = 0.349 * red + 0.686 * green + 0.168 * blue, new_blue = 0.272 * red + 0.534 * green + 0.131 * blue.
- **blur_image:** Đối với chức năng làm mờ ảnh ta sẽ sử dụng một kernel mờ (blur kernel) để tính toán giá trị trung bình của các điểm ảnh lân cận. Đối với mỗi điểm ảnh, ta thực hiện phép tính trung bình cộng của 9 điểm xung quanh (sử dụng kernel 5x5).
- **sharpen_image:** Để làm sắc nét ảnh ta sử dụng một kernel làm sắc nét (sharpen kernel) để tăng độ nét của các điểm ảnh lân cận. Đối với mỗi điểm ảnh, ta thực hiện phép tính với kernel sắc nét 3x3.
- **crop_center:** Để cắt ảnh theo kích thước (cắt ở trung tâm) thì xác định các tọa độ của phần cần cắt và sử dụng phương thức crop của PIL để cắt phần đó từ trung tâm ảnh.
- **crop_to_ellipse:** Muốn cắt ảnh theo khung elip. Tạo một mask elip và sử dụng phương thức paste của PIL để dán phần ảnh cần giữ lại vào một ảnh mới với nền đen.
- **crop_to_double_ellipse:** Cắt ảnh theo khung hai hình elip nghiêng chồng lên nhau. Sử dụng ImageDraw để vẽ hai elip chồng lên nhau lên một mask, sau đó sử dụng mask này để giữ lại phần ảnh bên trong elip và làm đen phần bên ngoài.

- **resize_image:** Chức năng phóng to/thu nhỏ ảnh thì ta sẽ sử dụng phương thức resize của PIL với tham số Image.LANCZOS để thay đổi kích thước ảnh.

3. Mô tả các hàm

3.1. read_img(img_path):

- Chức năng: Đọc ảnh từ đường dẫn img_path.
- Đầu vào:
  - **img_path** (str): Đường dẫn của ảnh.
- Đầu ra:
  - **Image:** Ảnh được đọc từ đường dẫn.
- Mô tả chi tiết: Sử dụng hàm **Image.open** từ thư viện PIL để mở và đọc ảnh từ đường dẫn được cung cấp. Nếu ảnh được mở thành công, hàm sẽ trả về đối tượng ảnh thuộc lớp **Image**. Nếu file không tồn tại hoặc có lỗi khi mở, hàm sẽ trả về **None.**

3.2. show_img(img)

- Chức năng: Hiển thị ảnh.
- Đầu vào: img (Image): Ảnh cần hiển thị.
- Đầu ra: Không có (hiển thị ảnh trực tiếp).
- Mô tả chi tiết: Sử dụng hàm **imshow** và **show** từ thư viện Matplotlib để hiển thị ảnh. Hàm **imshow** nhận đầu vào là đối tượng ảnh **Image** hoặc mảng **Numpy** để tạo ra hình ảnh nhưng không hiển thị ngay lập tức, cần sử dụng phương thức **show** để hiển thị hình ảnh ra màn hình.

3.3. save_img(img, img_path)

- Chức năng: Lưu ảnh vào đường dẫn img_path.
- Đầu vào:
  - **img (Image):** Ảnh cần lưu.
  - **img_path (str):** Đường dẫn để lưu ảnh.
- Đầu ra: Không có (lưu ảnh vào đường dẫn).
- Mô tả chi tiết: Sử dụng hàm **save** từ thư viện PIL để lưu ảnh vào đường dẫn được cung cấp.

3.4. change_brightness(img, factor)

- Chức năng: Thay đổi độ sáng của ảnh.
- Đầu vào:
  - **img** (Image): Ảnh cần thay đổi độ sáng.
  - **factor** (float): Hệ số thay đổi độ sáng (lớn hơn 1 làm sáng, nhỏ hơn 1 làm tối).
- Đầu ra:

- **Image**: Ảnh sau khi thay đổi độ sáng.

