ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN

IMAGE PROCESSING

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

Giảng viên hướng dẫn: **Phan Thị Phương Uyên**

Sinh viên thực hiện: **Đoàn Ngọc Mai**

Mã số sinh viên: **21127104**

Mục lục

1. Các chức năng đã hoàn thành

STT	Tên chức năng	Mức độ hoàn thành
1	Thay đổi độ sáng cho ảnh	100%
2	Thay đổi độ tương phản	100%
3	Lật ảnh ngang – dọc	100%
4	Chuyển đổi ảnh RGB thành ảnh xám	100%
5	Chuyển đổi ảnh RGB thành ảnh sepia	100%
6	Làm mờ ảnh	100%
7	Làm sắc nét ảnh	95%
	Cắt ảnh theo kích thước (cắt ở trung tâm)	100%
8	Cắt ảnh theo khung hình tròn	100%
9	Cắt ảnh theo khung là 2 hình elip chéo nhau	90%

2. Ý tưởng thực hiện và mô tả các hàm chức năng

a. Thư viện được khai báo

- numpy: là thư viện quan trọng dung để chuyển đổi hình ảnh về ma trận và thực thi các phép tính toán ma trận để hoàn thành các hàm chức năng
- PIL.Image: được dùng để đọc, lưu và ghi ảnh.
- Matplotlib.pylot: được dùng để hiển thị hình ảnh

b. Các hàm chức năng

i. Hàm adjust_brightness

Công dụng: Dùng để thay đổi độ sáng của bức ảnh

Ý tưởng: Thêm giá trị độ sáng (brightness) vào từng kênh màu (R, G, B) của hình ảnh, sau đó giới hạn các giá trị pixel trong khoảng từ 0 đến 255 để đảm bảo rằng hình ảnh vẫn có định dạng hợp lệ.

Tham số đầu vào:

- img: Hình ảnh gốc cần điều chỉnh độ sáng. Có thể là hình ảnh PIL hoặc mảng numpy biểu diễn hình ảnh.
- brightness: Giá trị điều chỉnh độ sáng của hình ảnh. Có thể là một số dương hoặc âm. Giá trị dương tăng độ sáng và giá trị âm giảm độ sáng.

Kết quả trả về: Hàm trả về một mảng numpy biểu diễn hình ảnh đã được điều chỉnh độ sáng.

Mô tả:

- Đảm bảo rằng giá trị brightness là một số thực bằng cách ép kiểu về float, để cho phép điều chỉnh độ sáng của hình ảnh với mức độ chi tiết tùy ý. Điều này cho phép chúng ta điều chỉnh độ sáng của hình ảnh theo những giá trị không chỉ là số nguyên, mà có thể là các số thập phân hoặc số âm.
- Chuyển đổi hình ảnh đầu vào thành mảng numpy để thực hiện các phép toán trên các pixel.
- Thêm giá trị brightness vào từng kênh màu (R, G, B) của hình ảnh. Điều này có tác động tương đương với tăng hoặc giảm độ sáng của hình ảnh.
- Giới hạn các giá trị pixel trong khoảng từ 0 đến 255 bằng cách sử dụng hàm np.clip(). Điều này đảm bảo rằng các giá trị pixel sau khi điều chỉnh nằm trong phạm vi hợp lệ cho định dạng hình ảnh 8-bit (uint8).
- Ép kiểu dữ liệu của mảng thành uint8, nếu hình ảnh đang sử dụng kiểu dữ liệu khác uint8, để đảm bảo rằng nó nằm trong phạm vi từ 0 đến 255.
- Trả về hình ảnh đã điều chỉnh độ sáng dưới dạng mảng numpy.

Hình ảnh kết quả:

Ảnh gốc (500x500)

Ảnh tăng thêm 50 độ sáng

ii. adjust_contrast

Công dụng: Dùng để điều chỉnh độ tương phản của bức ảnh.

Ý tưởng: Để điều chỉnh độ tương phản của hình ảnh, chúng ta sẽ áp dụng công thức **new_value = (value - min_value) * factor + min_value** (1, 2) cho từng kênh màu (R, G, B) của hình ảnh. Công thức này sẽ thực hiện việc điều chỉnh độ tương

phản bằng cách dịch chuyển giá trị pixel gốc của mỗi kênh về phạm vi giá trị mới có độ tương phản tăng lên hoặc giảm xuống.

Tham số đầu vào:

- image: Hình ảnh gốc cần điều chỉnh độ tương phản.
- factor: Giá trị điều chỉnh độ tương phản. Có thể là một số dương hoặc âm. Giá trị dương tăng độ tương phản và giá trị âm giảm độ tương phản.

Kết quả trả về: Hàm trả về một đối tượng hình ảnh đã được điều chỉnh độ tương phản.

Mô tả:

- Chuyển đổi hình ảnh đầu vào thành một mảng numpy để thực hiện các phép toán trên các pixel.
- Đối với từng kênh màu (R, G, B), tính giá trị pixel nhỏ nhất (min_value) trong kênh đó bằng cách sử dụng hàm np.min().
- Áp dụng công thức **new_value = (value - min_value) * factor + min_value** để điều chỉnh độ tương phản của từng kênh màu. Công thức này sẽ dịch chuyển giá trị pixel gốc của mỗi kênh về phạm vi giá trị mới có độ tương phản tăng lên hoặc giảm xuống dựa trên giá trị factor. Khi factor bằng 0, độ tương phản của hình ảnh sẽ không thay đổi.
- Giới hạn các giá trị pixel đã điều chỉnh trong khoảng từ 0 đến 255 bằng cách sử dụng hàm np.clip(). Điều này đảm bảo rằng các giá trị pixel sau khi điều chỉnh nằm trong phạm vi hợp lệ cho định dạng hình ảnh 8-bit (uint8).
- Chuyển đổi lại mảng numpy đã điều chỉnh độ tương phản thành đối tượng hình ảnh PIL bằng cách sử dụng Image.fromarray() và ép kiểu dữ liệu thành uint8.
- Trả về hình ảnh đã điều chỉnh độ tương phản dưới dạng đối tượng hình ảnh PIL.

