

Trường Đại học Khoa học Tự nhiên ĐHQG-HCM
Khoa Công nghệ Thông tin



BÁO CÁO
Đồ án 2: IMAGE PROCESSING

Môn học

Toán Ứng dụng và thống kê cho CNTT
(MTH00057)

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

Họ và tên Lê Thị Hoài Thư
MSSV 21127176
Lớp 21CLC02

Ngày 1 tháng 8 năm 2023

1 Tổng quan

Nội dung đồ án

Đồ án yêu cầu thực hiện các chức năng xử lý ảnh cơ bản sau:

- Thay đổi độ sáng cho ảnh
- Thay đổi độ tương phản cho ảnh
- Lật ảnh (ngang-dọc)
- Chuyển đổi ảnh RGB thành ảnh xám/sepia
- Làm mờ/sắc nét ảnh
- Cắt ảnh theo kích thước (cắt ở trung tâm)
- Mask ảnh theo khung hình tròn
- Mask ảnh theo khung ellipse (nâng cao)

Nội dung báo cáo

Báo cáo được viết nhằm giới thiệu và trình bày những nội dung liên quan đến đồ án bao gồm những tiêu chí được giảng viên yêu cầu như:

- Liệt kê các chức năng đã hoàn thành.
- Ý tưởng thực hiện, mô tả các hàm chức năng.
- Hình ảnh kết quả với từng chức năng.
- Nhận xét về các kết quả trên.
- Tài liệu tham khảo.

Mục lục

1	Tổng quan	1
2	Đánh giá mức độ hoàn thành	2
3	Tổ chức và ý tưởng, mô tả các hàm	2
3.1	Nhóm hàm chức năng	3
3.2	Nhóm hàm lấy dữ liệu (từ người dùng, từ ảnh, ...)	8
3.3	Nhóm hàm xuất dữ liệu	8
3.4	Nhóm hàm điều phối chương trình	9
4	Kết quả thử nghiệm	10
5	Nguồn tham khảo	15

2 Đánh giá mức độ hoàn thành

Yêu cầu chức năng	Tự đánh giá
Thay đổi độ sáng cho ảnh	100%
Thay đổi độ tương phản cho ảnh	100%
Lật ảnh (ngang-dọc)	100%
Chuyển đổi ảnh RGB thành ảnh xám/sepia	100%
Làm mờ/sắc nét ảnh	100%
Cắt ảnh theo kích thước (cắt ở trung tâm)	100%
Mask ảnh theo khung hình tròn	100%
Mask ảnh theo khung ellipse	100%
Yêu cầu báo cáo	Tự đánh giá
Tự giá mức độ hoàn thành	Đã thực hiện
Ý tưởng thực hiện, mô tả các hàm chức năng	Đã thực hiện
Hình ảnh kết quả	Đã thực hiện
Nhận xét	Đã thực hiện
Tài liệu tham khảo	Đã thực hiện

3 Tổ chức và ý tưởng, mô tả các hàm

Các thư viện được sử dụng trong đồ án bao gồm `numpy`, `PIL`, `matplotlib`. Cụ thể:

- Thư viện `numpy`: sử dụng tùy ý.
- Thư viện `PIL`:
 - + `Image.open()` sử dụng để mở & lấy dữ liệu ảnh.
- Thư viện `matplotlib`:
 - + `pyplot.imsave()` sử dụng để lưu ảnh.
 - + `pyplot.imshow()` sử dụng để hiển thị ảnh kết quả đầu ra.

Các hàm chức năng chính được khai báo lưu trữ bởi mảng danh sách các hàm và tên các hàm cũng được lưu trữ bởi mảng danh sách các tên đầu ra để dễ truy xuất. Các mảng danh sách kể trên bao gồm:

- `func_name[]`: gồm các chuỗi tên tương ứng với chức năng thực được thực thi.
- `function[]`: gồm các hàm chức năng.
- `direction[]`: dành cho chức năng 3 khi đồ án yêu cầu hỏi chiều lật của ảnh.

3.1 Nhóm hàm chức năng

- **change_brightness(img)**

Input: [img: np.ndarray, shape=(height, width, num_channels),
dtype='float']

Output: [res: list(np.ndarray, shape=(height, width, num_channels),
dtype='float')]

Ý tưởng: Màu bị tối là màu có giá trị gần với 0, còn màu bị sáng là màu có giá trị gần với 255.

Vì vậy để tăng độ sáng cần phải cộng từng điểm ảnh của ma trận ảnh với giá trị alpha để kênh màu có giá trị cao hơn, tiến về gần 255.

Ngược lại, để giảm độ sáng cần phải trừ từng điểm ảnh của ma trận ảnh với giá trị alpha để kênh màu có giá trị thấp hơn, tiến về gần 0.

