

ĐẠI HỌC QUỐC GIA
THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



MTH00057 - Toán ứng dụng và thống kê cho Công
nghệ thông tin

BÁO CÁO ĐỒ ÁN 2
Image Processing

Họ tên
Nguyễn Lê Hồ Anh Khoa

MSSV
23127211

Giảng viên hướng dẫn
Nguyễn Văn Quang Huy
Trần Hà Sơn
Nguyễn Đình Thúc
Nguyễn Ngọc Toàn

Ngày 25 tháng 7 năm 2025

Mục lục

1	Thông tin sinh viên	3
2	Dánh giá	3
2.1	Bảng tự đánh giá các yêu cầu đã hoàn thành	3
2.2	Dánh giá tổng thể mức độ hoàn thành của bài nộp	3
3	Ý tưởng thực hiện	3
3.1	Thay đổi độ sáng	3
3.2	Thay đổi độ tương phản	4
3.3	Lật ảnh ngang	4
3.4	Lật ảnh dọc	4
3.5	Chuyển đổi ảnh sang grayscale	4
3.6	Chuyển đổi ảnh sang sepia	4
3.7	Làm mờ ảnh	5
3.8	Làm nét ảnh	6
3.9	Cắt ảnh theo kích thước (cắt ở trung tâm)	6
3.10	Cắt ảnh theo khung hình tròn	7
3.11	Cắt ảnh theo khung hình elip chéo nhau	7
4	Mô tả các hàm	9
4.1	Các thư viện cần thiết	9
4.2	Hàm <code>read_img(img_path)</code>	9
4.3	Hàm <code>show_img(img_2d)</code>	9
4.4	Hàm <code>save_img(img, img_path)</code>	9
4.5	Hàm <code>increase_brightness(img_array, brightness)</code>	9
4.6	Hàm <code>adjust_contrast(img_array, contrast_factor)</code>	10
4.7	Hàm <code>flip_horizontal(img_array)</code>	10
4.8	Hàm <code>flip_vertical(img_array)</code>	10
4.9	Hàm <code>convert_to_grayscale(img_array)</code>	10
4.10	Hàm <code>convert_to_sepia(img_array)</code>	11
4.11	Hàm <code>apply_blur(img_array, kernel_size)</code>	11
4.12	Hàm <code>sharpen_image(img_array)</code>	12
4.13	Hàm <code>crop_center(img_array, crop_height, crop_width)</code>	13
4.14	Hàm <code>crop_circle(img_array)</code>	13
4.15	Hàm <code>crop_ellipse(img_array, ratio)</code>	14
5	Kết quả	15
5.1	Ảnh gốc	16
5.2	Thay đổi độ sáng	16
5.3	Thay đổi độ tương phản	17

5.4	Ảnh lật ngang	18
5.5	Ảnh lật dọc	19
5.6	Chuyển đổi sang ảnh xám	19
5.7	Chuyển đổi sang ảnh sepia	20
5.8	Làm mờ ảnh	20
5.9	Làm sắt nét ảnh	21
5.10	Cắt ảnh theo kích thước trung tâm	22
5.11	Cắt ảnh theo khung hình tròn	22
5.12	Cắt ảnh theo khung 2 hình elip chéo nhau	23
5.13	Thời gian chạy	23

6	Nhận xét	23
----------	-----------------	-----------

1 Thông tin sinh viên

Họ và tên: Nguyễn Lê Hồ Anh Khoa. MSSV: 23127211. Lớp: 23CLC09

2 Đánh giá

2.1 Bảng tự đánh giá các yêu cầu đã hoàn thành

Bảng 1: Bảng tự đánh giá đồ án

STT	Yêu cầu	Mức độ hoàn thành
1	Thay đổi độ sáng	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	Viết hàm main xử lý	100%
9	Cắt ảnh theo khung 2 hình elip chéo nhau (Nâng cao)	100%
	Tổng cộng	100%

2.2 Đánh giá tổng thể mức độ hoàn thành của bài nộp

Bài nộp đã hoàn thành đầy đủ các yêu cầu đề ra trong bài tập. Tất cả các yêu cầu đều đã được cài đặt và kiểm thử thành công. Tổng thể, bài nộp đã hoàn thành 100% các yêu cầu đề ra.

3 Ý tưởng thực hiện

3.1 Thay đổi độ sáng

Ý tưởng thực hiện hàm thay đổi độ sáng là cộng hoặc trừ một giá trị bù vào từng pixel của ảnh để tăng hoặc giảm độ sáng. Giá trị sau khi thay đổi được giới hạn trong khoảng hợp lệ để đảm bảo tính chính xác.

3.2 Thay đổi độ tương phản

Ý tưởng của hàm điều chỉnh độ tương phản là sử dụng giá trị trung bình của toàn bộ các pixel trong ảnh làm điểm tham chiếu. Dựa trên giá trị này, các pixel được điều chỉnh để tăng hoặc giảm khoảng cách của chúng so với giá trị trung bình. Hệ số tương phản được sử dụng để kiểm soát mức độ thay đổi: giá trị lớn hơn 1 làm tăng độ tương phản, trong khi giá trị nhỏ hơn 1 làm giảm độ tương phản.

Các giá trị pixel sau khi điều chỉnh được giới hạn trong phạm vi từ 0 đến 255 để đảm bảo không vượt quá giới hạn hợp lệ. Phương pháp này giúp phân bổ thay đổi một cách đồng đều giữa các vùng sáng và tối, giữ được chi tiết của ảnh và đảm bảo sự cân bằng màu sắc sau khi xử lý.