- Mô tả chi tiết: Sử dụng **numpy** để chuyển đổi ảnh thành mảng số. Nhân các giá trị pixel với **factor** và sử dụng **np.clip** để đảm bảo giá trị pixel nằm trong khoảng 0-255. Cuối cùng, chuyển đổi mảng số trở lại thành đối tượng **Image** của PIL. Các câu lệnh **np.clip** giúp giới hạn giá trị các pixel để tránh bị overflow hoặc underflow.

3.5. change_contrast(img, factor)

- Chức năng: Thay đổi độ tương phản của ảnh.
- Đầu vào:
  - **img** (Image): Ảnh cần thay đổi độ tương phản.
  - **factor** (float): Hệ số thay đổi độ tương phản.
- Đầu ra:
  - **Image**: Ảnh sau khi thay đổi độ tương phản.
- Mô tả chi tiết:

+ Chuyển đổi hình ảnh thành mảng NumPy: Hình ảnh đầu vào được chuyển đổi thành một mảng NumPy để dễ dàng truy cập và thao tác với các giá trị pixel. Mảng này lưu trữ các giá trị RGB của từng pixel trong hình ảnh.

+ Tính giá trị trung bình của các pixel: Giá trị trung bình của các pixel được tính toán bằng cách lấy trung bình của mảng NumPy theo các trục chiều cao (axis 0) và chiều rộng (axis 1) của hình ảnh. **keepdims=True** giữ nguyên các chiều của mảng sau khi tính trung bình, giúp việc thao tác mảng dễ dàng hơn.

+ Thay đổi độ tương phản của các pixel:

- Độ tương phản của các pixel được thay đổi bằng công thức: **(arr - mean) * factor + mean**. Công thức này làm cho các giá trị pixel xa trung bình hơn (nếu **factor** > 1) hoặc gần trung bình hơn (nếu **factor** < 1).
- Các giá trị pixel sau khi tính toán có thể vượt ra ngoài khoảng [0, 255], do đó, hàm **np.clip** được sử dụng để giới hạn các giá trị trong khoảng này. Các giá trị nhỏ hơn 0 sẽ được đặt thành 0 và các giá trị lớn hơn 255 sẽ được đặt thành 255.
- Sau đó, mảng NumPy được chuyển đổi lại thành kiểu **uint8** để phù hợp với định dạng ảnh.

+ Chuyển đổi mảng NumPy trở lại thành đối tượng Image: Mảng NumPy sau khi thay đổi độ tương phản được chuyển đổi trở lại thành đối tượng **Image** bằng phương thức **Image.fromarray**.

+ Trả về hình ảnh mới đã thay đổi độ tương phản: Hình ảnh mới sau khi thay đổi độ tương phản được trả về.

3.6. flip_image(img, direction)

- Chức năng: Lật ảnh theo chiều ngang hoặc chiều dọc.
- Đầu vào:
  - **img** (Image): Ảnh cần lật.
  - **direction** (str): Hướng lật ('horizontal' hoặc 'vertical').
- Đầu ra:
  - Image: Ảnh sau khi lật.
- Mô tả chi tiết: Sử dụng phương thức transpose của PIL để lật ảnh theo chiều ngang (**Image.FLIP_LEFT_RIGHT)** hoặc chiều dọc **(Image.FLIP_TOP_BOTTOM**). Nếu hướng lật

không hợp lệ, hàm sẽ trả về ảnh gốc. Phép lật ảnh giúp thay đổi cách hiển thị của ảnh mà không thay đổi nội dung ảnh.

3.7. convert_RGB_to_grayscale(image)

- Chức năng: Chuyển đổi ảnh màu RGB thành ảnh xám.
- Đầu vào:
  - **image** (Image): Ảnh màu cần chuyển đổi.
- Đầu ra:
  - **Image**: Ảnh xám.
- Mô tả chi tiết:

+ Chuyển đổi hình ảnh thành mảng NumPy: Hình ảnh đầu vào được chuyển đổi thành một mảng NumPy để dễ dàng truy cập và thao tác với các giá trị pixel.

