HO CHI MINH UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

----------o0o----------

Applied Mathematics and Statistics for Information Technology

Project 2: Image Processing

Student ID : 23127206
Student name : Lê Nguyễn Nhật Khánh
Class : 23CLC04

Ho Chi Minh City, July 23 2025

Mục lục

I. Ý tưởng thực hiện

1. *Tổng quan về đồ án*

- Trong thời đại số, hình ảnh đóng vai trò quan trọng trong lưu trữ, truyền thông, quảng cáo và trí tuệ nhân tạo. Cùng với sự phát triển của các thiết bị số, nhu cầu xử lý và chỉnh sửa ảnh ngày càng phổ biến hơn, không chỉ trong lĩnh vực chuyên môn mà còn cả trong đời sống hàng ngày.
- Xử lý ảnh số là quá trình áp dụng các thuật toán lên ma trận điểm ảnh (pixels) nhằm thay đổi đặc tính hoặc cấu trúc hình ảnh. Trong đồ án này, chúng ta sẽ thực một chương trình xử lý ảnh cơ bản với những chức năng đáng chú ý sau đây:
  - Điều chỉnh độ sáng, độ tương phản
  - Chuyển ảnh thành dạng grayscale và sepia
  - Lật ảnh và cắt ảnh
  - Làm mờ và làm rõ ảnh
- Đây là những kỹ thuật cơ bản nhưng có tính ứng dụng cao, giúp chúng ta hiểu rõ hơn về bản chất của ảnh số và cách các phép toán ảnh hưởng đến từng pixel trong ảnh.

2. *Cách thức nhập dữ liệu và xuất file ảnh*

- Cách thức nhập dữ liệu
  - Chương trình sẽ yêu cầu bạn nhập đường dẫn đến ảnh. Bạn có thể chỉ nhập tên file ảnh nếu file ảnh nằm cùng cấp với file .ipynb.
    Ex: 'D:/Personal/image_processing/photo/test1.png', 'photo.jpeg', 'abc.png', 'xyz.jpg', '/home/userA/ image_processing /test1.png',…
  - Sau đó chương trình xuất ra một menu trên màn hình. Bạn cần chọn chức năng mà bạn muốn thức hiện bằng cách nhập một trong các số từ "0-7".
  - Nếu bạn chọn một hoặc hai thì bạn cần phải nhập thêm một giá trị alpha bất kì để giúp chương trình thực hiện được chức năng mà bạn yêu cầu.
- Cách thức xuất file ảnh
  - Với mỗi chức năng từ 1-7 khi được thực hiện thì sẽ luôn xuất ra file ảnh sau khi đã xử lý dưới định dạng giống như ảnh đầu vào và lưu vào thư mục đang chứa source code.
  - Với chức năng 0, chương trình sẽ xuất tất cả các ảnh đã được xử lý ra thư mục chứa source code với mỗi ảnh đều có phần ghi chú đi kèm trong tên để nhận biết là được tạo ra từ chức năng nào.

3. *Mục tiêu và ý tưởng thực hiện*

- *Mục tiêu của đồ án*
  - Làm quen với các thao tác cơ bản trong xử lý ảnh.
  - Hiểu được bản chất dữ liệu ảnh số, cách ảnh được biểu diễn dưới dạng ma trận các giá trị màu.
  - Biết cách sử dụng jupyter notebook.
  - Rèn luyện kĩ năng lập trình python.
  - Tạo nền tảng cho việc học các kỹ thuật xử lý ảnh phức tạp hơn sau này.
- *Ý tưởng cốt lõi*
  - Đọc ảnh gốc từ đường dẫn người dùng nhập và chuyển thành mảng ba chiều, trong đó mỗi điểm ảnh chứa ba giá trị tương ứng với cường độ của ba kênh màu RGB.
  - Gọi đến hàm mà người dùng yêu cầu để xử lý ảnh đầu ra.
  - In ảnh lên màn hình và xuất ra file.