3.3 Lật ảnh ngang

Ý tưởng của hàm lật ảnh ngang là đảo ngược thứ tự các pixel theo chiều ngang (trái sang phải). Điều này được thực hiện bằng cách thay đổi thứ tự cột của ma trận ảnh, trong khi giữ nguyên thứ tự hàng.

3.4 Lật ảnh dọc

Ý tưởng của hàm lật ảnh dọc là đảo ngược thứ tự các pixel theo chiều dọc (trên xuống dưới). Điều này được thực hiện bằng cách thay đổi thứ tự hàng của ma trận ảnh, trong khi giữ nguyên thứ tự cột.

3.5 Chuyển đổi ảnh sang grayscale

Ý tưởng của hàm chuyển đổi ảnh sang grayscale là sử dụng công thức chuẩn được tham chiếu từ tài liệu OpenCV [1]. Công thức này tính toán giá trị độ sáng của từng pixel dựa trên các kênh màu RGB với các trọng số. Kết quả là một ảnh grayscale, trong đó mỗi pixel có cùng giá trị độ sáng trên cả ba kênh màu.

$$Gray = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \quad (1)$$

Hệ số trong công thức chuyển đổi ảnh sang grayscale được xác định dựa trên cách mắt người cảm nhận độ sáng của các màu sắc. Kênh xanh lá (0.587) có hệ số lớn nhất vì mắt nhạy cảm nhất với màu này, kênh đỏ (0.299) có hệ số trung bình, và kênh xanh dương (0.114) có hệ số nhỏ nhất do mắt ít nhạy cảm với màu xanh dương hơn. Các hệ số này đảm bảo ảnh grayscale phản ánh chính xác độ sáng và giữ được chi tiết của ảnh gốc. Kết quả là một ảnh grayscale, phản ánh chính xác độ sáng tổng thể của ảnh gốc và giữ được các chi tiết quan trọng.

3.6 Chuyển đổi ảnh sang sepia

Ý tưởng của hàm chuyển đổi ảnh sang sepia là áp dụng một ma trận chuyển đổi màu sắc lên từng pixel của ảnh. Ma trận này được thiết kế để tạo ra hiệu ứng màu sắc đặc

trung của ảnh sepia, với tông màu nâu ám và giảm độ bão hòa của các màu sắc. Công thức chuyển đổi được tham khảo từ tài liệu [2] và được áp dụng như sau:

$$\begin{aligned} Sepia_R &= 0.393 \cdot R + 0.769 \cdot G + 0.189 \cdot B \\ Sepia_G &= 0.349 \cdot R + 0.686 \cdot G + 0.168 \cdot B \\ Sepia_B &= 0.272 \cdot R + 0.534 \cdot G + 0.131 \cdot B \end{aligned} \quad (2)$$

Các giá trị sau khi chuyển đổi được giới hạn trong phạm vi từ 0 đến 255 để đảm bảo không vượt quá giới hạn hợp lệ. Kết quả là một ảnh sepia với tông màu cổ điển, mang lại cảm giác ấm áp và hoài cổ.

3.7 Làm mờ ảnh

Ý tưởng của hàm làm mờ ảnh là áp dụng một bộ lọc trung bình (mean filter) lên từng pixel của ảnh. Bộ lọc này sử dụng một ma trận kernel có kích thước xác định (ví dụ: 5x5) để tính giá trị trung bình của các pixel xung quanh. Công thức và phương pháp được tham khảo từ tài liệu [3].

Quá trình thực hiện bao gồm:

- Tạo một kernel với các giá trị bằng nhau (box blur), sao cho tổng các phần tử trong kernel bằng 1.
- Thêm padding vào ảnh gốc để đảm bảo rằng các pixel ở biên vẫn có đủ vùng lân cận để áp dụng kernel. Padding được thực hiện bằng cách phản chiếu các giá trị pixel ở biên.
- Mỗi pixel trong ảnh được thay thế bằng giá trị trung bình của các pixel trong vùng lân cận, được xác định bởi kích thước kernel.
- Các giá trị sau khi tính toán được giới hạn trong phạm vi từ 0 đến 255 để đảm bảo không vượt quá giới hạn hợp lệ.

Lý do chọn box blur:

- Box blur đơn giản và hiệu quả trong việc làm mờ ảnh, giúp làm mềm các chi tiết nhỏ mà không làm mất cấu trúc tổng thể của ảnh.
- Tính toán dễ dàng và nhanh chóng, phù hợp với các ứng dụng xử lý ảnh cơ bản.
- Box blur tạo ra hiệu ứng làm mịn tổng thể, giúp giảm nhiễu và làm mềm các cạnh sắc nét trong ảnh.

Kết quả là một ảnh mờ, trong đó các chi tiết nhỏ được làm mờ, mang lại hiệu ứng làm mịn tổng thể. Kích thước kernel tăng lên, ảnh sẽ trở nên mờ hơn vì giá trị trung bình được tính trên một vùng lớn hơn, làm giảm độ chi tiết của ảnh. Tuy nhiên, điều này cũng làm tăng thời gian xử lý vì số lượng phép tính cần thực hiện cho mỗi pixel tăng lên theo kích thước của kernel.

3.8 Làm nét ảnh

Ý tưởng của hàm làm nét ảnh là áp dụng một bộ lọc làm nét (sharpening filter) lên từng pixel của ảnh. Bộ lọc này sử dụng một ma trận kernel đặc biệt được tham khảo từ tài liệu [3] để làm nổi bật các cạnh và chi tiết trong ảnh.

