

University of Science
Viet Nam National University - Ho Chi Minh city



**TOÁN ỨNG DỤNG VÀ THỐNG KÊ
CHO CÔNG NGHỆ THÔNG TIN**
Đồ án 2: Image Processing

Giảng viên:
Vũ Quốc Hoàng
Lê Thanh Tùng
Phan Thị Phương Uyên
Nguyễn Văn Quang Huy

Lớp: 21CLC02
Sinh viên thực hiện:
Lê Hoàng Sang - 21127158

Mục lục

1 Lời nói đầu	2
2 Tham khảo	2
3 Đánh giá mức độ hoàn thành các yêu cầu	3
4 Mô tả các hàm bổ trợ	3
5 Mô tả các hàm thực hiện các yêu cầu (1 đến 8)	6
5.1 Thay đổi độ sáng cho ảnh	6
5.2 Thay đổi độ tương phản	7
5.3 Lật ảnh (ngang - dọc)	9
5.4 Chuyển đổi ảnh RGB thành ảnh xám, sepia	9
5.5 Làm mờ/ sắc nét ảnh	11
5.6 Cắt ảnh theo kích thước (cắt ở trung tâm)	14
5.7 Cắt ảnh theo khung hình tròn	15
5.8 Xử lý nhập xuất và hàm Main	16
6 Nhận xét và phân tích thời gian xử lí	17
6.1 Nhận xét	17
6.2 Thống kê thời gian	17
7 Hình minh họa	18

1 Lời nói đầu

- Đồ án này sử dụng ngôn ngữ lập trình Python và thư viện NumPy để xây dựng các chức năng xử lý ảnh. Chúng ta bắt đầu từ những phép biến đổi cơ bản như chuyển đổi màu ảnh sang ảnh xám, cắt và thu nhỏ ảnh, rồi tiếp tục đến những phép biến đổi phức tạp hơn như làm mờ và làm nét ảnh, tăng cường độ tương phản, chỉnh sửa độ sáng và tăng cường màu sắc.
- Điểm nổi bật trong đồ án này là việc thực hiện một số phép biến đổi theo cách thủ công, không sử dụng các thư viện mạnh mẽ hỗ trợ xử lý ảnh như OpenCV hay scikit-image. Điều này giúp hiểu rõ hơn về cách hoạt động của các phép biến đổi và thuật toán xử lý ảnh.
- Ngoài ra, đồ án cũng tập trung vào việc xử lý ảnh hình học. Tất cả những kỹ thuật và chức năng trong đồ án này đều có thể được ứng dụng vào nhiều lĩnh vực như xử lý ảnh y học, phân tích hình ảnh, xử lý ảnh nghệ thuật, và nhiều ứng dụng khác.

2 Tham khảo

- Kiến thức môn Toán Ứng dụng và Thống kê - Khoa Công nghệ thông tin, Trường đại học Khoa học Tự nhiên TPHCM
- Kernel (image processing)
- Grayscale to RGB Conversion
- How to convert a color image into sepia image
- numpy.convolve
- numpy.meshgrid() trong Python
- numpy.clip
- Display image as grayscale using matplotlib
- Gaussian blur
- What is the general equation of the ellipse that is not in the origin and rotated by an angle?
- numpy.pad

3 Đánh giá mức độ hoàn thành các yêu cầu

Bảng 1: Bảng thống kê mức độ hoàn thành các yêu cầu

STT	Chức năng	Đánh giá	Ghi chú
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, sepia	100%	
5	Làm mờ/ sắc nét ảnh	100%	
6	Cắt ảnh theo kích thước (cắt ở trung tâm)	100%	
7	Cắt ảnh theo khung hình tròn	100%	
8	Hàm main xử lý các yêu cầu	100%	
9	Cắt ảnh theo 2 hình elip	0%	Nâng cao

4 Mô tả các hàm bổ trợ

- **Hàm read_image():**

- Input của hàm là tên tệp ảnh mà người dùng muốn đọc và xử lý. Người dùng nhập tên tệp ảnh từ bàn phím khi hàm được gọi.
- Output bao gồm các thành phần sau:
 - * image_path: Tên tệp ảnh (input) mà người dùng đã nhập.
 - * image: Mảng NumPy biểu diễn ảnh RGB được đọc từ tệp ảnh. Đây là dữ liệu ảnh thô, cho phép thực hiện xử lý và biểu diễn các thay đổi trên ảnh.
 - * width: Chiều rộng của ảnh.
 - * height: Chiều cao của ảnh.
- Công dụng và hoạt động của hàm: cho phép người dùng nhập tên tệp ảnh, sau đó đọc và chuyển đổi ảnh thành một mảng NumPy. Dữ liệu trả về là tập hợp các thông tin về ảnh cần thiết để tiếp tục xử lý, sử dụng trong các hàm xử lý và hiển thị ảnh.

- **Hàm is_grayscale(image):**

- Input: Đầu vào của hàm là một mảng NumPy biểu diễn ảnh, có thể là ảnh màu (RGB) hoặc ảnh xám (grayscale).

- Output: Hàm trả về giá trị True hoặc False. Nếu giá trị trả về là True, thì ảnh đầu vào được xem là ảnh xám. Ngược lại, nếu giá trị trả về là False, thì ảnh đầu vào là ảnh màu (RGB).
- Công dụng và hoạt động:
 - * Được sử dụng để kiểm tra xem một ảnh có phải là ảnh xám hay không. Để xác định điều này, hàm kiểm tra kích thước của image.shape để xác định loại ảnh.
 - * Nếu image.shape có 2 chiều (chỉ có hai phần tử: chiều cao và chiều rộng), hàm trả về True. Điều này chỉ ra rằng ảnh là ảnh xám, không có thông tin màu sắc, chỉ có mức xám của từng điểm ảnh.
 - * Ví dụ về ma trận xám kích thước 3x3 và mỗi phần tử trong ma trận đại diện cho mức cường độ ánh sáng của điểm ảnh tại vị trí tương ứng trong ảnh xám. Một ma trận ánh xám có thể được biểu diễn bằng ma trận số nguyên, trong đó giá trị của mỗi phần tử nằm trong khoảng từ 0 đến 255, đại diện cho mức xám từ đen đến trắng:

```

1         pixels_gray = np.array([
2             [100, 200, 50],
3             [150, 75, 180],
4             [90, 120, 220]
5         ])

```

- * Nếu image.shape có 3 chiều và chiều thứ ba của nó bằng 1, hàm cũng trả về True. Trường hợp này cũng chỉ ra rằng ảnh là ảnh xám, tuy nhiên nó được biểu diễn dưới dạng mảng 3D với chỉ một kênh màu.
- * Trong trường hợp còn lại, tức là image là ảnh màu (RGB) được biểu diễn dưới dạng mảng 3D với ba kênh màu (R, G, B), hàm trả về False.