+ Tạo bản sao của hình ảnh gốc: Một bản sao của hình ảnh gốc được tạo ra để chứa các giá trị pixel mới sau khi chuyển đổi. Điều này đảm bảo rằng hình ảnh gốc không bị thay đổi trong quá trình xử lý.

+ Lấy kích thước của hình ảnh: Lấy chiều cao, chiều rộng và số lượng kênh màu của hình ảnh từ mảng NumPy. Thông tin này sẽ được sử dụng để duyệt qua từng pixel của hình ảnh.

+ Vòng lặp qua từng pixel trong hình ảnh: Hai vòng lặp **for** lồng nhau được sử dụng để duyệt qua từng pixel của hình ảnh. Vòng ngoài duyệt qua từng hàng (chiều cao của hình ảnh) và vòng trong duyệt qua từng cột (chiều rộng của hình ảnh).

+ Lấy giá trị RGB của từng pixel: Sử dụng phương thức **getpixel** của đối tượng **image** để lấy giá trị RGB tại từng tọa độ pixel. Tọa độ của pixel được xác định bởi các vòng lặp for.

+ Chuyển đổi giá trị RGB thành giá trị xám: Sử dụng công thức chuyển đổi giá trị RGB thành giá trị xám: **grey_scale = int(0.299 * red + 0.587 * green + 0.114 * blue).** Công thức này sử dụng các hệ số phản ánh độ nhạy của mắt người đối với các màu đỏ, xanh lá cây và xanh dương. Giá trị xám được tính bằng cách kết hợp các giá trị RGB theo các hệ số này.

+ Gán giá trị xám cho pixel trong hình ảnh mới: Sử dụng phương thức **putpixel** của đối tượng **new_image** để gán giá trị xám vừa tính được cho pixel tương ứng trong hình ảnh mới. Quá trình này được lặp lại cho tất cả các pixel trong hình ảnh.

+ Trả về hình ảnh mới đã chuyển đổi: Sau khi tất cả các pixel đã được chuyển đổi và gán giá trị xám, hình ảnh mới được trả về. Hình ảnh này bây giờ là một hình ảnh xám hoàn chỉnh.

3.8. convert_RGB_to_sepia(image)

- Chức năng: Chuyển đổi ảnh màu RGB thành ảnh sepia.
- Đầu vào:
  - **image** (Image): Ảnh màu cần chuyển đổi.
- Đầu ra:
  - **Image**: Ảnh sepia.
- Mô tả chi tiết:

+ Định nghĩa ma trận trọng số màu sepia: Ma trận **COLOR_WEIGHT_SEPIA** được định nghĩa để chuyển đổi giá trị RGB của từng pixel sang giá trị màu sepia. Ma trận này chứa các hệ số được sử dụng để tính toán giá trị mới cho mỗi kênh màu (đỏ, xanh lá cây và xanh dương).

+ Chuyển đổi hình ảnh thành mảng NumPy: Hình ảnh đầu vào được chuyển đổi thành một mảng NumPy để dễ dàng truy cập và thao tác với các giá trị pixel. Việc chuyển đổi này cũng đảm

bảo rằng các giá trị pixel có kiểu dữ liệu **float32** để thực hiện các phép toán đại số một cách chính xác.

+ Chuyển đổi ma trận trọng số màu sepia thành mảng NumPy: Ma trận COLOR_WEIGHT_SEPIA được chuyển đổi thành một mảng NumPy để sử dụng trong phép nhân ma trận với mảng pixel của hình ảnh.

+ Nhân ma trận để áp dụng hiệu ứng sepia:

- Sử dụng hàm **np.matmul**, mảng pixel của hình ảnh được nhân với ma trận trọng số màu sepia (đã chuyển vị **color_weight.T**). Phép nhân ma trận này áp dụng các hệ số màu sepia cho từng pixel, tạo ra một mảng mới **sepia_array** chứa các giá trị pixel đã được chuyển đổi.
- Hàm np.clip được sử dụng để giới hạn các giá trị pixel trong khoảng [0, 255] nhằm đảm bảo giá trị hợp lệ cho mỗi kênh màu.