Mô tả: Dùng hàm np.clip() bọc ngoài để chặn trên và chặn dưới cho mỗi điểm ảnh nhằm tránh phép cộng, trừ ma trận cho một giá trị bị lố khỏi vùng dữ liệu cho phép của np.uint8, làm cho hình bị hư/cháy hình.

- **change_contrast(img)**

Input: [img: np.ndarray, shape=(height, width, num_channels),
dtype='float']

Output: [res: list(np.ndarray, shape=(height, width, num_channels),
dtype='float')]

Ý tưởng: Độ tương phản thấy được là do sự chênh lệch sáng tối rõ ràng của 2 điểm ảnh gần nhau. Để điều chỉnh độ tương phản cần phải tăng giảm sự chênh lệch giá trị của 2 điểm ảnh theo tỉ lệ của giá trị alpha.

Vì vậy để tăng độ tương phản cần phải nhân ma trận điểm ảnh với một giá trị alpha>1.

Ngược lại, để giảm độ tương phản cần phải nhân ma trận điểm ảnh với một giá trị alpha<1.

Mô tả: Dùng hàm np.clip() bọc ngoài để chặn trên và chặn dưới cho mỗi điểm ảnh nhằm tránh phép nhân ma trận cho một giá trị bị lố khỏi vùng dữ liệu cho phép của np.uint8, làm cho hình bị hư/cháy hình.

- **flip_image(img, direction=-1)**

Input: [img: np.ndarray, shape=(height, width, num_channels),
dtype='float']

Chiều lật của ảnh, mặc định -1 là cả hai [direction : int]

Output: [res: list(np.ndarray, shape=(height, width, num_channels),
dtype='float')]

Ý tưởng: Lật ảnh là thay đổi vị trí các cột/hàng qua đường thẳng.

+ Để lật dọc ta hoán đổi vị trí theo cột (theo chiều thứ 2).

+ Để lật ngang ta hoán đổi vị trí theo dòng (theo chiều thứ 1).

Mô tả: Áp dụng phương pháp truy xuất indexing và slicing để lật ảnh.

- + Để lật dọc ta trả về `img[:, ::-1]`, nghĩa là truy xuất hết tất cả các hàng theo thứ tự mặc định, truy xuất tất cả các cột với bước nhảy $-1 (<0)$. Tức là từ điểm của cột cuối mỗi hàng truy xuất ngược lên đầu hàng.
- + Để lật ngang ta trả về `img[::-1, :]`, nghĩa là truy xuất hết tất cả các hàng với bước nhảy $-1 (<0)$, truy xuất tất cả các cột theo thứ tự mặc định. Tức là từ điểm của hàng cuối mỗi cột truy xuất ngược lên đầu cột.

• `convert_to_grayscale(img)`

Input: `[img: np.ndarray, shape=(height, width, num_channels),
dtype='float']`

Output: `[res: np.ndarray, shape=(height, width, 1),
dtype='float']`

Ý tưởng: Chuyển từ ảnh màu sang ảnh xám có thể hiểu là chuyển ảnh bao gồm 3 kênh màu thành ảnh bao gồm 1 kênh màu. Cách đơn giản nhất là mỗi pixel sẽ mang giá trị trung bình của 3 kênh màu cũ của chính pixel đó. Tuy nhiên cách làm này mang lại kết quả không được hoàn thiện về mặt thẩm mỹ (do không giữ được độ bóng nhất định của vài điểm trong hình, khiến tổng thể trở nên không rõ ràng). Vì vậy nên ưu tiên chuyển sang cách khác chính là tính màu mới của mỗi pixel bằng cách lấy tổng theo tỉ lệ hệ số giá trị từng kênh màu (công thức theo the weighted method hay còn là luminosity method) như sau:

$$\text{grayscale} = 0.2989R + 0.5870G + 0.1140B$$

Mô tả: Lấy lần lượt các kênh màu Red, Green, Blue nhân với hệ số theo công thức trên. Sau cùng cộng cả ba lại để ra được 1 kênh màu duy nhất đại diện cho ma trận pixels của ảnh xám. Các bước thực hiện trên thể hiện ý tưởng về việc tích tích vô hướng và được thực thi bởi hàm `np.dot()` với 2 tham số truyền vào là: ma trận điểm ảnh với giới hạn chiều thứ 3 là 3 (3 màu RGB), dãy các hệ số lần lượt ứng với các kênh màu đó. Sau cùng dùng hàm `np.clip()` bọc ngoài để tránh bị tràn vùng dữ liệu được cho phép của mỗi điểm ảnh.