- **Hàm display_image(image):**

- Input: Đầu vào của hàm là một mảng NumPy biểu diễn ảnh, có thể là ảnh màu (RGB) hoặc ảnh xám (grayscale).
- Output: Hàm không trả về giá trị, chỉ thực hiện hiển thị ảnh lên màn hình.
- Công dụng và hoạt động:
 - * Được sử dụng để hiển thị ảnh trên màn hình. Trước khi hiển thị, hàm kiểm tra loại ảnh bằng cách sử dụng hàm is_grayscale(image). Nếu ảnh là ảnh xám, hàm sử dụng plt.imshow với tham số cmap='gray' để hiển thị ảnh với colormap 'gray', làm cho ảnh hiển thị dưới dạng ảnh xám. Ngược lại, nếu ảnh là ảnh màu (RGB), hàm sử dụng plt.imshow(image) để hiển thị ảnh mà không cần sử dụng colormap.

- * Sau khi hiển thị ảnh, hàm plt.show() được sử dụng để hiển thị cửa sổ với ảnh được vẽ lên màn hình terminal.

- **Hàm comparison_display(image_input, image_output):**

- Input: Đầu vào của hàm là hai mảng NumPy biểu diễn hai ảnh: image_input là ảnh gốc (trước khi xử lý), image_output là ảnh đã được xử lý.
- Output: Hàm không trả về giá trị, chỉ thực hiện hiển thị hai ảnh (image_input và image_output) cùng một lúc trên cửa sổ màn hình.
- Công dụng và hoạt động:
 - * Được sử dụng để hiển thị hai ảnh (image_input và image_output) cùng một lúc, giúp người dùng so sánh trực quan giữa ảnh gốc và ảnh sau khi xử lý.
 - * Hàm tạo một hình ảnh đồ thị có hai cột, mỗi cột chứa một ảnh. Cột đầu tiên chứa ảnh gốc (image_input), và cột thứ hai chứa ảnh sau khi xử lý (image_output). Trước khi hiển thị, hàm kiểm tra loại của ảnh image_output bằng cách sử dụng hàm is_grayscale(image_output). Nếu ảnh sau khi xử lý là ảnh xám, hàm sử dụng plt.imshow(image_output, cmap='gray') để hiển thị ảnh với colormap 'gray', làm cho ảnh hiển thị dưới dạng ảnh xám. Ngược lại, nếu ảnh sau khi xử lý là ảnh màu (RGB), hàm sử dụng plt.imshow(image_output) để hiển thị ảnh mà không cần sử dụng colormap.
 - * Cuối cùng, hàm plt.show() được sử dụng để hiển thị cửa sổ với hai ảnh được vẽ lên màn hình terminal cùng một lúc, giúp người dùng dễ dàng so sánh sự thay đổi giữa hai ảnh và hiểu kết quả của quá trình xử lý hình ảnh.

- **Hàm ask_image_format():**

- Input: Hàm không nhận bất kỳ đối số nào.
- Output: Hàm trả về chuỗi đại diện cho định dạng ảnh được chọn sau khi người dùng nhập thông qua giao diện dòng lệnh.
- Công dụng và hoạt động:
 - * Được sử dụng để yêu cầu người dùng nhập định dạng của file ảnh, bao gồm "pdf" hoặc "png". Hàm sẽ tiếp tục yêu cầu nhập cho đến khi người dùng nhập đúng định dạng.
 - * Hàm sử dụng một vòng lặp vô hạn (while True) để yêu cầu người dùng nhập thông tin định dạng. Trong mỗi lần lặp, hàm sử dụng input() để nhận thông tin định dạng từ người dùng và chuyển thành chữ thường (lowercase) bằng cách sử dụng lower() để đảm bảo không bị phân biệt chữ hoa và chữ thường. Sau đó, hàm kiểm tra nếu định dạng là "pdf" hoặc "png" bằng cách

sử dụng điều kiện if. Nếu định dạng hợp lệ, hàm sẽ trả về giá trị định dạng mà người dùng đã nhập.

- * Nếu định dạng không hợp lệ (không phải "pdf" hoặc "png"), hàm sẽ in thông báo yêu cầu người dùng chọn lại định dạng hợp lệ và quay lại vòng lặp để tiếp tục yêu cầu nhập. Việc này tiếp tục cho đến khi người dùng nhập đúng định dạng hợp lệ, lúc đó hàm sẽ trả về giá trị định dạng hợp lệ được chọn.

- **Hàm save_image(rgb_array, image_path, type_edit):**

- Input: Hàm nhận ba tham số:
 - * rgb_array (numpy array): Mảng numpy biểu diễn ảnh màu.
 - * image_path (str): Đường dẫn và tên file ảnh để lưu.
 - * type_edit (str): Loại chỉnh sửa ảnh (được thêm vào tên file ảnh lưu).
- Output: Hàm không trả về giá trị nào, thay vào đó thực hiện lưu ảnh màu đã chỉnh sửa thành file mới.
- Công dụng và hoạt động:
 - * Được sử dụng để lưu ảnh đã chỉnh sửa thành file mới. Hàm sẽ tạo tên file mới bằng cách thêm type_edit vào tên file gốc và sau đó lưu ảnh thành file mới với định dạng được chọn.
 - * Hàm sử dụng câu lệnh image_path.split() để tách đuôi mở rộng của file gốc, sau đó thêm type_edit và đuôi mở rộng mới (image_format) để tạo tên file mới. Một định dạng ảnh được gán cứng là "png" bằng cách sử dụng image_format = "png". Ngoài ra, người dùng có thể lựa chọn định dạng bằng cách sử dụng ask_image_format() nếu cần.
 - * Tiếp theo, mảng numpy rgb_array được chuyển thành kiểu dữ liệu np.uint8 để đảm bảo rằng nó có giá trị từ 0 đến 255 (trong kiểu dữ liệu 8-bit unsigned integer) để lưu thành file ảnh. Mảng này sau đó được chuyển thành đối tượng PIL.Image thông qua PIL.Image.fromarray(rgb_array), sau đó hàm gọi image.save(image_path + image_format) để lưu ảnh thành file mới với tên và định dạng tương ứng.