Quá trình thực hiện bao gồm:

- Sử dụng một kernel làm nét, ví dụ:

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

Kernel này tăng cường giá trị của pixel trung tâm trong khi giảm giá trị của các pixel xung quanh, giúp làm nổi bật các cạnh và chi tiết.

- Thêm padding vào ảnh gốc để đảm bảo rằng các pixel ở biên vẫn có đủ vùng lân cận để áp dụng kernel. Padding được thực hiện bằng cách thêm các giá trị 0 xung quanh ảnh gốc.
- Mỗi pixel trong ảnh được thay thế bằng giá trị tính toán từ ma trận kernel áp dụng lên vùng lân cận của pixel đó.
- Các giá trị sau khi tính toán được giới hạn trong phạm vi từ 0 đến 255 để đảm bảo không vượt quá giới hạn hợp lệ.

Kết quả là một ảnh sắc nét hơn, trong đó các cạnh và chi tiết được làm nổi bật, mang lại hiệu ứng tăng cường độ rõ nét cho ảnh.

Có thể tăng kích thước kernel khi làm nét ảnh, nhưng điều này thường không mang lại hiệu quả tốt. Kernel làm nét thường được thiết kế nhỏ (ví dụ: 3x3) để tập trung vào các chi tiết nhỏ và các cạnh trong ảnh. Nếu tăng kích thước kernel, vùng ảnh bị ảnh hưởng sẽ lớn hơn, dẫn đến việc làm nét không còn tập trung vào các chi tiết nhỏ mà có thể làm biến dạng hoặc mất cân bằng các vùng lớn trong ảnh.

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

Ý tưởng của hàm cắt ảnh theo kích thước là lấy một phần ảnh có kích thước xác định, được cắt từ trung tâm của ảnh gốc.

Quá trình thực hiện bao gồm:

- Xác định tọa độ bắt đầu và kết thúc của vùng cắt dựa trên kích thước ảnh gốc và kích thước vùng cắt được yêu cầu.
- Tọa độ bắt đầu được tính bằng cách lấy khoảng cách từ mép ảnh đến trung tâm, sau đó trừ đi một nửa kích thước vùng cắt.
- Vùng ảnh nằm giữa tọa độ bắt đầu và kết thúc được trích xuất để tạo thành ảnh mới.

Kết quả là một ảnh được cắt từ trung tâm, với kích thước chính xác theo yêu cầu, đảm bảo giữ được các chi tiết quan trọng nhất

3.10 Cắt ảnh theo khung hình tròn

Ý tưởng của hàm cắt ảnh theo khung hình tròn là tạo một mặt nạ hình tròn được nối tiếp trong ảnh gốc, sau đó giữ lại các pixel nằm trong vùng hình tròn và tô đen các pixel bên ngoài.

Quá trình thực hiện bao gồm:

- Xác định bán kính của hình tròn dựa trên kích thước nhỏ hơn giữa chiều cao và chiều rộng của ảnh (nếu ảnh là hình chữ nhật).
- Tính tọa độ tâm của hình tròn, thường là trung tâm của ảnh gốc.
- Tạo mặt nạ hình tròn bằng công thức:

$$\text{Mask} = (x - \text{center}_x)^2 + (y - \text{center}_y)^2 \leq \text{radius}^2$$

Trong đó:

- x, y : Tọa độ pixel trong ảnh.
- $\text{center}_x, \text{center}_y$: Tọa độ tâm của hình tròn.
- radius: Bán kính của hình tròn.
- Áp dụng mặt nạ lên ảnh gốc, giữ lại các pixel nằm trong hình tròn và tô đen các pixel bên ngoài bằng cách đặt giá trị của chúng về 0.

Kết quả là một ảnh được cắt theo khung hình tròn, giữ lại các chi tiết nằm trong vùng hình tròn và loại bỏ các chi tiết bên ngoài.

3.11 Cắt ảnh theo khung hình elip chéo nhau

Ý tưởng của hàm cắt ảnh theo khung hình elip chéo nhau là tạo hai mặt nạ hình elip, được xoay chéo nhau 90 độ, và giữ lại các pixel nằm trong ít nhất một trong hai hình elip.

Quá trình thực hiện bao gồm:

- Xác định kích thước của hai hình elip dựa trên tỷ lệ kích thước (ratio) được cung cấp. Tỷ lệ này kiểm soát độ dài của các trục chính và phụ của hình elip so với kích thước ảnh gốc.
- Tính tọa độ tâm của hình elip, thường là trung tâm của ảnh gốc.
- Tạo hai mặt nạ hình elip bằng cách sử dụng công thức:

$$\text{Mask}_1 = \frac{(x + y - \text{center}_x - \text{center}_y)^2}{\text{ratio} \cdot h^2} + \frac{(x - y - \text{center}_x + \text{center}_y)^2}{(1 - \text{ratio}) \cdot h^2} \leq 1$$

$$\text{Mask}_2 = \frac{(x + y - \text{center}_x - \text{center}_y)^2}{(1 - \text{ratio}) \cdot h^2} + \frac{(x - y - \text{center}_x + \text{center}_y)^2}{\text{ratio} \cdot h^2} \leq 1$$

Trong đó:

- x, y : Tọa độ pixel một điểm trên elip.
 - $\text{center}_x, \text{center}_y$: Tọa độ tâm của hình elip.
 - h : Chiều cao của ảnh.
 - ratio: Tỷ lệ kiểm soát độ dài trục chính và phụ của hình elip (giá trị mặc định là 0.75).
- Kết hợp hai mặt nạ bằng phép toán logic OR ($\mid\mid$) để giữ lại các pixel nằm trong ít nhất một trong hai hình elip.
 - Áp dụng mặt nạ lên ảnh gốc, giữ lại các pixel nằm trong vùng hình elip và loại bỏ các pixel bên ngoài bằng cách đặt giá trị của chúng về 0.