• `convert_to_sepia(img)`

Input: `[img: np.ndarray, shape=(height, width, num_channels),
dtype='float']`

Output: `[res: np.ndarray, shape=(height, width, num_channels),
dtype='float']`

Ý tưởng: Tính lại giá trị kênh màu mới dựa trên kênh màu cũ giống cách chuyển về grayscale tuy nhiên ảnh sepia vẫn giữ nguyên 3 kênh màu. Bao gồm các kênh màu đỏ mới, xanh lá mới và xanh biển mới theo công thức:

$$tr = 0.393R + 0.769G + 0.189B$$

$$tg = 0.349R + 0.686G + 0.168B$$

$$tb = 0.272R + 0.534G + 0.131B$$

Mô tả: Dùng hàm `np.clip()` 3 lần với 2 tham số truyền vào là: ma trận điểm ảnh với giới hạn chiều thứ 3 là 3 (3 màu RGB), dãy các hệ số lần lượt ứng với các kênh màu đó để tính 3 kênh màu đỏ mới, xanh lá mới và xanh biển mới.

Tiếp đến khởi tạo ma trận toàn 0 có kích thước giống hệt ma trận ảnh bằng hàm `np.zeros_like()`. Lần lượt gán các kênh màu bằng các kênh màu đã được tính lại ở trên. Sau cùng dùng hàm `np.clip()` cho ma trận điểm ảnh để tránh bị tràn vùng dữ liệu được cho phép của mỗi điểm ảnh.

• grayscale_and_sepia(img)

Input: [img: np.ndarray, shape=(height, width, num_channels),
dtype='float']

Output: [res: list(np.ndarray, shape=(height, width, num_channels),
dtype='float')]]

Ý tưởng: Hàm dùng để trả về ma trận điểm ảnh grayscale và ma trận điểm ảnh sepia.

Mô tả: Trả về bằng cách gọi hàm `convert_to_grayscale` và hàm `convert_to_sepia`.

• blur_and_sharpen(img)

Input: [img: np.ndarray, shape=(height, width, num_channels),
dtype='float']

Output: [res: list(np.ndarray, shape=(height, width, num_channels),
dtype='float')]]

Ý tưởng: Thấy được ảnh rõ ràng là bởi vì sự chênh lệch giữa nét này và nét kia, màu sắc của 2 nét gần nhau là rõ ràng. Nhiệm vụ của việc làm mờ là làm giảm sự chênh lệch đường nét, trung hòa chúng. Cách 1 là lấy trung bình hàng xóm, Cách 2 là hàng xóm thân cận chia nhiều hơn.

Làm rõ hình là đem hình đã làm mờ đi gọi hàm `sobel`. Lưu ý rằng làm sắc nét ảnh grayscale có thể khiến bị hư một số điểm ảnh (noise).

Mô tả: Đầu tiên, khởi tạo một kernel 3x3 với các giá trị được xác định trước theo công thức **Gaussian Blur 3x3** như sau:

$$\text{kernel: } \frac{1}{16} \times \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Tiếp theo, hàm tạo một danh sách các mảng tạm thời rồi thực hiện quá trình làm mờ và làm sắc nét cho từng phần tử trong kernel. Đầu tiên, hàm tạo một bản sao của hình ảnh đầu vào và lăn các pixel của nó theo các chiều dọc và ngang của kernel (từ -1 đến 1) bằng hàm `np.roll()`. Sau đó, hàm nhân mỗi phần tử của mảng tạm thời với giá trị tương ứng của kernel và thêm mảng tạm thời đó vào danh sách mảng tạm thời.

Sau khi tạo danh sách mảng tạm thời, hàm tạo một mảng NumPy từ danh sách này và tính tổng của chúng theo trực đầu tiên (axis=0) để tạo ra ảnh đã được làm mờ và làm sắc nét cuối cùng. Cuối cùng, hàm trả về kết quả là ảnh đã được làm mờ và làm sắc nét.

Đối với phần trả về của phép làm sắc nét, sử dụng công thức thường được sử dụng của **Unsharp Masking** như sau:

sharpened = original + (original - blurred) x amount

Trong đó **amount** là một số dương thể hiện mức độ tác động của phép làm nét. Nếu **amount** càng lớn, thì sự khác biệt giữa ảnh gốc và ảnh được làm mờ càng được tăng lên và độ nét của ảnh sau khi được làm nét càng được cải thiện.

Tuy nhiên chương trình bao hàm trong đồ án không đòi hỏi mức độ xử lý ảnh nâng cao và chuyên sâu nên cài đặt cho hàm xử lý mặc định **amount=1**. Từ đó lấy ma trận ảnh gốc $\times 2$ trừ cho ma trận ảnh đã được làm mờ ở trên sẽ được ma trận ảnh đã được làm sắc nét.

• center_crop(img)

Input: [img: np.ndarray, shape=(height, width, num_channels),
dtype='float']
Output: [res: list(np.ndarray, shape=(height//2, width//2, num_channels),
dtype='float')]

Ý tưởng: Cắt theo trung tâm là cắt sao cho độ rộng các khoảng bỏ của mỗi chiều là bằng nhau. Thực hiện bằng cách slicing.