5 Mô tả các hàm thực hiện các yêu cầu (1 đến 8)

5.1 Thay đổi độ sáng cho ảnh

Hàm change_brightness(image, brightness_level = 40):

- Input: Hàm `change_brightness(image, brightness_level=40)` nhận hai tham số:
 - `image` (numpy array): Mảng NumPy biểu diễn ảnh màu.
 - `brightness_level` (int): Mức độ điều chỉnh độ sáng của ảnh, mặc định là 40.
- Output: Hàm trả về mảng NumPy biểu diễn ảnh màu đã được điều chỉnh độ sáng.
- Công dụng: hiện việc điều chỉnh độ sáng bằng cách thêm `brightness_level` vào từng kênh màu của ảnh (kênh đỏ, xanh lá cây và xanh dương). Trước khi thêm, `brightness_level` được chuyển thành mảng NumPy để có thể broadcast và thêm vào mỗi kênh màu một cách độc lập.
- Hoạt động:
 - Đầu tiên, hàm chuyển đổi kiểu dữ liệu của ảnh từ `np.uint8` thành `np.uint16`. Kiểu dữ liệu `np.uint16` cho phép các giá trị cường độ ánh sáng của điểm ảnh có giá trị lớn hơn 255.
 - Tiếp theo, hàm thực hiện phép cộng `brightness_level` vào từng điểm ảnh trong ảnh. Điều này tương đương với việc tăng hoặc giảm độ sáng của ảnh, tùy thuộc vào giá trị của `brightness_level`.
 - Sau khi thực hiện phép cộng, các giá trị cường độ ánh sáng có thể vượt quá khoảng từ 0 đến 255, gây lỗi trong việc hiển thị ảnh. Do đó, hàm sử dụng `np.clip` để giới hạn giá trị cường độ ánh sáng trong khoảng hợp lệ từ 0 đến 255. Nếu giá trị cường độ vượt quá khoảng này, nó sẽ được cắt đi (hàm clip) và giữ lại ở giá trị gần nhất trong khoảng.
 - Hàm `np.clip()` trong thư viện NumPy được sử dụng để giới hạn giá trị của một mảng (array) nằm trong một khoảng xác định.
 - Cú pháp:

```
1 np.clip(a, a_min, a_max, out=None)
```

- Cuối cùng, hàm chuyển đổi kiểu dữ liệu của ảnh trở lại thành `np.uint8` để đảm bảo rằng các giá trị cường độ ánh sáng nằm trong khoảng hợp lệ từ 0 đến 255.
- Kết quả của hàm là một ảnh mới có độ sáng được điều chỉnh theo `brightness_level`, với giá trị cường độ ánh sáng được giới hạn trong khoảng hợp lệ từ 0 đến 255.

5.2 Thay đổi độ tương phản

Hàm `adjust_contrast(image, contrast_level = 1.4)`:

- Input:
 - image: Đây là ma trận biểu diễn hình ảnh ban đầu. Đối với hình ảnh màu, ma trận này có kích thước (chiều cao, chiều rộng, 3) với 3 kênh màu (RGB). Đối với hình ảnh xám, ma trận này có kích thước (chiều cao, chiều rộng).
 - contrast_level (tùy chọn): Đây là mức độ điều chỉnh độ tương phản của hình ảnh.
- Output: Đây là ma trận biểu diễn hình ảnh đã điều chỉnh độ tương phản. Đối với hình ảnh màu, ma trận này có cùng kích thước (chiều cao, chiều rộng, 3) và chứa các giá trị màu mới đã được điều chỉnh tương ứng.
- Công dụng: điều chỉnh tương phản của ảnh đầu vào bằng cách nhân mỗi điểm ảnh với một giá trị số thực (contrast_level). Nói cách khác, hàm này thực hiện việc tăng hoặc giảm tương phản của ảnh dựa trên giá trị của contrast_level.
- Hoạt động:
 - Đầu tiên, hàm chuyển đổi kiểu dữ liệu của ảnh từ np.uint8 thành np.uint16. Kiểu dữ liệu np.uint16 cho phép các giá trị cường độ ánh sáng của điểm ảnh có giá trị lớn hơn 255.
 - Hàm nhân mỗi điểm ảnh trong ảnh với giá trị contrast_level. Việc nhân này tạo ra sự thay đổi về tương phản trong ảnh. Nếu contrast_level > 1, tương phản của ảnh sẽ tăng, làm cho các vùng sáng trở nên sáng hơn và các vùng tối trở nên tối hơn. Ngược lại, nếu contrast_level < 1, tương phản của ảnh sẽ giảm, làm cho sự khác biệt giữa các cường độ ánh sáng gần nhau hơn, mất mát chi tiết và độ tương phản trong ảnh.
 - Sau khi thực hiện phép nhân, các giá trị cường độ ánh sáng có thể vượt quá khoảng từ 0 đến 255, gây lỗi trong việc hiển thị ảnh. Do đó, hàm sử dụng np.clip để giới hạn giá trị cường độ ánh sáng trong khoảng hợp lệ từ 0 đến 255. Nếu giá trị cường độ vượt quá khoảng này, nó sẽ được cắt đi (hàm clip) và giữ lại ở giá trị gần nhất trong khoảng.
 - Cuối cùng, hàm chuyển đổi kiểu dữ liệu của ảnh trở lại thành np.uint8 để đảm bảo rằng các giá trị cường độ ánh sáng nằm trong khoảng hợp lệ từ 0 đến 255.
 - Kết quả của hàm là một ảnh mới có tương phản được điều chỉnh theo contrast_level, với giá trị cường độ ánh sáng được giới hạn trong khoảng hợp lệ từ 0 đến 255.