Hình ảnh kết quả:

Ảnh gốc (500x500)

Ảnh với factor là 2.0

iii. flip_horizontal

Công dụng: Hàm flip_horizontal được sử dụng để lật ngang (lật theo trục dọc) một hình ảnh.

Ý tưởng: Hàm này sẽ nhận một hình ảnh đầu vào và chuyển đổi nó thành một mảng numy. Sau đó, nó sử dụng hàm np.flipud() để lật ngang (lật theo trục dọc) mảng ảnh.

Tham số đầu vào: image là hình ảnh gốc cần lật ngang.

Kết quả trả về: Hàm trả về một mảng numpy biểu diễn hình ảnh sau khi đã lật ngang.

Mô tả:

- Chuyển đổi hình ảnh đầu vào thành một mảng numpy để thực hiện các phép toán trên các pixel.
- Sử dụng hàm np.flipud() (3) để lật ngang (lật theo trục dọc) mảng ảnh. Hàm này sẽ hoán đổi các hàng của mảng theo trục dọc, dẫn đến hiệu ứng lật ngang của hình ảnh.
- Trả về mảng numpy đã lật ngang làm kết quả của hàm.

Hình ảnh kết quả:

Ảnh gốc (500x500)

Ảnh lật dọc

iv. flip_vertical

Công dụng: Hàm được sử dụng để lật dọc (lật theo trục ngang) một hình ảnh.

Ý tưởng: Hàm này sẽ nhận một hình ảnh đầu vào và chuyển đổi nó thành một mảng numpy. Sau đó, nó sử dụng hàm np.fliplr() (4) để lật dọc (lật theo trục ngang) mảng ảnh.

Tham số đầu vào: image: Hình ảnh gốc cần lật dọc.

Kết quả trả về: Hàm trả về một mảng numpy biểu diễn hình ảnh sau khi đã lật dọc.

Mô tả:

- Chuyển đổi hình ảnh đầu vào thành một mảng numpy để thực hiện các phép toán trên các pixel.
- Sử dụng hàm np.fliplr()để lật dọc (lật theo trục ngang) mảng ảnh. Hàm này sẽ hoán đổi các cột của mảng theo trục ngang, dẫn đến hiệu ứng lật dọc của hình ảnh.
- Trả về mảng numpy đã lật dọc làm kết quả của hàm.

Hình ảnh kết quả:

Ảnh gốc (500x500)

Ảnh lật dọc

v. rgb_to_gray

Công dụng: Hàm được sử dụng để chuyển đổi một hình ảnh RGB thành hình ảnh grayscale.

Ý tưởng: Hàm này sẽ nhận một hình ảnh RGB đầu vào dưới dạng mảng numpy, sau đó tính toán giá trị pixel của hình ảnh grayscale bằng cách lấy trung bình của các kênh màu R, G và B theo tỷ lệ đã cho **(0.2989 * R + 0.5870 * G + 0.1140 * B)** (5).

Tham số đầu vào:

- rgb_image: Hình ảnh RGB gốc cần chuyển đổi thành grayscale. Đây là một mảng numpy có kích thước (height, width, 3) trong đó 3 là số kênh màu (R, G, B).

Kết quả trả về: Hàm trả về một mảng numpy biểu diễn hình ảnh grayscale sau khi đã chuyển đổi.

Mô tả:

- Hàm lấy các kênh màu R, G và B của hình ảnh RGB bằng cách cắt mảng numpy rgb_image theo các chiều thứ ba. Mỗi kênh sẽ là một mảng 2D (height, width) trong mảng 3D.
- Tiến hành tính giá trị pixel của hình ảnh grayscale bằng cách sử dụng công thức sau: **gray_image = 0.2989 * r + 0.5870 * g + 0.1140 * b**, trong đó r, g, b lần lượt là các mảng 2D biểu diễn kênh màu R, G và B.

- Điều chỉnh giá trị của gray_image thành kiểu dữ liệu uint8 bằng cách sử dụng astype(np.uint8) để đảm bảo rằng mảng pixel grayscale nằm trong phạm vi từ 0 đến 255, có định dạng hợp lệ cho hình ảnh 8-bit.

Ưu điểm của hàm:

- Đơn giản và dễ hiểu: Hàm này sử dụng các phép tính cơ bản để chuyển đổi hình ảnh RGB thành hình ảnh grayscale, dễ hiểu và dễ cài đặt.
- Không cần thư viện bên ngoài: Hàm này không sử dụng bất kỳ thư viện bên ngoài nào, như OpenCV, ... v.v., nên không yêu cầu cài đặt thư viện phụ thuộc.

Nhược điểm của hàm:

- Thiếu sự linh hoạt: Hàm này sử dụng công thức chuyển đổi cố định **(gray_image = 0.2989 * r + 0.5870 * g + 0.1140 * b**) với các hệ số cố định cho việc tính trung bình của kênh màu. Điều này làm giảm sự linh hoạt trong việc điều chỉnh cách chuyển đổi tùy thuộc vào mục tiêu và yêu cầu cụ thể của ứng dụng.
- Hiệu năng khi chuyển đổi số lượng lớn hình ảnh: Khi chuyển đổi một lượng lớn hình ảnh, việc tính toán các phép nhân và cộng trên mảng có thể trở nên chậm hơn so với việc sử dụng các thư viện tối ưu hơn như OpenCV.

Hình ảnh kết quả:

Ảnh gốc (500x500)

Ảnh xám

vi. rgb_to_sepia

Công dụng: Hàm rgb_to_sepia được sử dụng để chuyển đổi một hình ảnh RGB thành hình ảnh Sepia.

Ý tưởng: Hàm này sẽ nhận một hình ảnh RGB đầu vào dưới dạng mảng numpy. Sau đó, nó sử dụng các công thức biến đổi Sepia để tính toán giá trị mới cho các kênh màu R, G, B, tạo thành hình ảnh Sepia (6).