Mô tả: Cắt ảnh từ hàng $height//4$ đến hàng $height - height//4$ và từ cột $width//4$ đến cột $width - width//4$. Chiều thứ 3 gồm các kênh màu thì vẫn lấy đủ số kênh.

• circle_mask(img)

Input: [img: np.ndarray, shape=(height, width, num_channels),
dtype='float']
Output: [res: list(np.ndarray, shape=(height, width, num_channels),
dtype='float')]

Ý tưởng: Lấy đường tròn áp lên hình, những điểm ảnh nào nằm trên đường tròn thì ta giữ lại, ngược lại thì tắt màu đi. Nói cách khác là dùng phép indexing theo quy luật trong thì giữ, ngoài thì tắt.

Mô tả: Khởi tạo hai trục X, Y dựa trên hình dáng của bức ảnh với trục X kéo từ 0 đến chiều rộng của ảnh, trục Y kéo từ 0 đến chiều rộng của ảnh. So sánh giữa chiều cao (a) và chiều cao (b) của ảnh để chọn kích thước nhỏ hơn rồi chia đôi làm bán kính hình tròn (R). Tạo mask với phương trình tập hợp những điểm bên ngoài đường tròn ($> R^2$) như sau:

$$\text{mask} = (X - \frac{1}{2}a)^2 + (Y - \frac{1}{2}b)^2 > R^2$$

Tiếp đến, tạo ma trận điểm ảnh mới dựa trên ảnh gốc bằng hàm `np.copy()`. Sau cùng masking ma trận vừa tạo để tắt những điểm màu nằm ngoài vòng tròn về 0.

- **ellipse_mask(img)**

Input: [img: np.ndarray, shape=(height, width, num_channels),
 dtype='float']
Output: [res: list(np.ndarray, shape=(height, width, num_channels),
 dtype='float')]

Ý tưởng: Lấy hình ellipse áp lên hình, những điểm ảnh nào nằm trên hình ellipse thì ta giữ lại, ngược lại thì tắt màu đi. Nói cách khác là dùng phép indexing theo quy luật trong thì giữ, ngoài thì tắt.

Mô tả: Khởi tạo hai trục X, Y dựa trên hình dáng của bức ảnh với trục X kéo từ âm nửa chiều rộng đến nửa chiều rộng của ảnh, trục Y kéo từ âm nửa chiều cao đến nửa chiều cao của ảnh để thuận tiện tính toán đối xứng trực. Tâm là điểm (0,0). Tạo mask với phương trình tập hợp những điểm bên ngoài hình ellipse như sau:

$$\text{ellipse1} = \frac{(x \cos(\theta) - y \sin(\theta))^2}{a^2} + \frac{((x \sin(\theta) + y \cos(\theta))^2}{b^2} \leq 1$$

$$\text{ellipse2} = \frac{(x \cos(\theta + \pi/2) + y \sin(\theta + \pi/2))^2}{b^2} + \frac{(-x \sin(\theta + \pi/2) + y \cos(\theta + \pi/2))^2}{a^2} \leq 1$$

Với góc θ là góc lệch giữa đường chéo và cạnh của hình vuông. Tiếp đến, tạo ma trận điểm ảnh mới dựa trên ảnh gốc bằng hàm `np.copy()`. Sau cùng masking ma trận vừa tạo để tắt những điểm màu nằm ngoài vòng tròn về 0.

3.2 Nhóm hàm lấy dữ liệu (từ người dùng, từ ảnh, ...)

- **input_from_user()**

Input: Không có dữ liệu đầu vào

Output: File ảnh cần chỉnh sửa [filename: str]

Số thứ tự hàm chức năng chỉnh sửa mà người dùng cần [func_id: int]

Hướng lật của ảnh nếu người dùng chọn chức năng 3 [direction: str]

Ý tưởng: Số thứ tự của các chức năng lần lượt theo thứ tự như giới thiệu ở đầu đồ án (từ 1 đến 7). Nếu chức năng là 3 thì cho phép hỏi thêm thông tin về chiều lật của ảnh. Ngoài ra nếu lựa chọn là 0 thì mặc định thực hiện hết tất cả chức năng, bao gồm 2 chiều lật của chức năng 3 mà không cần hỏi thêm.

Mô tả: Sử dụng cấu trúc try...except... kết hợp với vòng lặp while True để ràng buộc dữ liệu đọc từ bàn phím vào (input không được để trống, sai định dạng,...) nhằm giảm thiểu lỗi nhập nhầm ký tự.