+ Chuyển đổi mảng NumPy trở lại thành đối tượng **Image**: Mảng NumPy **sepia_array** sau khi thay đổi được chuyển đổi trở lại thành đối tượng Image bằng phương thức **Image.fromarray**. Phương thức này chuyển đổi mảng NumPy trở lại định dạng ảnh để có thể lưu trữ hoặc hiển thị.

+ Trả về hình ảnh mới đã chuyển đổi thành sepia: Hình ảnh mới sau khi chuyển đổi thành sepia được trả về. Hình ảnh này có hiệu ứng màu sepia cổ điển.

3.9. blur_image(img)

- Chức năng: Làm mờ ảnh.
- Đầu vào:
  - img (Image): Ảnh cần làm mờ.
- Đầu ra:
  - Image: Ảnh sau khi làm mờ.
- Mô tả chi tiết:

+ Chuyển đổi hình ảnh thành mảng NumPy: Hình ảnh đầu vào được chuyển đổi thành một mảng NumPy để dễ dàng truy cập và thao tác với các giá trị pixel. Mảng này lưu trữ các giá trị RGB của từng pixel trong hình ảnh.

+ Tạo kernel làm mờ: Một kernel (nhân ma trận) 5x5 được tạo ra với các giá trị bằng 1/25 (hoặc 0.04). Kernel này sẽ được sử dụng để tính toán giá trị trung bình của các pixel xung quanh một pixel cụ thể, từ đó làm mờ hình ảnh.

+ Khởi tạo mảng kết quả: Một mảng NumPy mới **result** được khởi tạo với cùng kích thước và kiểu dữ liệu như mảng hình ảnh gốc để lưu trữ kết quả của quá trình làm mờ.

+ Kiểm tra định dạng của hình ảnh: Kiểm tra định dạng của mảng hình ảnh để xác định xem nó là ảnh xám (grayscale) hay ảnh màu (RGB). Điều này được thực hiện bằng cách kiểm tra số lượng chiều của mảng **arr.**

+ Làm mờ ảnh xám: Nếu hình ảnh là ảnh xám (mảng 2 chiều), mảng được làm phẳng và áp dụng hàm **np.convolve** với kernel làm mờ. Kết quả sau đó được giới hạn trong khoảng [0, 255] bằng hàm **np.clip** và định dạng lại về kích thước gốc.

+ Làm mờ ảnh màu: Nếu hình ảnh là ảnh màu (mảng 3 chiều), quá trình làm mờ được thực hiện riêng biệt cho từng kênh màu (R, G, B). Vòng lặp **for** được sử dụng để duyệt qua từng kênh màu. Mỗi kênh màu được làm phẳng, áp dụng hàm **np.convolve** với kernel làm mờ, giới hạn trong khoảng [0, 255], và định dạng lại về kích thước gốc.

+ Chuyển đổi mảng NumPy trở lại thành đối tượng Image: Mảng NumPy kết quả **result** sau khi làm mờ được chuyển đổi trở lại thành đối tượng **Image** bằng phương thức **Image.fromarray**. Phương thức này chuyển đổi mảng NumPy trở lại định dạng ảnh để có thể lưu trữ hoặc hiển thị.

+ Trả về hình ảnh mới đã được làm mờ: Hình ảnh mới sau khi được làm mờ được trả về.

3.10. sharpen_image(img)

- Chức năng: Làm sắc nét ảnh.
- Đầu vào:
  - img (Image): Ảnh cần làm sắc nét.
- Đầu ra:
  - Image: Ảnh sau khi làm sắc nét.
- Mô tả chi tiết:

+ Chuyển đổi hình ảnh thành mảng NumPy: Hình ảnh đầu vào được chuyển đổi thành một mảng NumPy để dễ dàng truy cập và thao tác với các giá trị pixel. Mảng này lưu trữ các giá trị RGB của từng pixel trong hình ảnh.