Giá trị của ratio:

- Có vô số hình elip thỏa điều kiện nằm trong và tiếp xúc với hình vuông. Vậy nên ta có tỷ lệ ratio nằm trong khoảng $0 < \text{ratio} < 1$ để kiểm soát tỷ lệ giữa các bán trục của hình elip:

- Nếu ratio gần 0.5, hai hình elip sẽ có xu hướng trở thành hình tròn nội tiếp hình vuông. Điều này xảy ra vì khi ratio tiến gần đến 0.5, các bán trục chính và phụ của hình elip trở nên xấp xỉ nhau. Cụ thể:

$$\text{Mask}_1 = \frac{(x + y - \text{center}_x - \text{center}_y)^2}{0.5 \cdot h^2} + \frac{(x - y - \text{center}_x + \text{center}_y)^2}{0.5 \cdot h^2} \leq 1$$

$$\text{Mask}_2 = \frac{(x + y - \text{center}_x - \text{center}_y)^2}{0.5 \cdot h^2} + \frac{(x - y - \text{center}_x + \text{center}_y)^2}{0.5 \cdot h^2} \leq 1$$

Khi các hệ số trong công thức trở nên xấp xỉ nhau, cả hai hình elip sẽ biến thành hai hình tròn nội tiếp hình vuông. Kết quả là vùng cắt sẽ trở thành một khung hình tròn duy nhất.

- Nếu ratio càng xa 0.5, hai hình elip sẽ trở nên càng hẹp, nổi bật hình dạng elip của chúng.
- Nếu ratio vượt ra ngoài khoảng $0 < \text{ratio} < 1$:

- ratio nhỏ hơn hoặc bằng 0 không hợp lệ, vì không thể tạo hình elip.
- ratio lớn hơn 1 sẽ làm hình elip bị biến dạng, không còn phù hợp với mục đích cắt ảnh.

Kết quả là một ảnh được cắt theo khung hình elip chéo nhau, tạo hiệu ứng trang trí độc đáo hoặc làm nổi bật vùng trung tâm

4 Mô tả các hàm

4.1 Các thư viện cần thiết

Trong đồ án này, các thư viện chính được sử dụng bao gồm:

- **numpy**: Thư viện xử lý ma trận và mảng số học hiệu năng cao, cung cấp các phép tính vector hóa giúp tăng tốc độ tính toán trên dữ liệu lớn.
- **PIL (Python Imaging Library)**: Thư viện xử lý ảnh, được sử dụng để đọc/ghi các định dạng ảnh phổ biến và thực hiện các thao tác cơ bản với ảnh.
- **matplotlib**: Thư viện dùng để hiển thị ảnh.
- Các thư viện bổ trợ:
 - **time**: Đo thời gian thực thi

4.2 Hàm `read_img(img_path)`

Hàm `read_img` có nhiệm vụ đọc hình ảnh từ đường dẫn được truyền vào, sau đó chuyển đổi thành một mảng NumPy 3 chiều đại diện cho ảnh RGB (Nếu ảnh grayscale, nó sẽ được chuyển đổi thành 3 kênh với cùng giá trị). Điều này cho phép máy tính xử lý hình ảnh dễ dàng hơn.

4.3 Hàm `show_img(img_2d)`

Hàm `show_img` có chức năng hiển thị ảnh từ mảng numpy 2D sử dụng thư viện `matplotlib`. Để giúp hình ảnh hiển thị rõ ràng và không bị che bởi các trục tọa độ, lệnh `plt.axis('off')` được sử dụng để tắt hiển thị trục.

4.4 Hàm `save_img(img, img_path)`

Hàm `save_img` có nhiệm vụ lưu hình ảnh đã xử lý vào đường dẫn được chỉ định. Mảng chứa các pixel ảnh được chuyển thành ảnh sử dụng hàm `Image.fromarray()` của thư viện PIL. Hình ảnh được lưu dưới dạng file ảnh, giúp người dùng lưu trữ kết quả xử lý.

4.5 Hàm `increase_brightness(img_array, brightness)`

Hàm `increase_brightness` có nhiệm vụ điều chỉnh độ sáng của hình ảnh. Quá trình thực hiện bao gồm các bước sau:

1. **Điều chỉnh độ sáng**: Giá trị độ sáng `brightness` được cộng vào từng phần tử trong mảng hình ảnh. Để đảm bảo không xảy ra lỗi tràn số khi cộng giá trị, mảng ảnh được chuyển sang kiểu dữ liệu `np.int16` trước khi thực hiện phép toán.
2. **Cắt giá trị pixel**: Sau khi điều chỉnh độ sáng, các giá trị pixel có thể vượt quá phạm vi hợp lệ (0 đến 255). Hàm `np.clip()` được sử dụng để giới hạn giá trị pixel trong khoảng từ 0 đến 255, ngăn chặn các giá trị không hợp lệ.