4. *Bảng đánh giá mức độ hoàn thành*

No	Chức năng	Hoàn thành
1	Tăng/giảm độ sáng	☑
2	Thay đổi độ tương phản	☑
3	Lật ảnh (ngang/dọc)	☑
4	Chuyển đổi ảnh thành grayscale	☑
5	Chuyển đổi ảnh thành sepia	☑
6	Làm mờ ảnh	☑
7	Làm nét ảnh	☑
8	Cắt ảnh ¼ từ trung tâm	☑
9	Cắt ảnh theo hình tròn và ellip	☑

II. Chi tiết thực hiện

1. *Hàm main*
   - Mục tiêu của hàm: đây là hàm chính để chạy chương trình, hàm điều phối mọi hoạt động của chương trình. Hàm sẽ nhận vào đường dẫn đến ảnh và yêu cầu của người dùng, sau đó hàm sẽ xử lý ảnh đó để tìm được ảnh đầu ra, cuối cùng là xuất ra màn hình và lưu vào file.
   - Mô tả hàm:
     - Hàm sẽ in ra hướng dẫn nhập ảnh và yêu cầu người dùng nhập đường dẫn đến ảnh cần xử lý.
     - Sau đó, hàm sẽ đọc ảnh và chuyển đổi sang dưới dạng numpy.array.
     - Trích xuất tên file ảnh từ đường dẫn mà người dùng nhập.
     - Xuất ra màn hình các tính năng mà chương trình có thể xử lý và yêu cầu người dùng chọn.
     - Gọi hàm "process_image" để xử lý ảnh, xuất ảnh sau xử lý ra màn hình và lưu thành file.
2. *Hàm process_image*
   - Mục tiêu của hàm: dùng để gọi đến các hàm xử lý tương ứng với từng chức năng mà người dùng yêu cầu.
   - Mô tả hàm: ứng với từng trường hợp, chúng ta sẽ in ra ảnh gốc, gọi đến hàm xử lý, in ảnh đã xử lý ra màn hình và lưu thành file.
   - Input của hàm:
     - "img": là mảng dưới dạng numpy, trong đó mỗi điểm ảnh chứa ba giá trị tương ứng với ba giá trị của RGB (ảnh gốc).
     - "name": tên của ảnh gốc.
     - "choice": là lựa chọn của người dùng.
3. *Hàm read_img*
   - Mục tiêu của hàm: đọc ảnh từ ổ đĩa, chuyển về định dạng RGB chuẩn.
   - Mô tả hàm: dùng thư viện PIL để mở ảnh và chuyển tất cả ảnh về 3 kênh màu RGB.
   - Input của hàm: đường dẫn đến file ảnh cần mở.
   - Output của hàm: ảnh ở định dạng PIL.Image với chế độ 3 kệnh màu RGB.
4. *Hàm show_img*
   - Mục tiêu của hàm: hiển thị ảnh bằng thư viện matplotlib.
   - Mô tả hàm:
     - Dùng "plt.imshow" để vẽ ảnh.
     - Tắt trục bằng axis để ảnh gọn gàng hơn.
     - Hiển thị ảnh lên màn hình.

- Input của hàm: mảng dưới dạng numpy.
- Output của hàm: hiển thị ảnh lên màn hình.