+ Tạo kernel làm sắc nét: Một kernel (nhân ma trận) 3x3 được tạo ra với các giá trị **[[0, -1, 0], [-1, 5, -1], [0, -1, 0]]**. Kernel này sẽ được sử dụng để tăng cường các cạnh và chi tiết trong hình ảnh, làm cho hình ảnh trở nên sắc nét hơn.

+ Khởi tạo mảng kết quả: Một mảng NumPy mới **result** được khởi tạo với cùng kích thước và kiểu dữ liệu như mảng hình ảnh gốc để lưu trữ kết quả của quá trình làm sắc nét.

+ Kiểm tra định dạng của hình ảnh: Kiểm tra định dạng của mảng hình ảnh để xác định xem nó là ảnh xám (grayscale) hay ảnh màu (RGB). Điều này được thực hiện bằng cách kiểm tra số lượng chiều của mảng **arr**.

+ Làm sắc nét ảnh xám: Nếu hình ảnh là ảnh xám (mảng 2 chiều), mảng được làm phẳng và áp dụng hàm **np.convolve** với kernel làm sắc nét. Kết quả sau đó được giới hạn trong khoảng [0, 255] bằng hàm **np.clip** và định dạng lại về kích thước gốc.

+ Làm sắc nét ảnh màu: Nếu hình ảnh là ảnh màu (mảng 3 chiều), quá trình làm sắc nét được thực hiện riêng biệt cho từng kênh màu (R, G, B). Vòng lặp **for** được sử dụng để duyệt qua từng kênh màu. Mỗi kênh màu được làm phẳng, áp dụng hàm **np.convolve** với kernel làm sắc nét, giới hạn trong khoảng [0, 255], và định dạng lại về kích thước gốc.

+ Chuyển đổi mảng NumPy trở lại thành đối tượng Image: Mảng NumPy kết quả **result** sau khi làm sắc nét được chuyển đổi trở lại thành đối tượng **Image** bằng phương thức **Image.fromarray**. Phương thức này chuyển đổi mảng NumPy trở lại định dạng ảnh để có thể lưu trữ hoặc hiển thị.

+ Trả về hình ảnh mới đã được làm sắc nét: Hình ảnh mới sau khi được làm sắc nét được trả về

3.11. crop_center(img, size)

- Chức năng: Cắt ảnh từ trung tâm theo kích thước size.
- Đầu vào:
  - img (Image): Ảnh cần cắt.
  - size (tuple): Kích thước (width, height) của phần cần cắt.
- Đầu ra:

- Image: Ảnh sau khi cắt.

- Mô tả chi tiết:

+ Lấy kích thước của hình ảnh gốc: **width, height = img.size**: Lấy chiều rộng và chiều cao của hình ảnh gốc.

+ Xác định kích thước của hình ảnh mới: **new_width, new_height = size**: Lấy chiều rộng và chiều cao của phần hình ảnh mới sau khi cắt từ tham số size.

+ Tính toán tọa độ của vùng cắt:

- **left = (width - new_width) / 2**: Tọa độ x của mép trái của vùng cắt.
- **top = (height - new_height) / 2**: Tọa độ y của mép trên của vùng cắt.
- **right = (width + new_width) / 2**: Tọa độ x của mép phải của vùng cắt.
- **bottom = (height + new_height) / 2**: Tọa độ y của mép dưới của vùng cắt.
- Các công thức trên đảm bảo rằng phần hình ảnh mới sẽ được cắt từ trung tâm của hình ảnh gốc.

+ Cắt phần hình ảnh từ trung tâm: **img.crop((left, top, right, bottom))**: Sử dụng phương thức **crop** của đối tượng **Image** để cắt phần hình ảnh xác định bởi tọa độ **left, top, right, bottom**. Phần hình ảnh mới được cắt từ trung tâm của hình ảnh gốc với kích thước mới.

+ Trả về hình ảnh mới đã được cắt: Hình ảnh mới sau khi được cắt từ trung tâm được trả về. Hình ảnh này có kích thước mới xác định bởi tham số **size**.