Tham số đầu vào:

- rgb_image: Hình ảnh RGB gốc cần chuyển đổi thành hình ảnh Sepia. Đây là một mảng numpy có kích thước (height, width, 3) trong đó 3 là số kênh màu (R, G, B).

Kết quả trả về: Hàm trả về một mảng numpy biểu diễn hình ảnh Sepia sau khi đã chuyển đổi.

Mô tả:

- Hàm lấy các kênh màu R, G và B của hình ảnh RGB bằng cách cắt mảng numpy rgb_image theo các chiều thứ ba. Mỗi kênh sẽ là một mảng 2D (height, width) trong mảng 3D.
- Tiến hành tính toán giá trị mới cho các kênh màu R, G, B theo các công thức Sepia (6).
    - **sepia_r = (0.393 * r) + (0.769 * g) + (0.189 * b)**
    - **sepia_g = (0.349 * r) + (0.686 * g) + (0.168 * b)**
    - **sepia_b = (0.272 * r) + (0.534 * g) + (0.131 * b)**
- Giới hạn các giá trị pixel của các kênh màu Sepia trong khoảng từ 0 đến 255 bằng cách sử dụng hàm np.clip để đảm bảo rằng chúng nằm trong phạm vi hợp lệ cho định dạng hình ảnh 8-bit (uint8).
- Tạo mảng ảnh Sepia từ các kênh màu R, G, B đã tính toán bằng cách sử dụng np.stack để ghép các mảng thành một mảng 3D mới.
- Trả về mảng numpy biểu diễn hình ảnh Sepia làm kết quả của hàm.

Ưu điểm:

- Đơn giản và dễ hiểu: Hàm này sử dụng các phép tính cơ bản và công thức Sepia rõ ràng, dễ hiểu và dễ cài đặt.
- Hiệu suất tốt: Cách tính toán sử dụng các phép tính số học đơn giản, không yêu cầu nhiều tài nguyên tính toán, do đó đảm bảo hiệu suất tốt.

Nhược điểm:

- Thiếu sự linh hoạt: Hàm này áp dụng các công thức Sepia cố định, không cho phép người dùng tùy chỉnh hoặc điều chỉnh các thông số Sepia như trong một số phương pháp chuyển đổi Sepia khác.
- Không cân nhắc đến gamma correction: Trong một số phương pháp Sepia, việc áp dụng gamma correction có thể giúp tăng tính chân thực và tự nhiên của hình ảnh Sepia. Tuy nhiên, trong hàm này không có bước gamma correction nào.
- Không khả quy: Phương pháp chuyển đổi này không có khả năng đi ngược lại (không khả quy), tức là không thể chuyển đổi hình ảnh Sepia trở lại hình ảnh RGB một cách chính xác.

Hình ảnh kết quả:

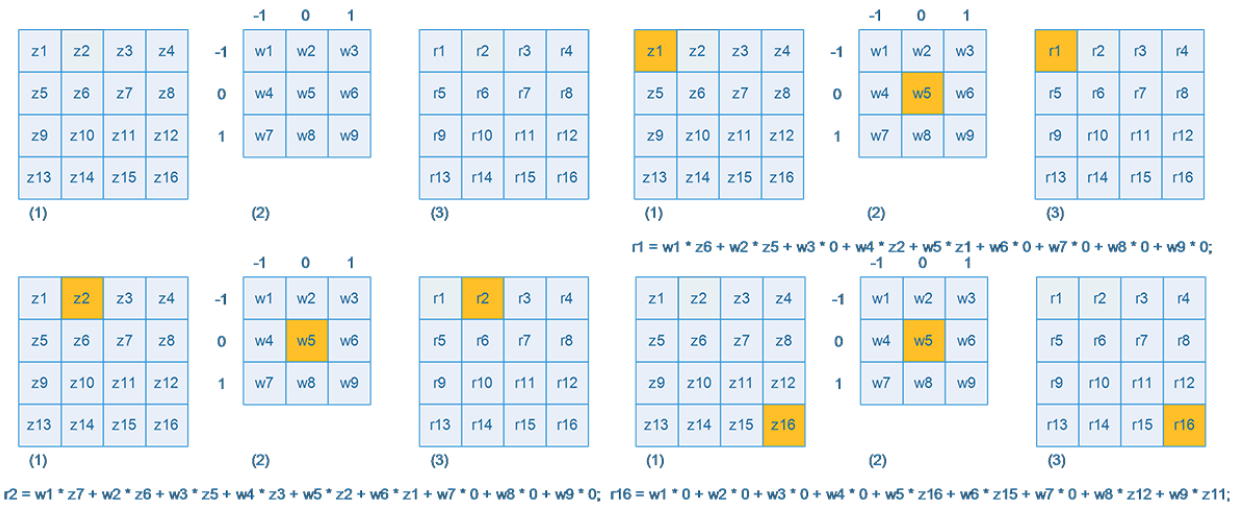
Ảnh gốc (500x500)

Ảnh Sephia

vii. apply_filter

Công dụng: Hàm apply_filter được sử dụng để áp dụng một bộ lọc (kernel) lên một hình ảnh. Là hàm tiền đề để làm mờ ảnh và làm sắc nét ảnh (thực thi trên ảnh xám).

Ý tưởng: Hàm này sẽ nhận một hình ảnh đầu vào dưới dạng mảng numpy và một bộ lọc (kernel) cũng dưới dạng mảng numpy. Sau đó, hàm sử dụng phép tích chập ([7], [8], [9], [10], [11], [12]) giữa hình ảnh và kernel để tạo ra một ma trận kết quả biểu diễn hình ảnh đã được lọc.



Hình ảnh minh họa phép tích chập

Ký hiệu: (1) ảnh nguồn, (2) kernel, (3) ảnh kết quả.

- Nguyên lý áp dụng phép tính chập sẽ là:
  - Duyệt qua từng cột, hàng trong ma trận ảnh đầu vào.
  - Lấy phần của ảnh mà kernal đè lên, tức là lấy phần ảnh có kích thước bằng kernal tại vị trí (x,y) trong ảnh.
  - Kiểm tra xem kích thước ảnh và kernal có bằng nhau không. Nếu bằng nhau, ta sẽ tiến hành thực hiện phép tích chập.
  - Phép tích chập được thực hiện bằng cách nhên từng phần tử của ảnh và kernal lại với nhau, sau đó tính tổng kết quả.
  - Sau cùng gán giá trị lọc vào ma trận cuối cùng.