5.3 Lật ảnh (ngang - dọc)

Hàm flip(image, option = "horizontal"):

- Input: Hàm nhận hai đối số:
 - image (NumPy array): Mảng NumPy biểu diễn ảnh màu hoặc ảnh xám.
 - option (str): Lựa chọn hướng lật ảnh. Mặc định là "horizontal" (lật theo chiều ngang).
- Output: Hàm trả về mảng NumPy biểu diễn ảnh sau khi đã được lật theo hướng chỉ định.
- Công dụng: được sử dụng để lật ảnh theo hướng ngang hoặc dọc, tham số option xác định hướng lật ảnh.
- Hoạt động:
 - Khi option là "horizontal" (mặc định), hàm thực hiện lật ảnh theo chiều ngang bằng cách sử dụng cú pháp `image[:, ::-1]`. Điều này có nghĩa là hàm sẽ giữ nguyên các hàng của ảnh (theo trục y), nhưng thay đổi thứ tự của các cột (theo trục x) bằng cách lật theo chiều ngang.
 - Khi option là "vertical", hàm thực hiện lật ảnh theo chiều dọc bằng cách sử dụng cú pháp `image[::-1]`. Điều này có nghĩa là hàm sẽ giữ nguyên các cột của ảnh (theo trục x), nhưng thay đổi thứ tự của các hàng (theo trục y) bằng cách lật theo chiều dọc.
 - Nếu option không phải "horizontal" hoặc "vertical", hàm trả về ảnh gốc không thay đổi.
 - Hàm này sử dụng cú pháp slicing của NumPy để thực hiện lật ảnh một cách nhanh chóng và hiệu quả.

5.4 Chuyển đổi ảnh RGB thành ảnh xám, sepia

Hàm convert(image, option="gray"):

- Input: Hàm `convert(image, option="gray")` nhận hai đối số:
 - image (NumPy array): Mảng NumPy biểu diễn ảnh màu.
 - option (str): Lựa chọn chế độ chuyển đổi màu. Mặc định là "gray" (chuyển đổi sang ảnh xám).

- Output: Hàm trả về mảng NumPy biểu diễn ảnh sau khi đã được chuyển đổi theo lựa chọn chỉ định.
- Công dụng và hoạt động:
 - Được sử dụng để chuyển đổi một ảnh màu thành ảnh xám hoặc ảnh sepia (nâu kiểu cổ điển).
 - Khi option là "gray", hàm sẽ thực hiện chuyển đổi ảnh màu thành ảnh xám. Để chuyển đổi, hàm sử dụng một trọng số (weight) cho mỗi kênh màu (red, green, blue) của ảnh gốc.
 - Trong đó, `image[..., :3]` là các kênh màu (red, green, blue) của ảnh gốc. Các kênh màu này sẽ được nhân với trọng số tương ứng và tính tổng để tạo ra ảnh xám. Các giá trị của ảnh xám sẽ nằm trong khoảng từ 0 đến 255.
 - Khi option là "sepia", hàm sẽ thực hiện chuyển đổi ảnh màu thành ảnh sepia. Để chuyển đổi, hàm sử dụng một ma trận trọng số (weight) được định nghĩa trước cho mỗi kênh màu của ảnh gốc.
 - Trong đó, `image[..., :3]` là các kênh màu (red, green, blue) của ảnh gốc. Các kênh màu này sẽ được nhân với ma trận trọng số và tính tổng theo từng pixel để tạo ra ảnh sepia. Sau đó, các giá trị của ảnh sepia sẽ được kiểm tra để đảm bảo nằm trong khoảng từ 0 đến 255.
 - Nếu options không phải là "gray" hoặc "sepia", hàm sẽ thông báo lỗi và trả về ảnh gốc không thay đổi.

Giải thích về các ma trận trọng số:

- Ma trận trọng số [0.3, 0.59, 0.11] được sử dụng trong phương pháp chuyển đổi ảnh màu sang ảnh xám (grayscale). Phương pháp chuyển đổi này được gọi là "average grayscale" hoặc "luminosity grayscale".
- Khi chuyển đổi ảnh màu sang ảnh xám, chúng ta cần xác định giá trị độ sáng của mỗi pixel trong ảnh xám. Một cách thông thường để tính giá trị độ sáng này là lấy trung bình có trọng số của các kênh màu (Red, Green, Blue) của mỗi pixel. Ma trận trọng số [0.3, 0.59, 0.11] được thiết kế dựa trên việc mắt người thường nhạy cảm với màu xanh lá cây hơn là màu đỏ và màu xanh dương.
- Cụ thể, giá trị độ sáng (luminosity) của mỗi pixel mới được tính bằng cách lấy trung bình có trọng số của các kênh màu theo công thức sau:

```

1     New_pixel = 0.3 * Red_pixel
2             + 0.59 * Green_pixel
3             + 0.11 * Blue_pixel

```

- Trong đó, Red_pixel, Green_pixel, Blue_pixel là giá trị của các kênh màu trong pixel gốc, và New_pixel là giá trị độ sáng mới của pixel trong ảnh xám.
- Sử dụng ma trận trọng số [0.3, 0.59, 0.11] cho phép chuyển đổi ảnh màu thành ảnh xám sao cho độ sáng được tính toán một cách phù hợp với độ nhạy của mắt người đối với các kênh màu khác nhau.
- Tương tự, ma trận trọng số [[0.393, 0.769, 0.189], [0.349, 0.686, 0.168], [0.272, 0.534, 0.131]] được sử dụng trong phương pháp chuyển đổi ảnh sang hiệu ứng sepia.
- Ma trận trọng số này được sử dụng để tính trung bình có trọng số của các kênh màu (R, G, B) trong ảnh. Cụ thể, mỗi pixel có giá trị mới được tính theo công thức sau:

```

1      New_Red_pixel = 0.393 * Red_pixel + 0.769 *
2          Green_pixel + 0.189 * Blue_pixel
2      New_Green_pixel = 0.349 * Red_pixel + 0.686 *
3          Green_pixel + 0.168 * Blue_pixel
3      New_Blue_pixel = 0.272 * Red_pixel + 0.534 *
        Green_pixel + 0.131 * Blue_pixel

```

- Trong đó, (Red_pixel, G_pixel, B_pixel) là giá trị của các kênh màu trong pixel gốc, và (New_Red_pixel, New_Green_pixel, New_Blue_pixel) là giá trị của các kênh màu trong pixel mới sau khi áp dụng hiệu ứng sepia.
- Chúng ta có thể áp dụng ma trận trọng số này để tạo ra hiệu ứng sepia trên ảnh màu bằng cách tính toán các kênh màu mới như trong hàm convert() đã được đề cập trước đó.