- **data_from_image(filename)**

Input: Tên file ảnh cần đọc/mở [filename: str]

Output: Ma trận 1D chứa các điểm ảnh

```
[img: np.ndarray, shape=(height, width, num_channels),
 dtype='float')]
```

Ý tưởng: Đọc dữ liệu ma trận điểm ảnh của ảnh gốc và ép về kiểu float cho thuận tiện tính toán. (Ví dụ như khi nhân ma trận với các hệ số là số thực hay kết quả phép tính lớn, kiểu dữ liệu unsigned int sẽ không cho giá trị vượt khoảng mà sẽ lấy giá trị bé nhất cộng vào giá trị dư đó, khiến cho tính toán bị sai)

Mô tả: Chương trình dùng hàm Image.open() để lấy dữ liệu từ ảnh sau đó truyền dữ liệu vừa lấy được vào np.array() để ép kiểu ma trận ảnh về np.ndarray. Đặc biệt rằng kiểu dữ liệu của các pixels được lấy ra mặc định là np.uint8 nên ta phải truyền thêm phép gán dtype='float' để thuận tiện cho các thao tác tính toán về sau.

3.3 Nhóm hàm xuất dữ liệu

- **save_img(img_list, filein, func_id, direction='both')**

Input: Danh sách các ma trận ảnh đã chỉnh sửa

```
[img_list: list(np.ndarray, dtype='float')]
```

Tên file ảnh gốc [filein: str]

Index của hàm chức năng trong danh sách [func_id: int]

Hướng lật của ảnh (chức năng 3) [direction: str]

Output: Không có dữ liệu đầu ra

Ý tưởng: Thực hiện tách chuỗi từ tên gốc để gán được tên mới hợp lệ cho ảnh đầu ra. Sau cùng lưu lại ảnh dưới tên vừa tạo.

Mô tả: Dùng phép rfind(.) cho chuỗi tên gốc để lấy được vị trí xuất hiện sau cùng của dấu . phân tách phần tên và định dạng ảnh. Từ đó gán được tên file ảnh output là từ index 0 đến vị trí vừa tìm được (filein[:index]), định dạng ảnh thì từ vị trí đó đến cuối chuỗi. (filein[index:]) Lặp đến mỗi ảnh và cộng chuỗi để ra được tên ảnh đầu ra, dùng hàm plt.imsave() để lưu ảnh được astype(np.uint8).

3.4 Nhóm hàm điều phối chương trình

- **program(filename, func_id, direction)**

Input: Tên file ảnh gốc [filename: str]
Index của hàm chức năng trong danh sách [func_id: int]
Hướng lật của ảnh (chức năng 3) [direction: str]

Output: Không có dữ liệu đầu ra

Ý tưởng: Điều phối việc lấy dữ liệu ảnh gốc, gọi các hàm chức năng và lưu ảnh.

Mô tả: Lấy dữ liệu ảnh đã chuyển về np.ndarray bằng cách gọi hàm `data_from_img()`. Sau đó gọi các hàm thực hiện chức năng cần và gọi hàm `save_img()` để lưu ảnh.

- **main()**

Input: Không có dữ liệu đầu vào

Output: Không có dữ liệu đầu ra

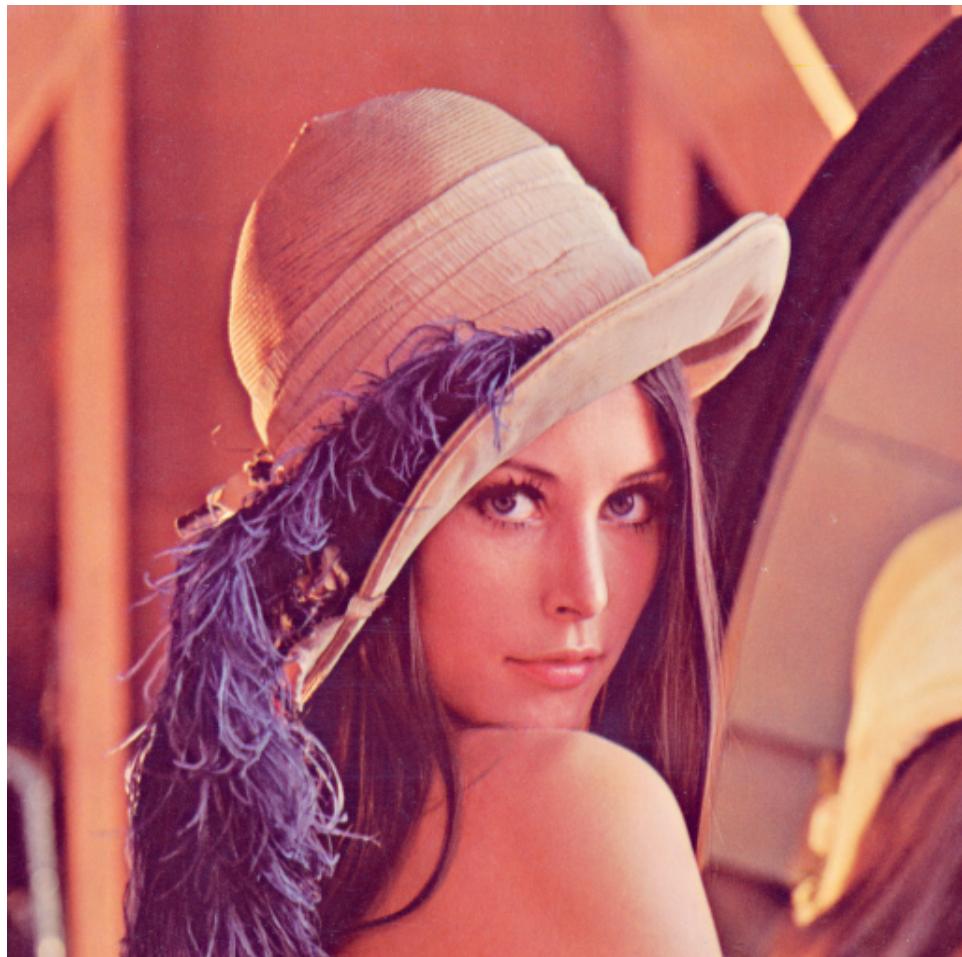
Ý tưởng: Lần lượt gọi thực thi hàm `input_from_user()` rồi truyền những giá trị trả về vào hàm `program()` để nó điều phối những tác vụ còn lại.