Tham số đầu vào:

- image: Hình ảnh gốc cần áp dụng bộ lọc lên. Đây là một mảng numpy 2D (height, width) biểu diễn hình ảnh grayscale.
- kernel: Bộ lọc (kernel) cần áp dụng lên hình ảnh. Đây là một mảng numpy 2D (kernel_height, kernel_width).

Kết quả trả về: Hàm trả về một mảng numpy biểu diễn hình ảnh sau khi đã áp dụng bộ lọc.

Mô tả:

- Lấy kích thước của hình ảnh và bộ lọc (kernel) thông qua thuộc tính shape của mảng numpy.

- Tạo một ma trận kết quả có cùng kích thước với hình ảnh, ban đầu được lấp đầy các giá trị 0 (màu đen).
- Áp dụng bộ lọc thông qua phép tích chập: Duyệt qua từng điểm ảnh trong hình ảnh, lấy phần của ảnh mà kernel "đè" lên (image_patch), sau đó tính giá trị lọc bằng cách nhân từng phần tử của image_patch với kernel và tính tổng các phần tử thu được.
- Gán giá trị lọc vào ma trận kết quả tại vị trí tương ứng.
- Trả về ma trận kết quả biểu diễn hình ảnh sau khi đã áp dụng bộ lọc.

Ưu điểm:

- Đơn giản và dễ hiểu: Hàm này sử dụng các phép tính cơ bản để thực hiện phép tích chập, dễ hiểu và dễ cài đặt.
- Khả năng tự định nghĩa bộ lọc: Người dùng có thể tự định nghĩa và sử dụng các bộ lọc (kernel) khác nhau tùy thuộc vào mục tiêu và yêu cầu cụ thể của ứng dụng.

Nhược điểm:

- Hiệu năng khi chuyển đổi số lượng lớn hình ảnh: Hàm này sử dụng vòng lặp kép để duyệt qua từng điểm ảnh và áp dụng bộ lọc, do đó, hiệu năng có thể không tốt khi xử lý số lượng lớn hình ảnh..

viii. blur_image

Công dụng: Hàm blur_image được sử dụng để làm mờ một hình ảnh bằng cách áp dụng bộ lọc.

Ý tưởng:

- Hàm này sẽ nhận một hình ảnh đầu vào dưới dạng mảng numpy và một kích thước bộ lọc trung bình (kernel_size) (13, 14). Sau đó, hàm tạo một bộ lọc trung bình có kích thước kernel_size x kernel_size với các giá trị được chuẩn hóa sao cho tổng các phần tử trong bộ lọc bằng 1 (để đảm bảo tính chuẩn xác của bộ lọc trung bình). Cuối cùng, hàm áp dụng bộ lọc trung bình này lên hình ảnh để làm mờ hình ảnh.
- Ngoài ra, có cách làm mờ ảnh khác bằng cách Gauss, tuy nhiên cách thực thi hàm của em khi dung Gauss còn nhiều thiếu sót nên không đưa nó vào bài làm, thay vào đó em chọn làm bộ lọc trung bình.

- Bộ lọc trung bình:

- Ưu điểm: Đơn giản và nhanh chóng để tính toán. Dễ dàng triển khai và hiểu.
- Nhược điểm: Gây hiện tượng "blurring" mạnh, làm mất nhiều chi tiết hơn. Không thích hợp cho việc làm mờ ảnh trong các bài toán yêu cầu bảo tồn cạnh và chi tiết hình ảnh.

Tham số đầu vào:

- image: Hình ảnh gốc cần làm mờ. Đây là một mảng numpy 2D (height, width) biểu diễn hình ảnh grayscale.
- kernel_size: Kích thước bộ lọc trung bình. Đây là một số nguyên dương, ví dụ: 3, 5, 7, ...

Kết quả trả về: Hàm trả về một mảng numpy biểu diễn hình ảnh đã được làm mờ sau khi áp dụng bộ lọc trung bình.

Mô tả:

- Tạo một bộ lọc trung bình có kích thước kernel_size x kernel_size bằng cách tạo một ma trận kernel chứa các giá trị 1 và chuẩn hóa tổng các phần tử trong kernel bằng kernel_size x kernel_size. Việc chuẩn hóa này đảm bảo tính chuẩn xác của bộ lọc trung bình.
- Áp dụng bộ lọc trung bình đã tạo lên hình ảnh image bằng cách gọi hàm apply_filter với image và kernel như là các tham số đầu vào.
- Hàm apply_filter thực hiện phép tích chập giữa hình ảnh và bộ lọc trung bình, sau đó trả về hình ảnh đã được làm mờ.
- Trả về hình ảnh đã được làm mờ làm kết quả của hàm.

Ưu điểm:

- Hiệu năng tốt: Bộ lọc trung bình là một bộ lọc đơn giản và nhanh chóng, do đó hàm này có hiệu năng tốt khi áp dụng lên hình ảnh.

Nhược điểm:

- Không xử lý biên giới: Hàm này không xử lý biên giới của hình ảnh khi áp dụng bộ lọc trung bình, có thể dẫn đến hiện tượng mờ hoặc viền đen ở lề hình ảnh sau khi làm mờ.
- Hiệu năng khi chuyển đổi số lượng lớn hình ảnh: Hàm này sử dụng vòng lặp kép để duyệt qua từng điểm ảnh và áp dụng bộ lọc, do đó, hiệu năng có thể không tốt khi xử lý số lượng lớn hình ảnh.

Hình ảnh kết quả:

Ảnh xám

Ảnh xám được làm mờ

ix. sharpen_image

Công dụng: Hàm được sử dụng để làm sắc nét một hình ảnh bằng cách áp dụng bộ lọc sắc nét (15, 16).