- Chuyển đổi lại về kiểu dữ liệu phù hợp: Cuối cùng, mảng đã được điều chỉnh được chuyển đổi về kiểu np.uint8 để đảm bảo không có lỗi xảy ra.

4.6 Hàm adjust_contrast(img_array, contrast_factor)

Hàm `adjust_contrast` có nhiệm vụ điều chỉnh độ tương phản của hình ảnh. Với `contrast_factor > 1` thì tăng độ tương phản, `< 1` thì giảm độ tương phản. Quá trình thực hiện bao gồm các bước sau:

- Kiểm tra factor truyền vào:** Nếu `contrast_factor` bằng 0 hoặc bằng 1, hàm sẽ trả về mảng ảnh gốc mà không thay đổi gì. Nếu `contrast_factor` nhỏ hơn 0, hàm sẽ tự động điều chỉnh về giá trị tuyệt đối của nó.
- Tính giá trị trung bình:** Giá trị trung bình của các pixel trong mảng hình ảnh được tính bằng hàm `np.mean()` để làm điểm tham chiếu cho việc điều chỉnh độ tương phản.
- Điều chỉnh độ tương phản:** Các pixel trong mảng hình ảnh được điều chỉnh bằng công thức:

$$\text{adjusted} = (\text{img_array} - \text{mean}) \times \text{contrast_factor} + \text{mean}$$

Điều này giúp tăng độ tương phản nếu `contrast_factor > 1` và giảm độ tương phản nếu `contrast_factor < 1`.

- Cắt giá trị pixel:** Sau khi điều chỉnh độ tương phản, các giá trị pixel có thể vượt quá phạm vi hợp lệ (0 đến 255). Hàm `np.clip()` được sử dụng để giới hạn giá trị pixel trong khoảng từ 0 đến 255.
- Chuyển đổi lại về kiểu dữ liệu phù hợp:** Cuối cùng, mảng đã được điều chỉnh được chuyển đổi về kiểu `np.uint8` để đảm bảo không có lỗi xảy ra.

4.7 Hàm flip_horizontal(img_array)

Hàm `flip_horizontal` có nhiệm vụ lật hình ảnh theo chiều ngang. Quá trình thực hiện bao gồm các bước sau:

- Lật ảnh ngang:** Mảng hình ảnh được đảo ngược thứ tự các cột bằng slicing của NumPy: `img_array[:, ::-1, :]`.

4.8 Hàm flip_vertical(img_array)

Hàm `flip_vertical` có nhiệm vụ lật hình ảnh theo chiều dọc. Quá trình thực hiện bao gồm các bước sau:

- Lật ảnh dọc:** Mảng hình ảnh được đảo ngược thứ tự các hàng bằng slicing của NumPy: `img_array[::-1, :, :]`.

4.9 Hàm convert_to_grayscale(img_array)

Hàm `convert_to_grayscale` có nhiệm vụ chuyển đổi hình ảnh RGB thành ảnh xám. Quá trình thực hiện bao gồm các bước sau:

1. **Tính giá trị xám:** Giá trị xám của từng pixel được tính bằng công thức:

$$\text{Gray} = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

2. **Tạo ảnh xám:** Mảng ảnh xám được tạo bằng cách nhân giá trị xám với mặt nạ và xếp chồng các kênh màu.
3. **Chuyển đổi kiểu dữ liệu:** Mảng ảnh xám được chuyển đổi về kiểu `np.uint8`, do các hệ số cộng lại bằng 1 nên không cần hàm `np.clip()` để giới hạn giá trị pixel.

4.10 Hàm convert_to_sepia(img_array)

Hàm `convert_to_sepia` có nhiệm vụ áp dụng hiệu ứng màu sepia lên hình ảnh. Quá trình thực hiện bao gồm các bước sau:

1. **Tính giá trị sepia:** Giá trị sepia của từng pixel được tính bằng công thức ma trận sepia:

$$\text{Sepia_R} = 0.393 \times R + 0.769 \times G + 0.189 \times B$$

$$\text{Sepia_G} = 0.349 \times R + 0.686 \times G + 0.168 \times B$$

$$\text{Sepia_B} = 0.272 \times R + 0.534 \times G + 0.131 \times B$$

2. **Tạo ảnh sepia:** Mảng ảnh sepia được tạo bằng cách xếp chồng các kênh màu sepia.
3. **Cắt giá trị pixel:** Các giá trị pixel được giới hạn trong khoảng từ 0 đến 255 bằng hàm `np.clip()`.
4. **Chuyển đổi kiểu dữ liệu:** Mảng ảnh sepia được chuyển đổi về kiểu `np.uint8`.

4.11 Hàm apply_blur(img_array, kernel_size)

Hàm `apply_blur` có nhiệm vụ làm mờ hình ảnh bằng cách áp dụng bộ lọc trung bình (mean filter). Quá trình thực hiện bao gồm các bước sau:

1. **Tạo kernel làm mờ:** Kernel được tạo bằng hàm `np.ones()` với kích thước `kernel_size` và được chuẩn hóa bằng cách chia tổng các phần tử trong kernel.
2. **Thêm padding vào ảnh:** Để đảm bảo rằng các pixel ở biên vẫn có đủ vùng lân cận để áp dụng kernel, ảnh được thêm padding bằng hàm `np.pad()` với chế độ `reflect`. Chế độ này làm cho giá trị padding được điền bằng cách phản chiếu các giá trị biên của mảng.
3. **Khởi tạo mảng kết quả:** Một mảng mới `blurred_img` có cùng kích thước với ảnh gốc được khởi tạo để lưu trữ kết quả làm mờ. Mảng này được khởi tạo với kiểu dữ liệu `np.float32` để đảm bảo độ chính xác trong quá trình tính toán.
4. **Áp dụng kernel lên từng pixel:** Quá trình áp dụng kernel để làm mờ ảnh được thực hiện bằng cách tính giá trị trung bình của vùng lân cận từng pixel. Các bước chi tiết như sau:

- (a) **Duyệt qua từng kênh màu:** Vòng lặp đầu tiên duyệt qua từng kênh màu (R, G, B) của ảnh gốc.
 - (b) **Duyệt qua từng pixel:** Vòng lặp tiếp theo duyệt qua từng pixel trong ảnh. Với mỗi pixel tại vị trí (i, j) , vùng lân cận của pixel đó được xác định bằng cách lấy một phần của ảnh đã được thêm padding. Vùng lân cận này có kích thước bằng kích thước kernel.
 - (c) **Tính giá trị trung bình:** Giá trị trung bình của vùng lân cận được tính bằng cách nhân từng phần tử trong vùng lân cận với kernel và sau đó tính tổng bằng hàm `np.sum()`. Kết quả được lưu vào vị trí tương ứng trong mảng `blurred_img`.
 - (d) **Lặp lại cho tất cả các pixel:** Quá trình trên được lặp lại cho tất cả các pixel trong ảnh và tất cả các kênh màu.
 - (e) **Giới hạn giá trị pixel:** Sau khi tính toán, các giá trị trong mảng `blurred_img` được giới hạn trong khoảng từ 0 đến 255 bằng hàm `np.clip()` để đảm bảo giá trị hợp lệ.
5. **Chuyển đổi kiểu dữ liệu:** Cuối cùng, mảng kết quả được chuyển đổi về kiểu `np.uint8` để phù hợp với định dạng ảnh.

4.12 Hàm sharpen_image(img_array)

Hàm `sharpen_image` có nhiệm vụ làm nét hình ảnh bằng cách áp dụng bộ lọc làm nét (sharpening filter). Quá trình thực hiện bao gồm các bước sau:

1. **Kiểm tra kích thước vùng cắt:** Nếu kích thước vùng cắt to hơn kích thước ảnh gốc, hàm sẽ trả về ảnh gốc mà không thay đổi gì.
2. **Tạo kernel làm nét:** Kernel làm nét được định nghĩa trước, ví dụ:

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

3. **Thêm padding vào ảnh:** Để đảm bảo rằng các pixel ở biên vẫn có đủ vùng lân cận để áp dụng kernel, ảnh được thêm padding bằng hàm `np.pad()` với chế độ `constant`. Chế độ này làm cho giá trị padding được điền bằng 0.
4. **Khởi tạo mảng kết quả:** Một mảng mới `sharpened` có cùng kích thước với ảnh gốc được khởi tạo để lưu trữ kết quả làm nét. Mảng này được khởi tạo với kiểu dữ liệu `float` để đảm bảo độ chính xác trong quá trình tính toán.
5. **Áp dụng kernel lên từng pixel:** Quá trình áp dụng kernel để làm nét ảnh được thực hiện bằng cách tính giá trị mới của từng pixel dựa trên vùng lân cận của nó. Các bước chi tiết như sau:
 - (a) **Duyệt qua từng pixel:** Vòng lặp đầu tiên duyệt qua từng pixel trong ảnh. Với mỗi pixel tại vị trí (y, x) , vùng lân cận của pixel đó được xác định bằng cách lấy một phần của ảnh đã được thêm padding. Vùng lân cận này có kích thước bằng kích thước kernel (3×3 trong trường hợp này).

- (b) **Duyệt qua từng kênh màu:** Vòng lặp tiếp theo duyệt qua từng kênh màu (R, G, B) của ảnh gốc. Với mỗi kênh màu, giá trị mới của pixel được tính bằng cách nhân từng phần tử trong vùng lân cận với kernel và sau đó tính tổng bằng hàm `np.sum()`.
 - (c) **Lưu kết quả:** Giá trị mới của pixel sau khi tính toán được lưu vào vị trí tương ứng trong mảng `sharpened`.
 - (d) **Lặp lại cho tất cả các pixel:** Quá trình trên được lặp lại cho tất cả các pixel trong ảnh và tất cả các kênh màu.
6. **Giới hạn giá trị pixel:** Sau khi tính toán, các giá trị trong mảng `sharpened` được giới hạn trong khoảng từ 0 đến 255 bằng hàm `np.clip()` để đảm bảo giá trị hợp lệ.
 7. **Chuyển đổi kiểu dữ liệu:** Cuối cùng, mảng kết quả được chuyển đổi về kiểu `np.uint8` để phù hợp với định dạng ảnh.

4.13 Hàm `crop_center(img_array, crop_height, crop_width)`

Hàm `crop_center` có nhiệm vụ cắt hình ảnh theo kích thước được chỉ định, với vùng cắt nằm ở trung tâm của ảnh gốc. Nếu cắt hình vuông thì chiều cao và chiều rộng của vùng cắt sẽ bằng nhau. Quá trình thực hiện bao gồm các bước sau:

1. **Tính toán tọa độ vùng cắt:** Tọa độ bắt đầu và kết thúc của vùng cắt được tính dựa trên kích thước ảnh gốc (`h` và `w`) và kích thước vùng cắt (`crop_height` và `crop_width`). Công thức cụ thể như sau:

$$\text{start_y} = \frac{h - \text{crop_height}}{2}$$

$$\text{start_x} = \frac{w - \text{crop_width}}{2}$$

$$\text{end_y} = \text{start_y} + \text{crop_height}$$

$$\text{end_x} = \text{start_x} + \text{crop_width}$$

Trong đó: - `start_y` và `start_x` là tọa độ bắt đầu của vùng cắt. - `end_y` và `end_x` là tọa độ kết thúc của vùng cắt.