5.5 Làm mờ/ sắc nét ảnh

Hàm custom_sharpness(image, kernel, kernel_size):

- Input:
 - image: Ảnh đầu vào được đại diện bằng một mảng NumPy có kích thước (height, width, channels) với các giá trị màu nằm trong khoảng [0, 255].
 - kernel: Một ma trận (kernel_size x kernel_size) chứa các trọng số tùy chỉnh được sử dụng trong phép tích chập. Kernel này thường có các trọng số dương ở vị trí giữa và trọng số âm ở các vị trí xung quanh giữa để làm tăng độ tương phản và làm nổi bật các chi tiết trong ảnh.

- `kernel_size`: Kích thước của kernel, là một số nguyên dương lẻ, ví dụ: 3, 5, 7,... để đảm bảo tính đối xứng của ma trận xung quanh một điểm đang xét trong ma trận đầu vào.
- Output: Một ảnh mới sau khi thực hiện phép tích chập với kernel tùy chỉnh. Ảnh này có cùng kích thước và số kênh màu với ảnh đầu vào, các giá trị màu nằm trong khoảng [0, 255].
- Công dụng: giúp làm nổi bật các chi tiết và đường nét trong ảnh, tăng độ tương phản, rực rỡ.
- Hoạt động:
 - Đầu tiên, ta tạo một ma trận trống `blurred_image` có kích thước và kiểu dữ liệu tương tự với ảnh đầu vào để lưu trữ kết quả làm nổi bật chi tiết.
 - Thêm viền cho ảnh đầu vào bằng cách bọc các cạnh của ảnh với các giá trị màu từ các điểm biên của ảnh ban đầu. Viền này có kích thước (`kernel_size // 2`, phép chia nguyên trong Python) và được thêm vào các cạnh theo chiều dọc và chiều ngang của ảnh để trường hợp xét các điểm rìa viền ảnh vẫn đảm bảo có đủ dòng cột tạo thành ma trận vuông xung quanh nó.
 - Duyệt qua từng kênh màu của ảnh và từng điểm ảnh trong ảnh gốc, tại mỗi điểm ảnh, tính tổng các phần tử của kernel nhân với các giá trị màu tương ứng trong một vùng của ảnh. Kết quả tích chập được lưu vào ma trận `blurred_image`.
 - Đảm bảo giá trị màu trong ảnh nằm trong khoảng [0, 255] bằng cách sử dụng hàm `np.clip`, sau đó chuyển kiểu dữ liệu của ma trận `blurred_image` về `np.uint8`. Cuối cùng, hàm trả về ảnh đã làm nổi bật chi tiết và tăng độ tương phản.

Hàm `custom_sharpness_handler(image, option)`:

- Input:
 - `image`: Ảnh đầu vào được đại diện bằng một mảng numpy có kích thước (`height, width, channels`) với các giá trị màu nằm trong khoảng [0, 255].
 - `options`: Một chuỗi xác định loại chức năng cần thực hiện trên ảnh. Có hai tùy chọn hợp lệ là "blur" (làm mờ) và "sharp" (làm sắc nét).
- Output: Một ảnh mới sau khi thực hiện chức năng tùy chỉnh. Ảnh này có cùng kích thước và số kênh màu với ảnh đầu vào, các giá trị màu nằm trong khoảng [0, 255].
- Công dụng: cho phép người dùng thực hiện các chức năng tùy chỉnh trên độ nét và độ tương phản của ảnh, làm mờ (blur) hoặc làm sắc nét (sharp) ảnh.

- Hoạt động:

- Kiểm tra giá trị của options để xác định chức năng tùy chỉnh cần thực hiện trên ảnh.
- Nếu option là "blur", tạo một kernel ma trận tùy chỉnh 5×5 có các giá trị được xác định bằng cách chia tất cả các phần tử của kernel bởi 256. Đây là kernel để thực hiện làm mờ ảnh.
- Nếu options là "sharp", tạo một kernel ma trận tùy chỉnh 3×3 để thực hiện làm sắc nét ảnh.
- Gọi hàm `custom_sharpness(image, kernel, kernel_size)` để thực hiện chức năng tùy chỉnh trên ảnh với kernel tương ứng.
- Trả về ảnh đã được thực hiện chức năng tùy chỉnh. Nếu option không hợp lệ, trả về ảnh gốc.

Giải thích về ma trận Kernel:

- Kernel Gaussian Blur: đây là một trong những kernel phổ biến nhất được sử dụng để làm mờ ảnh. Kernel này có hình dạng giống với hàm Gaussian 2D và được sử dụng để làm mờ ảnh bằng cách tính trung bình có trọng số của các pixel xung quanh.
- Kernel này được sử dụng trong phép tích chập với ảnh đầu vào để làm mờ ảnh. Phép tích chập giữa kernel và ảnh đầu vào được thực hiện bằng cách di chuyển kernel qua từng vị trí trên ảnh và tính trung bình có trọng số của các pixel xung quanh. Kết quả là một ảnh mới với các pixel được làm mờ.
- Kernel Gaussian Blur giúp làm mờ ảnh và loại bỏ các chi tiết nhỏ không mong muốn trong ảnh. Nó là một công cụ hữu ích để giảm nhiễu và làm mờ các vùng không cần thiết trong ảnh, giúp làm nổi bật các đối tượng chính. Làm mờ ảnh cũng có thể làm giảm kích thước tệp ảnh và giảm độ phức tạp tính toán trong một số trường hợp xử lý ảnh.
- Kernel Gaussian Blur có nguồn gốc từ lý thuyết xác suất và phân phối Gauss (Gaussian distribution). Lý do người ta sử dụng kernel này trong phép làm mờ ảnh là bởi vì tính chất đặc biệt của phân phối Gauss làm cho nó trở thành một lựa chọn lý tưởng cho việc làm mờ ảnh.
- Tính chất của phân phối Gauss:
 - Đôi xứng: Phân phối Gauss là đối xứng qua giá trị trung bình, điều này có nghĩa là giá trị trung bình là điểm đối xứng của phân phối.