Ý tưởng: Hàm này sẽ nhận một hình ảnh đầu vào dưới dạng mảng numpy và một kích thước bộ lọc sắc nét (kernel_size). Sau đó, hàm tạo một bộ lọc sắc nét có kích thước kernel_size x kernel_size, với giá trị 2 tại vị trí trung tâm và các giá trị còn lại bằng -1/(kernel_size*kernel_size). Cuối cùng, hàm áp dụng bộ lọc sắc nét này lên hình ảnh để làm sắc nét hình ảnh.

Tham số đầu vào:

- image: Hình ảnh gốc cần làm sắc nét. Đây là một mảng numpy 2D (height, width) biểu diễn hình ảnh grayscale.
- kernel_size: Kích thước bộ lọc sắc nét. Đây là một số nguyên dương, ví dụ: 3, 5, 7, ...

Kết quả trả về:

Hàm trả về một mảng numpy biểu diễn hình ảnh đã được làm sắc nét sau khi áp dụng bộ lọc sắc nét.

Mô tả:

- Tạo một bộ lọc sắc nét có kích thước kernel_size x kernel_size bằng cách tạo một ma trận kernel có giá trị 2 tại vị trí trung tâm và giá trị -1/(kernel_size*kernel_size) ở các vị trí còn lại. Điều này sẽ tạo ra một bộ lọc với tổng các phần tử bằng 0, là một bộ lọc có hiệu ứng làm sắc nét.
- Công thức tạo bộ lọc sắc nét có kích thước kernel_size x kernel_size như sau:
    - Tại vị trí trung tâm của bộ lọc, gán giá trị 2.0. Điều này đảm bảo rằng pixel tại vị trí trung tâm của bộ lọc sẽ có trọng số cao hơn khi áp dụng bộ lọc.
    - Các giá trị còn lại của bộ lọc đều gán giá trị -1/(kernel_size*kernel_size), đảm bảo tổng các giá trị trong bộ lọc bằng 0, nhằm duy trì độ sáng tổng thể của hình ảnh sau khi áp dụng bộ lọc. Nếu tổng các giá trị trong bộ lọc khác 0, hình ảnh sau khi áp dụng bộ lọc có thể thay đổi độ sáng tổng thể.
- Ý tưởng sau khi áp dụng bộ lọc này lên một điểm ảnh trong hình ảnh là như sau:
    - Giá trị mới của điểm ảnh tại vị trí (i, j) sẽ bằng tổng các giá trị của điểm ảnh trong vùng bộ lọc tại vị trí tương ứng nhân với giá trị tương ứng trong bộ lọc.
    - Tại vị trí trung tâm, giá trị của bộ lọc là 2.0, nên điểm ảnh tại vị trí trung tâm sẽ được nhân với 2.0, làm tăng độ sáng.
    - Tại các vị trí xung quanh vị trí trung tâm, giá trị của bộ lọc là -1/(kernel_size*kernel_size), nên điểm ảnh tại các vị trí này sẽ bị trừ đi một phần nhỏ từ các điểm ảnh lân cận, làm giảm độ sáng.
    - Kết quả là các đặc trưng sắc nét trong hình ảnh được làm nổi bật, tăng độ tương phản và độ sắc nét của hình ảnh.
- Áp dụng bộ lọc sắc nét đã tạo lên hình ảnh image bằng cách gọi hàm apply_filter với image và kernel như là các tham số đầu vào.
- Hàm apply_filter thực hiện phép tích chập giữa hình ảnh và bộ lọc sắc nét, sau đó trả về hình ảnh đã được làm sắc nét.
- Trả về hình ảnh đã được làm sắc nét làm kết quả của hàm.

Phần chưa thực hiện được:
Đối với những ảnh có nhiều chi tiết, khi thực thi hàm này sẽ có những lằn sọc do filter để lại.
Phương án cải thiện:

Dùng hàm gaussian_kernel để tính kernal rồi sau đó mới dung apply_filter để thực thi hàm làm sắc nét, làm như vậy số kernal được chọn sẽ tối ưu hơn là người dung tự chọn.

Ưu điểm:

- Hiệu năng tốt: Bộ lọc sắc nét là một bộ lọc đơn giản và nhanh chóng, do đó hàm này có hiệu năng tốt khi áp dụng lên hình ảnh.

Nhược điểm:

- Hiệu năng khi chuyển đổi số lượng lớn hình ảnh: Hàm này sử dụng vòng lặp kép để duyệt qua từng điểm ảnh và áp dụng bộ lọc, do đó, hiệu năng có thể không tốt khi xử lý số lượng lớn hình ảnh.Hình ảnh kết quả:

Hình ảnh kết quả:

Ảnh xám được làm mờ

Ảnh xám được làm sắc nét

Ảnh xám được làm mờ

Ảnh xám được làm sắc nét

x. crop_center_image

Công dụng: Hàm crop_center_image được sử dụng để cắt một phần ảnh chính giữa (trung tâm) của hình ảnh ban đầu.

Ý tưởng: Hàm này sẽ nhận một hình ảnh đầu vào và hai tham số là target_width và target_height, đại diện cho chiều rộng và chiều cao của khu vực cần cắt chính giữa từ hình ảnh ban đầu. Sau đó, hàm sẽ thực hiện việc tính toán các chỉ số cắt để xác định khu vực cần cắt ở giữa hình ảnh ban đầu và thực hiện cắt.

Tham số đầu vào:

- image: Hình ảnh ban đầu cần cắt. Đây là một đối tượng PIL.Image.
- target_width: Chiều rộng của khu vực cần cắt chính giữa từ hình ảnh ban đầu.
- target_height: Chiều cao của khu vực cần cắt chính giữa từ hình ảnh ban đầu.

Kết quả trả về:

Hàm trả về một đối tượng PIL.Image biểu diễn ảnh sau khi đã được cắt chính giữa.