2. **Cắt ảnh:** Vùng ảnh nằm giữa tọa độ bắt đầu (`start_y, start_x`) và kết thúc (`end_y, end_x`) được trích xuất để tạo thành ảnh mới. Quá trình này được thực hiện bằng slicing của NumPy:

```
cropped_img = img_array[start_y : end_y, start_x : end_x, :]
```

4.14 Hàm `crop_circle(img_array)`

Hàm `crop_circle` có nhiệm vụ cắt hình ảnh theo khung hình tròn nội tiếp trong ảnh gốc. Quá trình thực hiện bao gồm các bước sau:

1. Tính toán bán kính và tâm hình tròn:

- Bán kính của hình tròn được tính bằng kích thước nhỏ hơn giữa chiều cao (h) và chiều rộng (w) của ảnh, chia đôi:

$$\text{radius} = \frac{\min(h, w)}{2}$$

- Tâm của hình tròn được xác định là trung tâm của ảnh gốc:

$$\text{center_y} = \frac{h}{2}, \quad \text{center_x} = \frac{w}{2}$$

2. Tạo mặt nạ hình tròn:

- Mặt nạ hình tròn được tạo bằng công thức Euclid, xác định các pixel nằm trong hình tròn:

$$\text{Mask} = (x - \text{center_x})^2 + (y - \text{center_y})^2 \leq \text{radius}^2$$

- Trong đó:

- x và y là các mảng tọa độ được tạo bằng hàm `np.ogrid[:h, :w]`.
 - Mask là một mảng boolean, với giá trị `True` cho các pixel nằm trong hình tròn và `False` cho các pixel bên ngoài.

3. Áp dụng mặt nạ lên ảnh:

- Một mảng mới `circular_img` có cùng kích thước với ảnh gốc được khởi tạo để lưu trữ kết quả.
- Vòng lặp qua từng kênh màu (R, G, B) của ảnh gốc:

$$\text{circular_img}[:, :, c] = \text{img_array}[:, :, c] \times \text{Mask}$$

- Các pixel nằm ngoài hình tròn (giá trị `False` trong `Mask`) được đặt về 0, giữ lại chỉ các pixel nằm trong hình tròn.

4.15 Hàm `crop_ellipse(img_array, ratio)`

Hàm `crop_ellipse` có nhiệm vụ cắt hình ảnh theo khung hình elip chéo nhau. Quá trình thực hiện bao gồm các bước sau:

1. Tính toán tỷ lệ và tâm hình elip:

- Tỷ lệ `ratio` kiểm soát độ dài trục chính và phụ của hình elip. Giá trị `ratio` nằm trong khoảng từ 0.5 đến 1.
- Tâm của hình elip được xác định là trung tâm của ảnh gốc:

$$\text{center_y} = \frac{h}{2}, \quad \text{center_x} = \frac{w}{2}$$

2. Tạo hai mặt nạ hình elip:

- Hai mặt nạ hình elip được tạo bằng công thức toán học:

$$\text{Mask}_1 = \frac{(x + y - \text{center}_x - \text{center}_y)^2}{\text{ratio} \cdot h^2} + \frac{(x - y - \text{center}_x + \text{center}_y)^2}{(1 - \text{ratio}) \cdot h^2} \leq 1$$

$$\text{Mask}_2 = \frac{(x + y - \text{center}_x - \text{center}_y)^2}{(1 - \text{ratio}) \cdot h^2} + \frac{(x - y - \text{center}_x + \text{center}_y)^2}{\text{ratio} \cdot h^2} \leq 1$$

- Trong đó:

- x và y là các mảng tọa độ được tạo bằng hàm `np.ogrid[:h, :w]`.
- `Mask_1` và `Mask_2` là các mảng boolean, với giá trị `True` cho các pixel nằm trong hình elip và `False` cho các pixel bên ngoài.

3. Kết hợp mặt nạ:

- Hai mặt nạ được kết hợp bằng phép toán logic OR (`|`) để giữ lại các pixel nằm trong ít nhất một trong hai hình elip:

$$\text{Mask} = \text{Mask}_1 | \text{Mask}_2$$

4. Áp dụng mặt nạ lên ảnh:

- Một mảng mới `elliptical_image` có cùng kích thước với ảnh gốc được khởi tạo để lưu trữ kết quả.
- Vòng lặp qua từng kênh màu (R, G, B) của ảnh gốc:

$$\text{elliptical_image}[:, :, c] = \text{img_array}[:, :, c] \times \text{Mask}$$

- Các pixel nằm ngoài hình elip (giá trị `False` trong `Mask`) được đặt về 0, giữ lại chỉ các pixel nằm trong hình elip.

5 Kết quả

Kết quả của đồ án được thể hiện qua các hình ảnh sau đây. Kích thước ảnh gốc là 512 x 512 pixel.