- Đỉnh độc lập: Phân phối Gauss có một đỉnh duy nhất và không có điểm đặc biệt nào khác nằm ở cùng một giá trị y.
- Tính chất ổn định: Tích hai phân phối Gauss lại tạo thành một phân phối Gauss.
- Tính chất trung tâm: Giá trị trung bình của phân phối Gauss là vị trí của đỉnh.
- Kernel Sharp là một kernel được sử dụng trong các phép biến đổi hình ảnh để làm tăng độ nét và làm rõ các đặc trưng trong hình ảnh. Và kernel này giúp tăng độ tương phản và làm nổi bật các biên cạnh trong ảnh.
- Cách hoạt động của Kernel Sharp:
 - Trung tâm của kernel có giá trị là 5, làm cho pixel ở tâm trở nên nổi bật hơn.
 - Các giá trị xung quanh trung tâm bằng -1, khi nhân với các pixel xung quanh, nó sẽ làm giảm giá trị của các pixel này so với pixel trung tâm, giúp tạo ra hiệu ứng làm nổi bật biên cạnh và đặc trưng trong hình ảnh.
 - Các pixel ở ngoài biên cạnh sẽ có giá trị bằng 0, nên khi nhân với các pixel này, nó không ảnh hưởng đến giá trị của pixel trung tâm và các pixel xung quanh.

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

Hàm `center_cut(image, padding = 250)`:

- Input:
 - `image`: Ảnh đầu vào được đại diện bằng một mảng NumPy có kích thước (height, width, channels) với các giá trị màu nằm trong khoảng [0, 255].
 - `padding`: Số nguyên xác định lượng lề cần loại bỏ từ các cạnh của ảnh, giá trị mặc định là 250.
- Output: Một ảnh mới sau khi đã cắt ở phần trung tâm và loại bỏ lề. Ảnh này có kích thước mới là (`min_size`, `min_size`, `channels`), trong đó `min_size` là kích thước nhỏ hơn giữa chiều cao và chiều rộng của ảnh gốc.
- Công dụng: được sử dụng để cắt ảnh ở phần trung tâm và giữ lại vùng ảnh đó, loại bỏ 4 phần lề.
- Hoạt động:
 - Đầu tiên, ta lấy kích thước của ảnh đầu vào (chiều cao, chiều rộng) và số kênh màu.

- Tính kích thước nhỏ hơn giữa chiều cao và chiều rộng của ảnh gốc trừ đi lượng lề xác định (`min_size = height - padding`).
- Tính tọa độ (`top_left_x, top_left_y`) của điểm góc trên bên trái của vùng cần cắt sao cho ảnh cắt ra nằm ở phần trung tâm của ảnh gốc (`top_left_x = (width - min_size) // 2, top_left_y = (height - min_size) // 2`) bằng cách chia nguyên.
- Thực hiện cắt ảnh từ tọa độ (`top_left_x, top_left_y`) tới (`top_left_x + min_size, top_left_y + min_size`) để lấy phần trung tâm của ảnh.
- Cuối cùng, ta trả về ảnh sau khi đã cắt và loại bỏ lề.

5.7 Cắt ảnh theo khung hình tròn

Hàm `rounded_cut(image)`:

- Input: ảnh đầu vào được đại diện bằng một mảng NumPy có kích thước (`height, width, channels`) với các giá trị màu nằm trong khoảng [0, 255].
- Output: một ảnh mới sau khi đã cắt thành hình tròn. Các điểm ảnh nằm ngoài đường tròn sẽ bị loại bỏ (đặt giá trị màu bằng 0), và chỉ giữ lại các điểm ảnh nằm trong hình tròn.
- Công dụng: được sử dụng để cắt ảnh thành hình tròn và giữ lại phần ảnh nằm trong hình tròn, loại bỏ phần ảnh nằm ngoài hình tròn.
- Hoạt động:

- Đầu tiên, ta lấy kích thước của ảnh đầu vào (chiều cao, chiều rộng) và số kênh màu.
- Xác định tọa độ (`center_x, center_y`) của tâm hình tròn là trung tâm của ảnh (`center_x = width // 2, center_y = height // 2`).
- Tính bán kính của hình tròn là giá trị nhỏ hơn chọn ra từ chiều rộng và chiều cao của ảnh trước khi cắt (`radius = min(width, height) // 2`) để đảm bảo hình tròn nội tiếp ảnh.
- Phương trình đường tròn với tâm (`center_x, center_y`) và bán kính `radius`:

$$(x - center_x)^2 + (y - center_y)^2 = radius^2$$

- Tạo ma trận lưới các điểm trong ảnh.

- Hàm np.meshgrid() trong thư viện NumPy được sử dụng để tạo ra các ma trận 2D có các giá trị tương ứng với tọa độ (x, y) của các điểm trong một mặt phẳng. Hàm này hỗ trợ tạo ma trận tọa độ từ một dãy giá trị của trục x và một dãy giá trị của trục y.
- Tính khoảng cách từ từng điểm đến tâm hình tròn. Và tạo mặt nạ (mask) bằng cách đánh dấu các điểm ảnh nằm trong hình tròn (khoảng cách đến tâm nhỏ hơn hoặc bằng bán kính) bằng giá trị True, các điểm nằm ngoài hình tròn được đánh dấu là False.
- Copy ảnh gốc để làm ảnh cắt (cropped_image = image.copy()). Đặt giá trị màu của các điểm nằm ngoài hình tròn (theo mặt nạ) thành 0 để loại bỏ chúng và trả về ảnh sau khi đã cắt thành hình tròn.