Mô tả:

- Chuyển đổi hình ảnh đầu vào thành mảng numpy bằng cách sử dụng hàm np.array.
- Lấy chiều cao và chiều rộng của hình ảnh ban đầu từ mảng numpy đã chuyển đổi.
- Tính toán các chỉ số cắt để xác định khu vực cần cắt ở giữa hình ảnh ban đầu. Các chỉ số cắt được tính dựa trên target_width, target_height và kích thước ban đầu của hình ảnh.
    - start_x: Chỉ số cắt bắt đầu (x-coordinate) trong chiều ngang. Để lấy phần ảnh chính giữa, thì cần dịch chuyển phần cắt bắt đầu đến giữa chiều ngang của hình ảnh gốc. Để làm điều này, ta cần lấy giá trị bằng cách trừ chiều rộng của hình ảnh gốc (original_width) với chiều rộng của phần cần cắt (target_width) và sau đó chia cho 2 để đảm bảo rằng phần cắt nằm chính giữa. Điều này làm cho phần cắt bắt đầu từ vị trí start_x đến vị trí end_x.
    - start_y: Chỉ số cắt bắt đầu (y-coordinate) trong chiều dọc. Tương tự như start_x, chúng ta lấy giá trị bằng cách trừ chiều cao của hình ảnh

gốc (original_height) với chiều cao của phần cần cắt (target_height) và sau đó chia cho 2 để đảm bảo rằng phần cắt nằm chính giữa. Điều này làm cho phần cắt bắt đầu từ vị trí start_y đến vị trí end_y.
    - end_x: Chỉ số cắt kết thúc (x-coordinate) trong chiều ngang. Để xác định phần cắt kết thúc, chúng ta đơn giản lấy giá trị start_x cộng với target_width.
    - end_y: Chỉ số cắt kết thúc (y-coordinate) trong chiều dọc. Tương tự như end_x, chúng ta đơn giản lấy giá trị start_y cộng với target_height.
- Thực hiện cắt ảnh dựa trên các chỉ số đã tính toán, bằng cách trích xuất khu vực cần cắt từ mảng numpy.
- Chuyển đổi kết quả cắt về đối tượng PIL.Image bằng cách sử dụng hàm Image.fromarray.
- Trả về ảnh sau khi đã được cắt chính giữa làm kết quả của hàm.

Ưu điểm:

- Đơn giản và dễ hiểu: Hàm này sử dụng các phép tính cơ bản để thực hiện cắt hình ảnh chính giữa, dễ hiểu và dễ cài đặt.
- Tính tổng quát: Hàm này có thể được sử dụng để cắt chính giữa của hình ảnh với bất kỳ kích thước nào, giúp tái sử dụng cho nhiều tình huống.

Nhược điểm:

- Chưa xử lý các trường hợp đặc biệt: Hàm này không xử lý các trường hợp đặc biệt như trường hợp kích thước cắt lớn hơn kích thước hình ảnh ban đầu hoặc kích thước hình ảnh ban đầu nhỏ hơn kích thước cần cắt, có thể dẫn đến lỗi hoặc kết quả không như mong đợi.
- Không đảm bảo độ tương tự về tỷ lệ: Khi cắt chính giữa của hình ảnh, kích thước cắt có thể khác với tỷ lệ của kích thước ban đầu, do đó không đảm bảo độ tương tự về tỷ lệ giữa kích thước cắt và hình ảnh gốc.

Hình ảnh kết quả:

Ảnh gốc (500x500)

Ảnh đã cắt (200x200)

xi. crop_circular_image

Công dụng: Hàm crop_circular_image được sử dụng để cắt một hình ảnh thành hình tròn chính giữa.

Ý tưởng: Hàm này sẽ nhận một hình ảnh đầu vào dưới dạng một đối tượng PIL.Image. Sau đó, nó sẽ tính toán tâm của hình ảnh, xác định bán kính của hình tròn (chọn bán kích thước nhỏ hơn nửa kích thước ảnh) và tạo một mặt nạ hình tròn bằng cách sử dụng mảng NumPy. Cuối cùng, hàm sẽ áp dụng mặt nạ hình tròn để cắt ảnh và trả về hình ảnh đã được cắt thành hình tròn chính giữa.

Tham số đầu vào:

image: Hình ảnh ban đầu cần cắt thành hình tròn. Đây là một đối tượng PIL.Image.

Kết quả trả về:

Hàm trả về một mảng NumPy biểu diễn hình ảnh sau khi đã được cắt thành hình tròn chính giữa.

Mô tả:

- Chuyển đổi hình ảnh đầu vào thành mảng numpy bằng cách sử dụng hàm np.array.
- Lấy chiều cao và chiều rộng của hình ảnh ban đầu từ mảng numpy đã chuyển đổi.
- Tính tâm của hình ảnh bằng cách chia chiều rộng cho 2 và chiều cao cho 2.

- Tính bán kính của hình tròn bằng cách lấy giá trị nhỏ nhất giữa center_x và center_y.
- Tạo một mặt nạ hình tròn bằng cách sử dụng mảng numpy và biểu thức so sánh mask = $(X - center\_x)^2 + (Y - center\_y)^2 <= radius^2$. Tạo một mặt nạ hình tròn bằng cách kiểm tra từng điểm trên lưới tọa độ có nằm trong vùng hình tròn hay không. Cụ thể, điểm đó nằm trong vùng hình tròn nếu khoảng cách từ nó tới tâm (center_x, center_y) nhỏ hơn hoặc bằng bán kính (radius). Mặt nạ này sẽ có giá trị True tại các vị trí (Y, X) thuộc hình tròn và giá trị False tại các vị trí nằm ngoài hình tròn.
- Áp dụng mặt nạ hình tròn để cắt hình ảnh theo hình tròn. Điều này được thực hiện bằng cách tạo một mảng cropped_image mới có cùng kích thước với hình ảnh gốc và gán các giá trị của hình ảnh gốc tại các vị trí có mask=True. Các vị trí không thuộc hình tròn sẽ có giá trị 0 trong mảng cropped_image.
- Trả về mảng numpy biểu diễn hình ảnh sau khi đã được cắt thành hình tròn chính giữa làm kết quả của hàm.