4 Kết quả thử nghiệm

Thử nghiệm từng chức năng.

Dùng thư viện `timeit` để đo runtime của các hàm chức năng chính, độ chính xác về thời gian ghi trong báo cáo được làm tròn đến chữ số thập phân thứ 5.

Hình ảnh testcase



Hình 1: Ảnh dùng làm testcase. Nguồn: [Github](#)

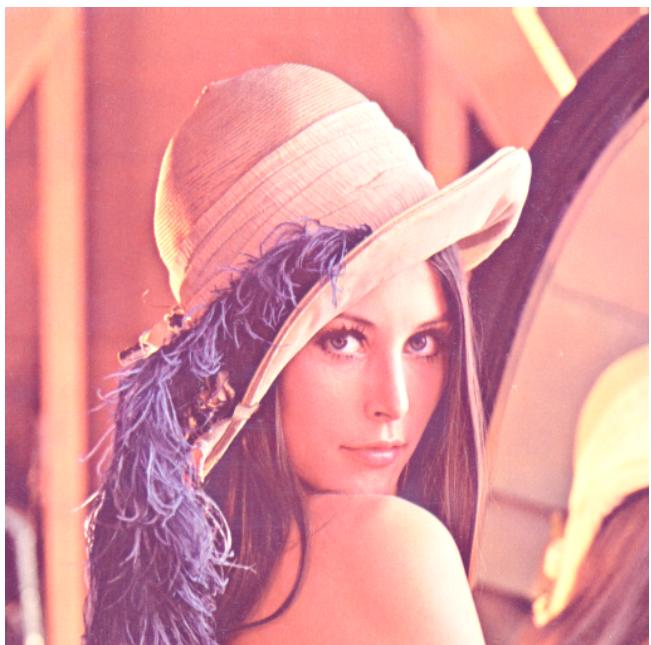
Thông số ảnh dùng làm testcases:

Chiều rộng:	512
Chiều cao:	512
Dimension:	(512×512) pixels
Size:	462 KB

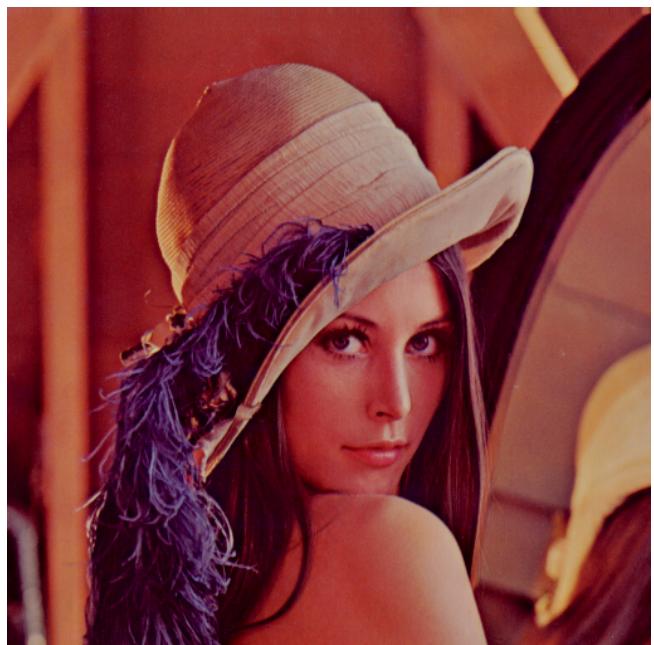
Bảng thời gian kết quả chạy	
change_brightness()	0.01161s
change_contrast()	0.00843s
flip_image()	0.00104s
grayscale_and_sepia()	0.02768s
blur_and_sharpen()	0.15369s
center_crop()	0.00097s
circle_mask()	0.00450s
ellipse_mask()	0.02055s