5.8 Xử lý nhập xuất và hàm Main

Hàm user_handler(image_path, image, choice):

- Được thiết kế để thực hiện các chức năng xử lý ảnh dựa vào lựa chọn của người dùng. Điểm đặc biệt trong hàm này là việc sử dụng đệ quy để thực hiện tất cả các chức năng xử lý ảnh khi người dùng chọn lựa chọn '0'.
- Input:
 - image_path: Đường dẫn tới tập tin ảnh.
 - image: Ảnh đầu vào được biểu diễn bằng một mảng numpy có kích thước (height, width, channels) với các giá trị màu nằm trong khoảng [0, 255].
 - choice: Lựa chọn của người dùng (chữ số từ 1 đến 7, hoặc '0' nếu muốn thực hiện tất cả các chức năng).
- Output: Thời gian thực hiện chức năng xử lý ảnh (trong giây) của chức năng được lựa chọn.
- Công dụng: Thực hiện các chức năng xử lý ảnh dựa trên lựa chọn của người dùng. Nếu lựa chọn là '0', hàm sẽ thực hiện tất cả các chức năng xử lý ảnh một lượt và trả về tổng thời gian thực hiện.
- Hoạt động:
 - Nếu choice là '0', hàm sẽ thực hiện lần lượt từng chức năng từ 1 đến 7 bằng cách gọi đệ quy.
 - Dựa vào choice, hàm sẽ thực hiện các chức năng tương ứng như sau:

- * '1': Thay đổi độ sáng cho ảnh (hàm change_brightness).
- * '2': Thay đổi độ tương phản cho ảnh (hàm adjust_contrast).
- * '3': Lật ảnh (ngang hoặc dọc) (hàm flip).
- * '4': Chuyển đổi ảnh RGB thành ảnh xám và sepia (hàm convert).
- * '5': Làm mờ và sắc nét ảnh (hàm custom_sharpness_handler).
- * '6': Cắt ảnh theo kích thước (cắt ở trung tâm) (hàm center_cut).
- * '7': Cắt ảnh theo khung hình tròn (hàm rounded_cut).
- Hiển thị kết quả và lưu ảnh sau mỗi chức năng.
- Tính tổng thời gian thực hiện các chức năng (nếu chọn '0') hoặc trả về thời gian (nếu chọn từng chức năng).

6 Nhận xét và phân tích thời gian xử lý

6.1 Nhận xét

- Nhận xét 1: Đa số các chức năng đều thực hiện tốt với ảnh có kích thước nhỏ và trung bình. Chức năng làm mờ và làm nét trở lại cho thời gian lâu nhất.
- Nhận xét 2: Sử dụng convolution thay vì các vòng lặp cho việc xử lý ảnh nhanh hơn. Tuy nhiên, việc dùng hàm convolution khá dễ sai dẫn đến hình bị sai màu.
- Nhận xét 3: Hàm user_handler có thể cải thiện tính tối ưu của hàm bằng cách sử dụng các hàm song song để xử lý nhiều chức năng cùng lúc.
- Nhận xét 4: Có thể tối ưu hóa việc lưu ảnh bằng cách sử dụng các định dạng hình ảnh có nén như JPEG (một định dạng hình ảnh mất mát) thay vì PNG (định dạng hình ảnh không mất mát) để giảm kích thước tập tin, thời gian và không gian lưu trữ.

6.2 Thống kê thời gian

- Thời gian xử lý của thuật toán được chạy thử trên các ảnh với các kích thước 512 pixels, và không kể thời gian chương trình chờ người dùng nhập các thông tin như tên ảnh, định dạng.
- Việc chạy thuật toán và thống kê thời gian được thực hiện trên laptop Macbook Pro M1 và ứng dụng Visual Studio Code.

Bảng 2: Bảng thống kê thời gian chạy của ảnh lena.png kích thước 512×512

STT	Chức năng	Thời gian (giây)	Đánh giá	Ghi chú
1	Thay đổi độ sáng cho ảnh	0,000554s	100%	
2	Thay đổi độ tương phản	0,002959s	100%	
3	Lật ảnh (ngang - dọc)	0,000001s	100%	
4	Chuyển đổi ảnh RGB thành ảnh xám, sepia	0,0194s	100%	Cả gray và sepia
5	Làm mờ/ sắc nét ảnh	2,82485s	100%	Chạy cả 2
6	Cắt ảnh theo kích thước (cắt ở trung tâm)	0,00000179s	100%	
7	Cắt ảnh theo khung hình tròn	0,0042779s	100%	
8	Chạy tất cả các yêu cầu	2,80126s	100%	Lựa chọn 0

Bảng 3: Bảng thống kê thời gian chạy của ảnh phong cảnh khác kích thước 512×512

STT	Chức năng	Thời gian (giây)	Đánh giá	Ghi chú
1	Thay đổi độ sáng cho ảnh	0,001325s	100%	
2	Thay đổi độ tương phản	0,00455s	100%	
3	Lật ảnh (ngang - dọc)	0,0000013s	100%	
4	Chuyển đổi ảnh RGB thành ảnh xám, sepia	0,02482s	100%	Cả gray và sepia
5	Làm mờ/ sắc nét ảnh	3,7744s	100%	Chạy cả 2
6	Cắt ảnh theo kích thước (cắt ở trung tâm)	0,000001s	100%	
7	Cắt ảnh theo khung hình tròn	0,00733s	100%	
8	Chạy tất cả các yêu cầu	3,770708s	100%	Lựa chọn 0

7 Hình minh họa

Bảng 4: Bảng thống kê thời gian chạy của ảnh phong cảnh kích thước to hơn (1080 x 1080)

STT	Chức năng	Thời gian (giây)	Đánh giá	Ghi chú
1	Thay đổi độ sáng cho ảnh	0,00140s	100%	
2	Thay đổi độ tương phản	0,00747s	100%	
3	Lật ảnh (ngang - dọc)	0,000005s	100%	
4	Chuyển đổi ảnh RGB thành ảnh xám, sepia	0,02018s	100%	Cả gray và sepia
5	Làm mờ/ sắc nét ảnh	4,611988s	100%	Chạy cả 2
6	Cắt ảnh theo kích thước (cắt ở trung tâm)	0,000002s	100%	
7	Cắt ảnh theo khung hình tròn	0,0091s	100%	
8	Chạy tất cả các yêu cầu	4,58293s	100%	Lựa chọn 0