Ưu điểm:

- Đơn giản và dễ hiểu: Hàm này sử dụng các phép tính cơ bản để thực hiện việc tính toán và tạo mặt nạ hình tròn, dễ hiểu và dễ cài đặt.
- Tính tổng quát: Hàm này có thể được sử dụng để cắt hình ảnh thành hình tròn chính giữa với bất kỳ kích thước nào, giúp tái sử dụng cho nhiều tình huống.

Hình ảnh kết quả:

Ảnh gốc (500x500)

Ảnh cắt hình tròn

xii. crop_ellipse

Công dụng: Hàm crop_ellipse được sử dụng để cắt một hình ảnh thành hình elip chéo dựa trên góc quay đã cho.

Ý tưởng: Hàm này sẽ nhận một hình ảnh đầu vào dưới dạng một đối tượng PIL.Image và một góc quay (angle) được xác định bởi người dùng (mặc định là 45 độ). Sau đó, nó sẽ tính toán tâm của hình ảnh, tạo một mặt nạ elip chéo bằng cách sử dụng mảng numpy, xoay trục của elip và áp dụng mặt nạ elip chéo để cắt hình ảnh theo hình elip chéo. (17, 18, 19, 20, 21).

- Xác định kích thước của ảnh và tìm tâm của ảnh bằng cách tính trung điểm của chiều rộng và chiều cao.
- Tạo một mặt nạ elip có dạng dấu '+' bằng cách sử dụng các phương trình elip đơn giản. Điều này được thực hiện bằng cách tính các điểm thuộc elip thông qua hai bán trục elip. Tất cả các điểm thuộc elip được gán giá trị True, còn lại là False.
- Xoay mặt nạ elip với góc được chỉ định. Để xoay mặt nạ, sử dụng các phép biến đổi hình học để tính toán vị trí mới của mỗi điểm trong mặt nạ sau khi xoay. Cụ thể, tính toán các giá trị mới của x và y dựa trên các giá trị cũ và góc. Sau đó, kiểm tra xem các giá trị mới có nằm trong kích thước của ảnh hay không và cập nhật mặt nạ đã xoay.
- Áp dụng mặt nạ đã xoay để cắt ảnh theo hình dạng elip dấu '+'. Điều này thực hiện bằng cách tạo một mảng mới chứa các giá trị từ ảnh gốc chỉ tại những điểm thuộc elip trong mặt nạ.

Tham số đầu vào:

- image: Hình ảnh ban đầu cần cắt thành hình elip chéo. Đây là một đối tượng PIL.Image.
- angle (tùy chọn): Góc quay của hình elip chéo. Mặc định là 45 độ.

Kết quả trả về:

Hàm trả về một mảng numpy biểu diễn hình ảnh sau khi đã được cắt thành hình elip chéo.

Mô tả:

- Chuyển đổi hình ảnh đầu vào thành mảng numpy. Sau đó, lưu chiều cao và chiều rộng của hình ảnh ban đầu từ mảng numpy đã chuyển đổi.
- Tính tâm của hình ảnh bằng cách chia chiều rộng cho 2 và chiều cao cho 2.

- Tạo một mặt nạ elip chéo bằng cách sử dụng mảng numpy và biểu thức so sánh. Mặt nạ này sẽ có giá trị True tại các vị trí (Y, X) thuộc hai elip chéo và giá trị False tại các vị trí nằm ngoài hai elip chéo.
- Xoay trục của elip bằng cách tính toán các giá trị mới của mặt nạ elip sau khi đã xoay vị trí của mỗi điểm. Việc này được thực hiện bằng cách sử dụng ma trận quay 2D, với cos_a và sin_a là các hàm lượng giác của góc quay.
- Áp dụng mặt nạ elip chéo đã xoay để cắt hình ảnh theo hình elip chéo. Điều này được thực hiện bằng cách tạo một mảng cropped_image mới có cùng kích thước với hình ảnh gốc và gán các giá trị của hình ảnh gốc tại các vị trí có rotated_ellipse_mask=True. Các vị trí không thuộc hình elip chéo sẽ có giá trị 0 trong mảng cropped_image.
- Trả về mảng numpy biểu diễn hình ảnh sau khi đã được cắt thành hình elip chéo làm kết quả của hàm.

Phần chưa làm được:

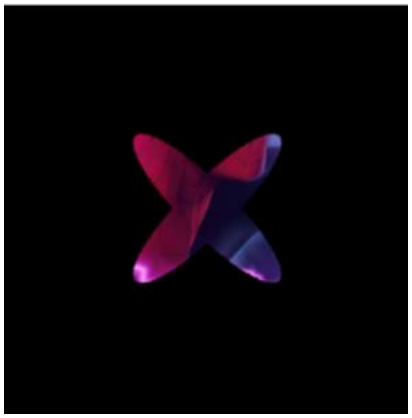
Khung hình elip chéo không bao trọn ảnh mà chỉ nằm lưng chừng, do ý tưởng ban đầu là 2 đường chéo chính của 2 elip chéo là trục x và trục y, sau đó quay khung một góc 45 độ. Việc này dẫn đến 2 hình elip chéo sẽ không phủ hết tấm hình được.

Khung hình ban đầu

Khung hình sau khi quay 45 độ

Ý tưởng cho phần chưa thực hiện:

Kéo dài hình elip ra sao cho trục lớn của hình elip là đường chéo của hình ảnh. Tuy nhiên, khi em thực hiện cách này thì hình ảnh cắt ra không đúng với yêu cầu đưa ra, khung ảnh không đúng.



Khung hình khi thử kéo dài

Ưu điểm:

- Đơn giản và dễ hiểu: Hàm này sử dụng các phép tính cơ bản để thực hiện việc tính toán và tạo mặt nạ elip chéo, dễ hiểu và dễ cài đặt.
- Tính tổng quát: Hàm này có thể được sử dụng để cắt hình ảnh thành hình elip chéo với các góc quay khác nhau, giúp tái sử dụng cho nhiều tình huống.