Hình ảnh kết quả

- Hàm thay đổi độ sáng



(a) Brighter

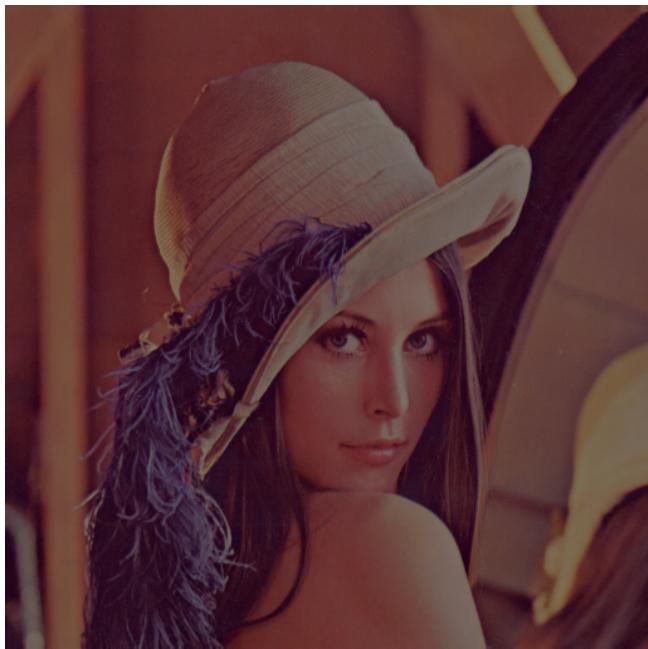


(b) Darker

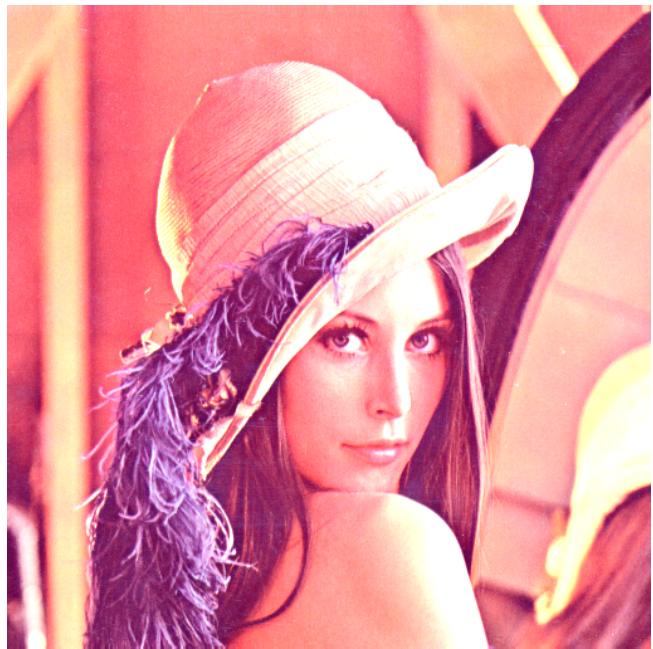
Nhận xét:

- Hàm được chạy với giá trị alpha=50
- Thời gian chạy <15s theo yêu cầu đề án.
- Về kích thước: cả 2 ảnh đều ra vẫn giữ được y nguyên kích thước ban đầu.
- Ảnh kết quả cho thấy được sự thay đổi độ sáng tối so với ảnh gốc.
- Dung lượng của 2 ảnh lần lượt là 520KB và 526KB, đều lớn hơn ảnh gốc.