Hình 1: Ảnh gốc 512 x 512



Hình 2: Ảnh sau tăng độ sáng



Hình 3: Ảnh sau khi tăng độ tương phản



Hình 4: Ảnh sau khi lật theo chiều dọc



Hình 5: Ảnh sau chuyển về ảnh xám



Hình 6: Ảnh sau khi áp bộ lọc Sepia



Hình 7: Ảnh sau khi làm mờ



Hình 8: Ảnh mờ sau khi làm nét lại



Hình 9: Ảnh sau khi cắt ở trung tâm



Hình 10: Ảnh mờ sau khi cắt theo khung tròn



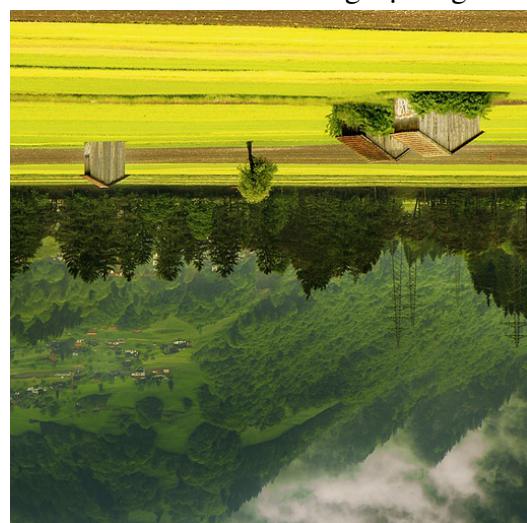
Hình 11: Ảnh gốc 512 x 512



Hình 12: Ảnh sau tăng độ sáng



Hình 13: Ảnh sau khi tăng độ tương phản



Hình 14: Ảnh sau khi lật theo chiều ngang



Hình 15: Ảnh sau chuyển về ảnh xám



Hình 16: Ảnh sau khi áp bộ lọc Sepia



Hình 17: Ảnh sau khi làm mờ



Hình 18: Ảnh mờ sau khi làm nét lại



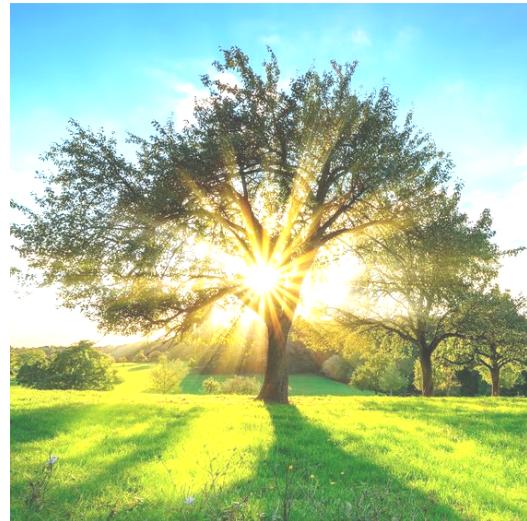
Hình 19: Ảnh sau khi cắt ở trung tâm



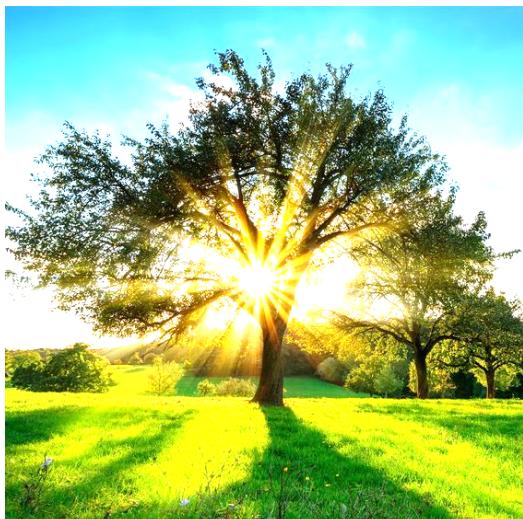
Hình 20: Ảnh mờ sau khi cắt theo khung tròn



Hình 21: Ảnh gốc 1080 x 1080



Hình 22: Ảnh sau tăng độ sáng



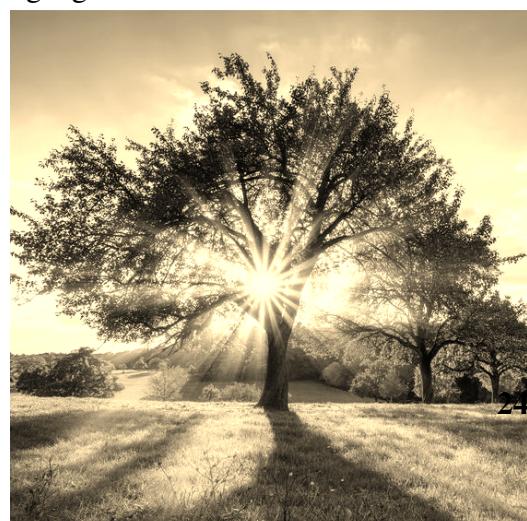
Hình 23: Ảnh sau khi tăng độ tương phản



Hình 24: Ảnh sau khi lật theo chiều ngang



Hình 25: Ảnh sau chuyển về ảnh xám



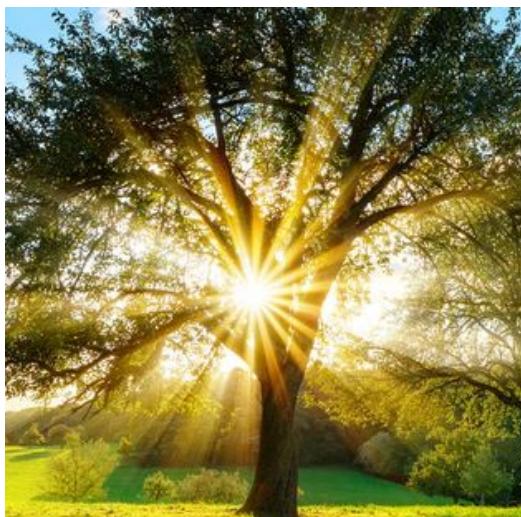
Hình 26: Ảnh sau khi áp bộ lọc Sepia



Hình 27: Ảnh sau khi làm mờ



Hình 28: Ảnh mờ sau khi làm nét lại



Hình 29: Ảnh sau khi cắt ở trung tâm



Hình 30: Ảnh mờ sau khi cắt theo khung tròn