Nhược điểm:

- Cắt hình elip chéo trên hình ảnh chữ nhật: Hàm này chỉ thực hiện cắt hình elip chéo trên hình ảnh chữ nhật ban đầu. Đối với hình ảnh có tỷ lệ khác nhau (không phải hình ảnh vuông), việc cắt thành hình elip chéo có thể làm biến đổi tỷ lệ của các đối tượng trong hình ảnh.
- Thiếu điều chỉnh: Hàm này không cho phép người dùng điều chỉnh góc quay của hình elip chéo, do đó, không linh hoạt trong việc tùy chỉnh hình dạng và hướng của hình elip chéo.

Hình ảnh kết quả:

Ảnh gốc (500x500)

Ảnh được cắt bởi elip 2 chéo

c. Hàm main

- Hàm main là hàm chính, dung để tổng hợp các hàm chức năng trên để tạo thành một menu, cho phép người dung lựa chọn chức năng mà họ muốn thực thi trên ảnh đầu vào. Các chức năng chính của hàm main.
- Cho phép người dùng nhập tên ảnh vào.
- Cho phép người dung lựa chọn chức năng mà họ muốn với bức ảnh vừa nhập vào khi nãy.

```
Enter the image filename: man.jpg

Menu:
0: Process all functions and save images
1: Adjust brightness and save the image
2: Adjust contrast and save the image
3: Flip the image horizontally/vertically and save it
4: Convert RGB to grayscale and sepia and save images
5: Blur and sharpen the image and save images
6: Crop the image (center) and save it
7: Crop the image in a circular region and save it
8: Crop the image in an elliptical region and save it
9: Exit

Enter your choice (0-9):
```

- Sau khi thực hiện chức năng, chương trình sẽ hiển thị bức ảnh kết quả thông qua sự hỗ trợ của thư viện mayplotlib. Với những bức ảnh đó cũng được lưu về máy theo dạng .jpg với <tên ảnh>_<chức năng>.jpg

3. Tài liệu tham khảo

1. *Adjusting contrast of image purely with numpy. (n.d.). Stack Overflow. https://stackoverflow.com/questions/48406578/adjusting-contrast-of-image-purely-with-numpy*
2. *Bauckhage, C. (2015). NUMPY / SCIPY Recipes for Image Processing: "Simple" intensity Transformations. ResearchGate. https://doi.org/10.13140/RG.2.1.4125.3606*
3. *Numpy.Flipud — NumPy v1.25 Manual. (n.d.). https://numpy.org/doc/stable/reference/generated/numpy.flipud.html*
4. *Numpy.fliplr — NumPy v1.25 Manual. (n.d.-b). https://numpy.org/doc/stable/reference/generated/numpy.fliplr.html*
5. *Why does rgb2gray use these weights for the weighted sum? (n.d.). MATLAB Answers - MATLAB Central. https://www.mathworks.com/matlabcentral/answers/234419-why-does-rgb2gray-use-these-weights-for-the-weighted-sum*
6. *How to convert a color image into sepia image - Image Processing Project - dyclassroom | Have fun learning :-). (n.d.). https://dyclassroom.com/image-processing-project/how-to-convert-a-color-image-into-sepia-image*
7. *Python image convolution using NumPy only. (n.d.). Stack Overflow. https://stackoverflow.com/questions/64587303/python-image-convolution-using-numpy-only*

8. *Jake @Scicoding. (2023). 4 ways to calculate convolution in Python. Scicoding. https://scicoding.com/convolution-in-python-3-essential-packages/*
9. *How to gauss-filter (blur) a floating point numpy array. (n.d.). Stack Overflow. https://stackoverflow.com/questions/29920114/how-to-gauss-filter-blur-a-floating-point-numpy-array*
10. *Blur - Ý Tưởng và Giải Thuật Làm Mờ Ảnh Đơn Giản — Article. (n.d.). IO Stream. https://www.iostream.vn/article/blur-y-tuong-va-giai-thuat-lam-mo-anh-don-gian-FkmL3*
11. *Wikipedia contributors. (2023). Kernel (image processing). Wikipedia. https://en.wikipedia.org/wiki/Kernel_(image_processing)*
12. *Phép Tích Chập trong Xử Lý Ảnh (Convolution) — Article. (n.d.). IO Stream. https://www.iostream.vn/article/phep-tich-chap-trong-xu-ly-anh-convolution-r1vHu1*
13. *Yacine, R. (n.d.). Image Filtering and Blurring with OpenCV and Python | Don't Repeat Yourself. https://dontrepeatyourself.org/post/image-filtering-and-blurring-with-opencv-and-python/*
14. *Pooyakalahroodi. (n.d.). ImageBlurring/pythonImageBlurring.ipynb at main · pooyakalahroodi/ImageBlurring. GitHub. https://github.com/pooyakalahroodi/ImageBlurring/blob/main/pythonImageBlurring.ipynb*

15. *Maharaj, S. (2023). Sharpening An Image using OpenCV Library in Python. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/08/sharpening-an-image-using-opencv-library-in-python/*

16. *2.6.8.7. Image sharpening — Scipy lecture notes. (n.d.). https://scipy-lectures.org/advanced/image_processing/auto_examples/plot_sharpen.html*

17. *Coding with Ashwin. (2021, December 10). Python - NuMPY Image Processing - Masterclass [Crash Course] [Video]. YouTube. https://www.youtube.com/watch?v=BfAtAJGN7Lk*

18. *Michael Galarnyk. (2015, October 24). Ellipse Shaped Image Cropping using MATLAB [Video]. YouTube. https://www.youtube.com/watch?v=igPtFauGoA8*

19. *Kobiso. (n.d.). GitHub - kobiso/Image-Rotation-and-Cropping-tensorflow: Image rotation and cropping out the black borders in TensorFlow. GitHub. https://github.com/kobiso/Image-Rotation-and-Cropping-tensorflow*

20. *How to capture image in ellipse form in OpenCV? (2021, March 24). OpenCV. https://forum.opencv.org/t/how-to-capture-image-in-ellipse-form-in-opencv/2334/3*

21. *Extract an ellipse form from an image instead of drawing it inside - OpenCV Q&A Forum. (n.d.). https://answers.opencv.org/question/25523/extract-an-ellipse-form-from-an-image-instead-of-drawing-it-inside/*