3.12. crop_to_ellipse(img)

- Chức năng: Cắt ảnh theo khung elip.

- Đầu vào:

  - img (Image): Ảnh cần cắt.

- Đầu ra:

  - Image: Ảnh sau khi cắt theo khung elip.

- Mô tả chi tiết:

+ Lấy kích thước của hình ảnh gốc: **width, height = img.size**: Lấy chiều rộng và chiều cao của hình ảnh gốc.

+ Tạo mặt nạ (mask) hình ellipse:

- **mask = Image.new('L', (width, height), 0):** Tạo một ảnh mới với chế độ 'L' (ảnh xám, mỗi pixel là một giá trị từ 0 đến 255) có cùng kích thước với ảnh gốc, tất cả các pixel ban đầu có giá trị 0 (đen).
- **draw = ImageDraw.Draw(mask):** Tạo một đối tượng **ImageDraw** để vẽ lên mặt nạ.
- **draw.ellipse((0, 0, width, height), fill=255):** Vẽ một hình ellipse bên trong mặt nạ với tọa độ từ (0, 0) đến (width, height), và làm đầy ellipse với giá trị 255 (trắng). Điều này tạo ra một mặt nạ với hình ellipse trắng trên nền đen.

+ Tạo hình ảnh kết quả:

- **result = Image.new('RGB', (width, height)):** Tạo một ảnh mới với chế độ 'RGB' có cùng kích thước với ảnh gốc. Mặc định, ảnh này sẽ có nền đen (giá trị pixel (0, 0, 0)).
- **result.paste(img, (0, 0), mask):** Dán ảnh gốc lên ảnh kết quả tại vị trí (0, 0) và sử dụng mặt nạ để chỉ giữ lại các phần của ảnh gốc nằm bên trong ellipse. Các phần nằm ngoài ellipse sẽ giữ lại màu nền của ảnh kết quả (đen).

+ Trả về hình ảnh mới đã được cắt theo hình ellipse: Hình ảnh mới sau khi được cắt theo hình ellipse được trả về.

3.13. crop_to_double_ellipse(img)

- Chức năng: Cắt ảnh theo khung hai hình elip nghiêng chồng lên nhau.
- Đầu vào:
  - img (Image): Ảnh cần cắt.
- Đầu ra:
  - Image: Ảnh sau khi cắt theo khung hai hình elip.
- Mô tả chi tiết:

  + Lấy kích thước của hình ảnh gốc: **width, height = img.size**: Lấy chiều rộng và chiều cao của hình ảnh gốc để sử dụng trong việc xác định kích thước và vị trí của các hình ellipse.

  + Tạo mặt nạ (mask) hình ellipse:
  - **mask = Image.new('L', (width, height), 0):** Tạo một ảnh mới với chế độ 'L' (ảnh xám, mỗi pixel là một giá trị từ 0 đến 255) có cùng kích thước với ảnh gốc, ban đầu tất cả các pixel có giá trị 0 (đen).
  - **draw = ImageDraw.Draw(mask)**: Tạo một đối tượng ImageDraw để vẽ lên mặt nạ.

  + Vẽ hai hình ellipse trên mặt nạ:
  - **draw.ellipse([(0, height // 4), (width, 3 * height // 4)], fill=255)**: Vẽ một hình ellipse dọc trải dài từ 1/4 đến 3/4 chiều cao của hình ảnh, chiều rộng bằng chiều rộng hình ảnh. Hình ellipse này được làm đầy màu trắng (255).
  - **draw.ellipse([(width // 4, 0), (3 * width // 4, height)], fill=255)**: Vẽ một hình ellipse ngang trải dài từ 1/4 đến 3/4 chiều rộng của hình ảnh, chiều cao bằng chiều cao hình ảnh. Hình ellipse này cũng được làm đầy màu trắng (255).
  - Việc vẽ hai hình ellipse làm đầy màu trắng trên mặt nạ đen sẽ tạo ra một mặt nạ với các khu vực trắng tương ứng với các khu vực của hình ảnh sẽ được giữ lại khi cắt.