- Hàm thay đổi độ tương phản



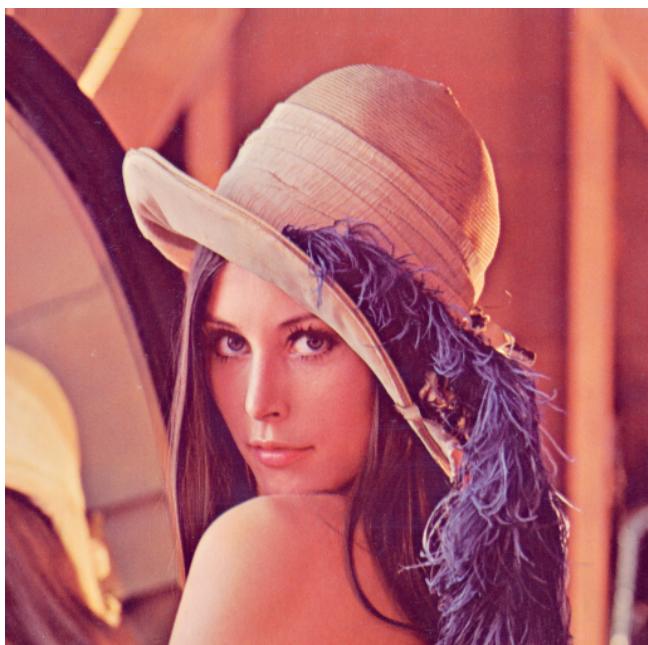
(a) Low contrast



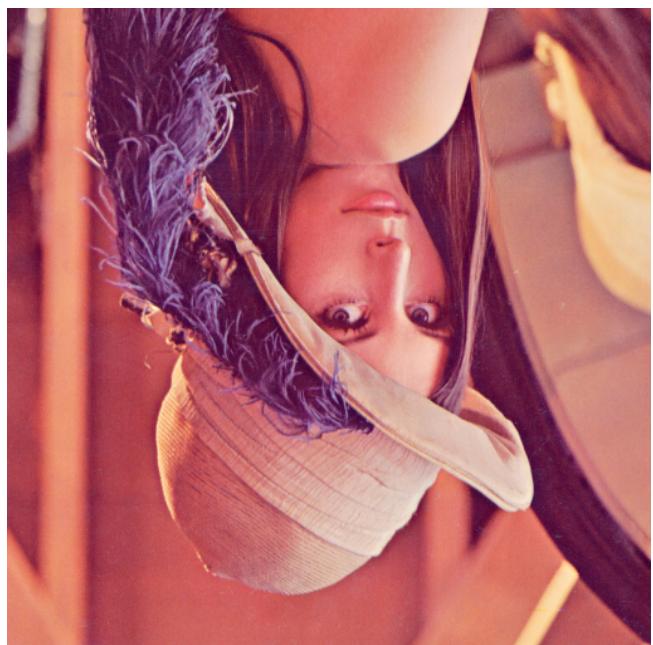
(b) High contrast

Nhận xét: Hàm được chạy với giá trị $\alpha=1,5$ và $\alpha=0,5$. Ảnh kết quả cho thấy được sự thay đổi độ tương phản ánh sáng so với ảnh gốc. Ảnh trái không nhận biết được trong khi ảnh phải cho ta thấy mặt phải cô gái đang hướng về ánh sáng. Dung lượng 2 ảnh lần lượt là 448KB (bé hơn ảnh gốc) và 506KB (lớn hơn ảnh gốc).

- Hàm lật ảnh



(a) Horizontal Flip



(b) Vertical Flip

Nhận xét: Do chỉ sử dụng phép indexing và slicing nên đây là 1 trong 2 hàm chạy nhanh nhất. Dung lượng của 2 ảnh kết quả đều bằng 547KB và đều lớn hơn ảnh gốc.

- Hàm chuyển ảnh xám/sepia



(a) Grayscale



(b) Sepia

Nhận xét: Ảnh xám giữ được độ bóng và rõ giữa các chi tiết. Cả hai ảnh đều không bị hư hay nhiễu. Dung lượng của ảnh xám chỉ chiếm 1 kênh màu nên nhẹ hơn ảnh sepia ($273KB < 411KB$).

- Hàm làm mờ/sắc nét



(a) Blur



(b) Sharpen

Nhận xét: Đây là hàm phức tạp và chạy lâu nhất. Ảnh trái được làm mờ nhẹ và có dung lượng (395KB) nhẹ hơn ảnh rõ nét (627KB), do phải lưu trữ pixels khác biệt càng nhiều càng rõ. Cả hai ảnh đều không bị hư hay nhiễu.

- Hàm cắt ở trung tâm & mask hình tròn



(a) Center Crop



(b) Circle Mask

Nhận xét: Hàm cắt trung tâm chạy nhanh nhất trong các hàm. Dung lượng của ảnh này cũng nhỏ nhất do kích thước đã được giảm đi ($147KB$ cho ảnh (256×256)). Còn ảnh đã được mask hình tròn có dung lượng $447KB$ gần bằng với ảnh có độ tương phản thấp ($448KB$).

- Hàm mask hình ellipse

**Hình 8:** Ellipse Mask

Nhận xét: Ảnh đã được mask hình ellipse có dung lượng $512KB$ (lớn hơn ảnh gốc). Hàm này chạy cũng tương đối nhanh.

5 Nguồn tham khảo

- Tài liệu hướng dẫn Lab và thực hành **Đồ án 2** của môn Toán Ứng dụng & Thống kê.
- Tài liệu thư viện *Numpy*
- Tài liệu thư viện *Pillow*.
- Giới hạn *input* cho người dùng.
- Hàm *numpy.clip()*.
- Numpy *Iterating*.
- Hàm *numpy.dot()*.
- Tham khảo phương pháp làm *mờ ảnh*.
- Tham khảo phương pháp làm *sắc nét ảnh*.
- Tham khảo phương pháp mask *đường tròn*.
- Hàm *rfind()* để xử lý chuỗi.
- Hướng dẫn *lưu ảnh grayscale*.