5. *Hàm save_img*
   - Mục tiêu của hàm: lưu ảnh vừa tạo được thành file. Tên của ảnh mới sẽ được bổ sung phần mô tả cho biết ảnh này được tạo từ chức năng nào.
   - Mô tả hàm:
     - Trích xuất tên ảnh, phần mở rộng từ tên ảnh được truyền vào hàm.
     - Tạo tên mới bằng tên ảnh, phần mô tả và phần mở rộng.
     - Lưu ảnh vào ổ đĩa bằng lệnh "save".
   - Input của hàm:
     - "img" là mảng dưới dạng numpy array (ảnh gốc).
     - "name" là tên của ảnh gốc.
     - "spec" là phần mô tả về chức năng đã tạo thành ảnh.
   - Output của hàm: lưu ảnh thành công vào đĩa.
6. *Hàm adjust_brightness*
   - Mục tiêu của hàm: tăng hoặc giảm độ sáng của ảnh bằng cách cộng một giá trị cố định vào từng pixel.
   - Mô tả hàm:
     - Chuyển kiểu dữ liệu ảnh sang int16 để tránh hiện tượng tràn số khi cộng với giá trị nhập vào.
     - Cộng giá trị "val" vào từng pixel.
     - Dùng npy.clip để đảm bảo mọi giá trị nằm trong khoảng hợp lệ [0, 255].
     - Chuyển kết quả về lại unit8 và trả về kết quả.
   - Input của hàm:
     - "img" là mảng dưới dạng numpy array (ảnh gốc).
     - "val" là giá trị dùng để cộng vào ma trận. Nếu người dùng không truyền giá trị vào hàm thì mặc định là 40.
   - Output của hàm:
     - "img" là mảng dưới dạng numpy array (ảnh sau khi thay đổi độ sáng).
7. *Hàm adjust_contrast*
   - Mục tiêu của hàm: điều chỉnh độ tương phản của ảnh bằng hệ số nhân "val".
   - Mô tả hàm:
     - Nếu giá trị của hệ số tương phản "val > 1" thì ảnh sau xử lý sẽ được tăng độ tương phản và ngược lại.
     - Kiểm tra trường hợp "val = 0" thì coi như là không thay đổi và trả về ảnh gốc.
     - Chuyển về kiểu float để tránh tràn số và tăng độ chính xác.
     - Chuẩn hóa giá trị các pixel về [0, 1], sau đó trừ cho 0.5 chúng ta sẽ được các giá trị âm và dương đối xứng nhau. Miền giá trị lúc này là [-0.5, 0.5].
     - Bây giờ chúng ta nhân các pixel với với hệ số tương phản, các điểm ảnh sáng thì sẽ tiến về sáng và ngược lại. Nhân với hệ số càng lớn thì tương phản càng rõ ràng.
     - Cuối cùng là chúng ta cộng cho 0.5 và nhân với 255 để trả về mức pixel gốc.
     - Dùng numpy.clip để đảm bảo mọi giá trị nằm trong khoảng hợp lệ và trả về kiểu unit8.
   - Input của hàm:
     - "img" là mảng dưới dạng numpy array (ảnh gốc).

- "val" là giá trị dùng để nhân vào ma trận. Nếu người dùng không truyền giá trị vào hàm thì mặc định giá trị này là 1.4.

- Output của hàm:
  - "img_final" là mảng dưới dạng numpy array (ảnh sau khi thay đổi tương phản).

8. *Hàm flip_image*
   - Mục tiêu của hàm: tạo ảnh được lật ngang và lật dọc từ ảnh gốc.
   - Mô tả hàm
     - Lật dọc ảnh bằng cách đảo thứ tự các dòng "img[::-1, :, :]".
     - Lật ngang ảnh bằng cách đảo thứ tự các cột "img[:, ::-1, :]".
   - Input của hàm:
     - "img" là mảng dưới dạng numpy array (ảnh gốc).
   - Output của hàm:
     - "vertical_flip" là mảng dưới dạng numpy array (ảnh sau khi lật dọc).
     - "horizontal_flip" là mảng dưới dạng numpy array (ảnh sau khi lật ngang).
9. *Hàm convert_to_grayscale*
   - Mục tiêu của hàm: chuyển đổi ảnh màu RGB thành ảnh xám bằng cách sử dụng công thức tổng hợp độ sáng từ ba kênh màu.
   - Mô tả hàm
     - Hàm sử dụng công thức chuẩn để chuyển từ RGB sang grayscale

       Gray = 0.299 x R + 0.587 x G + 0.114B
     - Thực hiện phép nhân vô hướng 2 ma trận bằng hàm "numpy.dot".
     - Từ đó tạo ra ảnh mới với mỗi pixel chỉ mang 1 giá trị.
   - Input của hàm:
     - "img" là mảng dưới dạng numpy array (ảnh gốc).
   - Output của hàm:
     - "img_grayscale" là mảng dưới dạng numpy array (ảnh grayscale).
10. *Hàm convert_to_sepia*
    - Mục tiêu của hàm: chuyển ảnh màu RGB sang ảnh theo tông màu sepia – một tông màu mang phong cách cổ điển, tạo cảm giác xưa cũ như những thước phim thời trước.
    - Mô tả hàm
      - Chuyển đổi numpy array sang dạng float32 nhằm tăng độ chính xác và kích thước của nó.
      - Sử dụng bộ lọc sepia có dạng như sau:

        newRed = 0.393*R + 0.769*G + 0.189*B
        newGreen = 0.349*R + 0.686*G + 0.168*B
        newBlue = 0.272*R + 0.534*G + 0.131*B
      - Nhân từng pixel với ma trận sepia chuyển vị để tạo ảnh mới.
      - Dùng numpy.clip để đảm bảo mọi giá trị nằm trong khoảng hợp lệ và trả về kiểu unit8.
    - Input của hàm:
      - "img" là mảng dưới dạng numpy array (ảnh gốc).
    - Output của hàm:
      - "sepia_img" là mảng dưới dạng numpy array (ảnh sepia).
11. *Hàm apply_blur*
    - Mục tiêu của hàm: làm mờ ảnh bằng bộ lọc Gaussian kernel 5x5.
    - Một số ưu điểm của Gaussian kernel 5x5
      - Hiệu quả với ảnh có độ phân giải cao vì ảnh có độ phân giải cao cần tính toán vùng rộng hơn để hiệu ứng làm mờ trông rõ ràng hơn.

- Sẽ cho hiệu ứng làm mờ mượt hơn, mềm mại hơn so với 3x3 hoặc blur. Vì nó phải tính toán trên một vùng rộng hơn xung quanh mỗi pixel, do đó nó sẽ cho ra hiệu ứng làm mờ mềm mại và tự nhiên hơn.
- Giảm nhiễu tốt hơn, ít gây mất chi tiết cục bộ hơn.

- Mô tả hàm
  - Chúng ta sẽ sử dụng kernel làm mờ Gaussion 5x5 để làm mờ ảnh trong đồ án này

$$\text{Kernel} = \frac{1}{256}\begin{matrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{matrix}$$

  - Mở rộng ma trận bằng cách thêm padding với mode = "reflect" để có thể áp dụng kernel lên các pixel biên mà không bị giới hạn về kích thước. Ở đây chúng ta sử dụng mode "reflect" thay vì "constant" để tránh việc các vùng biên bị tối hoặc sáng một cách không tự nhiên.
  - Tạo ảnh kết quả rỗng bằng "npy.zero_like" có cùng kích thước với ảnh gốc.
  - Trên từng kênh màu, chúng ta sẽ trích xuất các vùng 5x5 và tính tổng phép nhân giữa 2 ma trận để trả về giá trị của pixel tại đó.
  - Cuối cùng dùng numpy.clip để đảm bảo mọi giá trị nằm trong khoảng hợp lệ và trả về kiểu unit8
- Input của hàm:
  - "img" là mảng dưới dạng numpy array (ảnh gốc).
- Output của hàm:
  - "img_blur" là mảng dưới dạng numpy array (ảnh sau khi được làm mờ).

12. <u>Hàm apply_sharpen</u>

- Mục tiêu của hàm: làm ảnh nét hơn bằng cách sử dụng bộ lọc kernel làm nét.
- Mô tả hàm
  - Chúng ta sẽ sử dụng kernel làm nét ảnh như sau:

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

  - Mở rộng ma trận bằng cách thêm padding với mode = "reflect" để có thể áp dụng kernel lên các pixel biên mà không bị giới hạn về kích thước. Ở đây chúng ta sử dụng mode "reflect" thay vì "constant" để tránh việc các vùng biên bị tối hoặc sáng một cách không tự nhiên.
  - Tạo ảnh kết quả rỗng bằng "npy.zero_like" có cùng kích thước với ảnh gốc.
  - Trên từng kênh màu, chúng ta sẽ trích xuất các vùng 3x3 và tính tổng phép nhân giữa 2 ma trận để trả về giá trị của pixel tại đó.
  - Cuối cùng dùng numpy.clip để đảm bảo mọi giá trị nằm trong khoảng hợp lệ và trả về kiểu unit8
- Input của hàm:
  - "img" là mảng dưới dạng numpy array (ảnh gốc).
- Output của hàm:
  - "img_sharp" là mảng dưới dạng numpy array (ảnh sau khi được làm nét).

13. <u>Hàm crop_14_from_center</u>

- Mục tiêu của hàm: dùng để cắt ¼ vùng ảnh giữa tính từ trung tâm.

- Mô tả hàm
  - Chúng ta sẽ lấy chiều dài và rộng của ảnh bằng hàm "shape".
  - Vị trí bắt đầu của chúng ta lúc này sẽ là ¼ chiều dài theo trục Ox, Oy.
  - Vị trí kết thúc sẽ cách vị trị bắt đầu 1 khoảng bằng nửa chiều dài.
  - Ta tìm được các vị trí đó vì nếu diện tích của ảnh mới bằng ¼ ảnh cũ thì ta dễ dàng tính được cạnh của ảnh mới và từ đó đối xứng qua các trục Ox, Oy.
  - Cuối cùng là dùng slicing để cắt ảnh theo các vị trí đã tính toán.
- Input của hàm:
  - "img" là mảng dưới dạng numpy array (ảnh gốc).
- Output của hàm:
  - "cropped_img" là mảng dưới dạng numpy array (ảnh sau khi cắt).