  + Tạo hình ảnh kết quả:
  - **result = Image.new('RGB', (width, height)):** Tạo một ảnh mới với chế độ 'RGB' có cùng kích thước với ảnh gốc. Mặc định, ảnh này sẽ có nền đen (giá trị pixel (0, 0, 0)).
  - **result.paste(img, (0, 0), mask):** Dán ảnh gốc lên ảnh kết quả tại vị trí (0, 0) và sử dụng mặt nạ để chỉ giữ lại các phần của ảnh gốc nằm bên trong hai ellipse. Các phần nằm ngoài hai ellipse sẽ giữ lại màu nền của ảnh kết quả (đen).

  + Trả về hình ảnh mới đã được cắt theo hai hình ellipse:
  - Hình ảnh mới sau khi được cắt theo hai hình ellipse được trả về.

3.14. resize_image(img, factor)

- Chức năng: Phóng to/thu nhỏ ảnh theo hệ số factor.
- Đầu vào:
  - img (Image): Ảnh cần thay đổi kích thước.
  - factor (float): Hệ số phóng to/thu nhỏ (lớn hơn 1 là phóng to, nhỏ hơn 1 là thu nhỏ).
- Đầu ra:
  - Image: Ảnh sau khi thay đổi kích thước.
- Mô tả chi tiết: Sử dụng phương thức resize của PIL với tham số Image.LANCZOS để thay đổi kích thước ảnh theo hệ số factor. Kết quả cuối cùng được chuyển đổi trở lại thành đối tượng Image của PIL.

3.15. Main

- Chức năng: Hàm main cung cấp giao diện dòng lệnh cho người dùng để thực hiện các chức năng xử lý ảnh khác nhau. Người dùng có thể chọn các chức năng như thay đổi độ sáng, thay đổi độ tương phản, lật ảnh, chuyển đổi ảnh RGB thành ảnh xám hoặc sepia, làm mờ hoặc làm sắc nét ảnh, cắt ảnh, cắt ảnh theo hình elip, và phóng to/thu nhỏ ảnh.
- Đầu vào:
    - Không có đầu vào từ hàm main trực tiếp. Tuy nhiên, trong quá trình thực hiện, hàm sẽ yêu cầu người dùng nhập các thông tin như đường dẫn tệp ảnh, hệ số thay đổi độ sáng, độ tương phản, hướng lật ảnh, chế độ chuyển đổi màu, hành động làm mờ/sắc nét, kích thước cắt ảnh, loại hình elip để cắt, và loại phóng to/thu nhỏ.
- Đầu ra:
    - Không có đầu ra trực tiếp từ hàm main. Tuy nhiên, các ảnh đã được xử lý sẽ được lưu vào tệp và hiển thị trên màn hình.
- Mô tả:
    - In bảng mô tả cách chọn chức năng trong màn hình console.
    - Yêu cầu nhập từ 0-8 cho từng chức năng tương ứng. Option 9 dùng để thoát chương trình.
    - Kiểm tra xem người dùng muốn thực hiện chức năng nào và xuất, lưu ảnh sau khi thực hiện chức năng.

4. Tài liệu tham khảo

[1] Github [Trực tuyến]. Available: https://github.com/nqq203/image-processing/tree/main/Document [Đã truy cập 24 7 2024].

[2] J. A. C. a. c. F. Lundh, "Pillow Documentation," Pillow, 2010-2014. [Trực tuyến]. Available: https://pillow.readthedocs.io/en/stable/. [Đã truy cập 15 6 2024].

[3] D. D. E. F. M. D. a. t. M. d. t. John Hunter, "Using Matplotlib," The Matplotlib development team, 2012–2024. [Trực tuyến]. Available: https://matplotlib.org/stable/users/index. [Đã truy cập 24 7 2024].

[4] N. Developers, "NumPy reference," 16 9 2023. [Online]. Available: https://numpy.org/doc/stable/reference/index.html. [Accessed 24 7 2024].