5.1 Ảnh gốc



Hình 1: Ảnh gốc

5.2 Thay đổi độ sáng



Hình 2: Ảnh tăng sáng nếu nhập vào số dương (50)

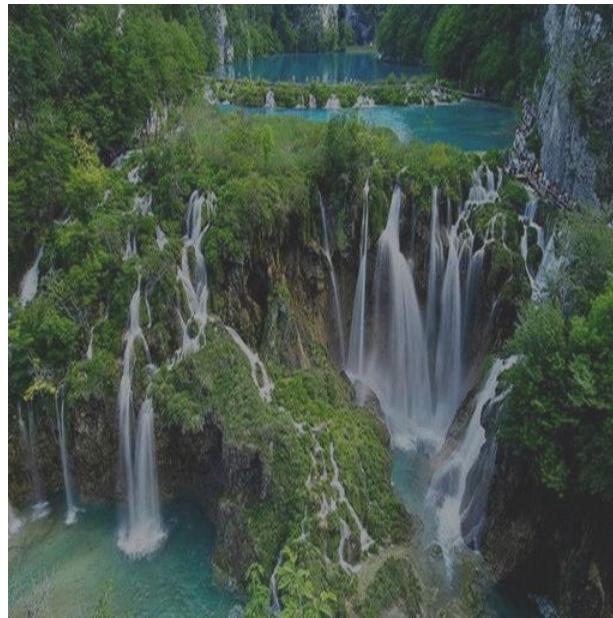


Hình 3: Ảnh giảm sáng nếu nhập vào số âm (-50)

5.3 Thay đổi độ tương phản



Hình 4: Ảnh tăng độ tương phản nếu nhập vào số lớn hơn 1 (1.5)



Hình 5: Ảnh giảm độ tương phản nếu nhập vào số nhỏ hơn 1 (0.5)

5.4 Ảnh lật ngang



Hình 6: Ảnh lật ngang

5.5 Ảnh lật dọc



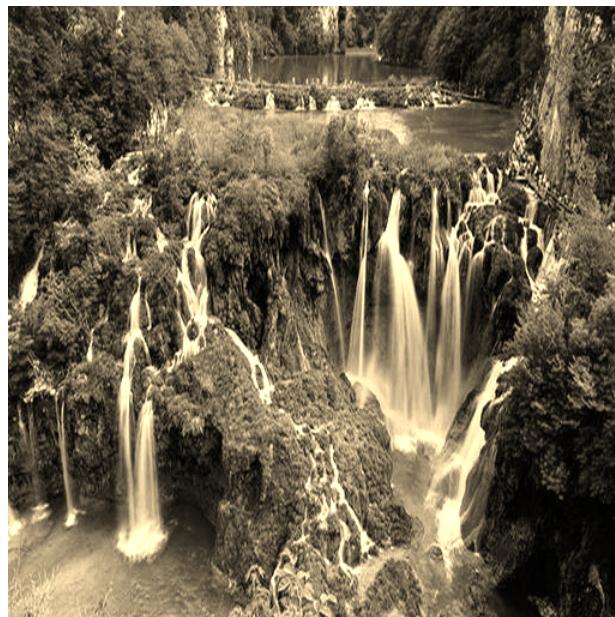
Hình 7: Ảnh lật dọc

5.6 Chuyển đổi sang ảnh xám



Hình 8: Ảnh chuyển sang ảnh xám

5.7 Chuyển đổi sang ảnh sepia



Hình 9: Ảnh chuyển sang ảnh sepia

5.8 Làm mờ ảnh



Hình 10: Ảnh làm mờ với kernel 3x3



Hình 11: Ảnh làm mờ với kernel 5x5

5.9 Làm sắt nét ảnh



Hình 12: Ảnh làm sắt nét với kernel 3x3

5.10 Cắt ảnh theo kích thước trung tâm



Hình 13: Ảnh cắt theo kích thước trung tâm (200 x 200 pixel)

5.11 Cắt ảnh theo khung hình tròn



Hình 14: Ảnh cắt theo khung hình tròn

5.12 Cắt ảnh theo khung 2 hình elip chéo nhau



Hình 15: Ảnh cắt theo khung 2 hình elip chéo nhau (ratio = 0.75)

5.13 Thời gian chạy

Bảng 2: Thời gian chạy của các thuật toán xử lý ảnh

Thuật toán	Thời gian chạy (giây)
Thay đổi độ sáng	< 1
Thay đổi độ tương phản	< 1
Lật ảnh ngang	< 1
Lật ảnh dọc	< 1
Chuyển đổi sang ảnh xám	< 1
Chuyển đổi sang ảnh sepia	< 1
Làm mờ ảnh (kernel 3x3)	5.32
Làm mờ ảnh (kernel 5x5)	8.05
Làm sắc nét ảnh	< 1
Cắt ảnh theo kích thước trung tâm	< 1
Cắt ảnh theo khung hình tròn	< 1
Cắt ảnh theo khung 2 hình elip chéo nhau	< 1

6 Nhận xét

Tài liệu

- [1] OpenCV Documentation, *Color conversions*, Jul 03, 2025,
https://docs.opencv.org/4.12.0/de/d25/imgproc_color_conversions.html

- [2] dyclassroom.com, *How to convert a color image into sepia image*
[https://dyclassroom.com/image-processing-project/
how-to-convert-a-color-image-into-sepia-image](https://dyclassroom.com/image-processing-project/how-to-convert-a-color-image-into-sepia-image)
- [3] GeeksforGeeks, *Convolutional Kernels*, Jul 23, 2025,
<https://www.geeksforgeeks.org/deep-learning/types-of-convolution-kernels/>