14. <u>Hàm crop_to_ellipse</u>

- Mục tiêu của hàm: dùng để tạo một ảnh mới với những điểm ảnh chỉ được giữ lại khi nó nằm trong hình ellipse.
- Mô tả hàm
  - Đầu tiên chúng ta sẽ tìm tâm của hình vuông gốc.
  - Tiếp theo chúng ta tính bán trục lớn và bán trục nhỏ.
    - Bán trục lớn: chúng ta tính được một nửa của đường chéo sẽ bằng $\frac{\sqrt{2}}{2}$n , bằng phương pháp vẽ hình và đo lường thì chúng ta rút ra được công thức phù hợp của a theo n như sau:

$$a = \left(\frac{\sqrt{2}}{2} * n\right) * 0.875$$

    - Bán trục nhỏ: tương tự như trên ta sẽ có công thức phù hợp của bán trục nhỏ như sau:

$$b = \frac{\sqrt{2}}{4} * n$$

  - Tạo lưới tọa độ ảnh 2 chiều và dịch tâm lưới về chính giữa ảnh. Điều này giúp ích cho chúng ta trong việc xét mask sau này, công thức sẽ gọn hơn.
  - Sử dụng công thức xoay một điểm (x, y) quanh gốc tọa độ theo góc 45°:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \cos(45°) & \sin(45°) \\ -\sin(45°) & \cos(45°) \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}$$

  - Ta sẽ tìm được u và v như sau:

$$u = \frac{(x+y)}{\sqrt{2}} \qquad v = \frac{-x+y}{\sqrt{2}}$$

  - Chúng ta lại có công thức của phương trình ellipse:

$$\frac{u^2}{a^2} + \frac{v^2}{b^2} \leq 1$$

  - Tạo mặt nạ mask1 với ellipse xoay 45°. Sau đó hoán đổi a và b, ta sẽ có được hình ellipse thứ 2 vì chúng vuông góc với nhau. Vậy chúng ta có đầy đủ 2 mặt nạ cần thiết là mask1 và mask2. Dùng phép "or" chúng lại với nhau, ta được mask cần tìm.
  - Tạo ảnh đen có cùng kích thước với ảnh gốc.
  - Áp dụng mask để chép lại các điểm ảnh gốc nếu True và ngược lại.
  - Vậy là chúng ta đã xử lý xong việc cắt ảnh theo hình ellipse.
- Input của hàm:
  - "img" là mảng dưới dạng numpy array (ảnh gốc).

- Output của hàm:
  - "ellip_img" là mảng dưới dạng numpy array (ảnh sau cắt).

15. *Hàm crop_to_circle*
- Mục tiêu của hàm: cắt ảnh thành hình tròn nội tiếp từ ảnh gốc hình vuông.
- Mô tả hàm
  - Chúng ta dễ dàng tìm được tâm và bán kính hình tròn nội tiếp. Với tâm sẽ là (n//2, n//2) và bán kính sẽ là n/2.
  - Tạo các ma trận cột và dòng bằng "npy.ogrid".
  - Tạo mặt nạ mask theo công thức để tìm 1 điểm có thuộc hình tròn hay không như sau:

$$(X - a)^2 + (Y - b)^2 \leq R^2$$

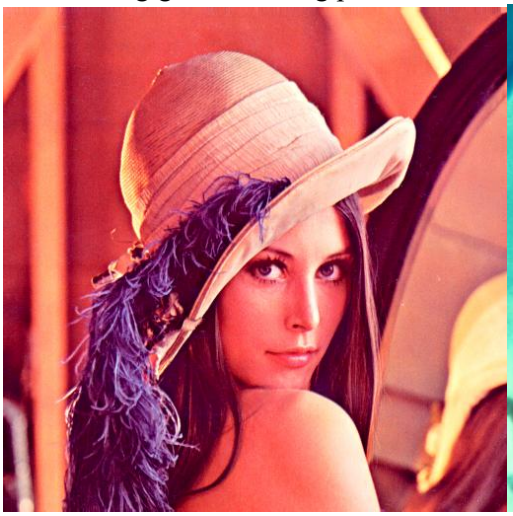
  - Tạo ảnh mới rỗng với kích thước như ảnh gốc bằng câu lệnh "npy.zeros_like".
  - Với mask là ma trận True/False, ta áp mặt nạ lên, với những pixel nằm trong mặt nạ True thì sẽ lấy giá trị từ ảnh gốc và ngược lại sẽ giữ nguyên giá trị (0, 0, 0) tức là màu đen.
  - Từ đó ta đã có được ảnh của hình tròn nội tiếp.
- Input của hàm:
  - "img" là mảng dưới dạng numpy array (ảnh gốc).
- Output của hàm:
  - "masked_img" là mảng dưới dạng numpy array (ảnh sau cắt).

III. Kết quả và kết luận

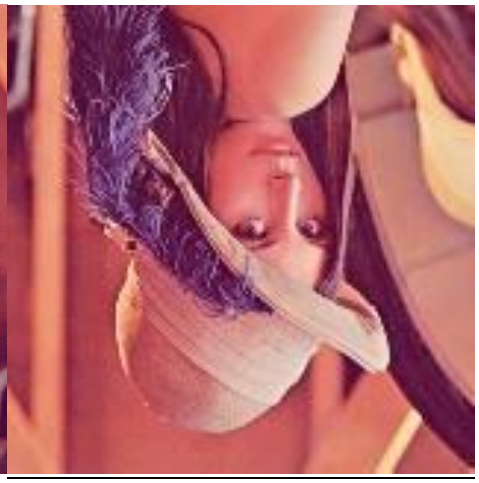
1. *Kết quả*
- Với ảnh gốc "Lenna.png" như sau

- Sau khi tăng/giảm độ sáng

- Sau khi tăng/giảm độ tương phản



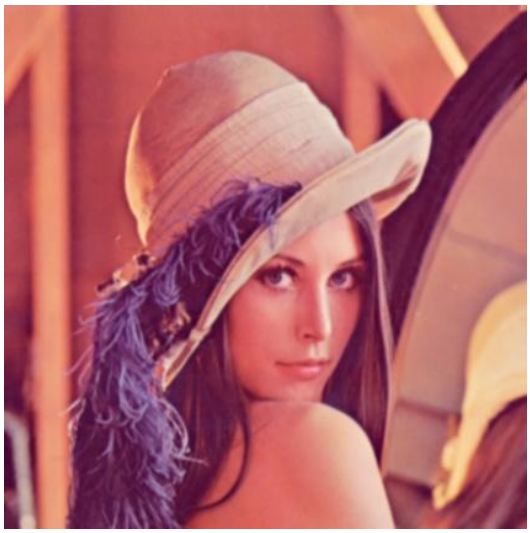

- Lật ảnh ngang + dọc

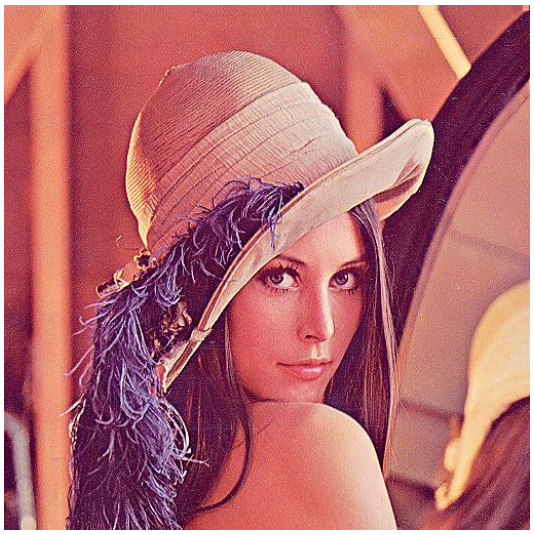
- Chuyển ảnh sang grayscale

- Chuyển ảnh sang sepia

- Làm mờ ảnh



- Làm nét ảnh



- Cắt ¼ ảnh từ tâm ảnh

- Cắt ảnh thành hình tròn

- Cắt ảnh thành 2 hình ellipse

2. *Nhận xét chung về đồ án*

- Thông qua đồ án, chúng ta đã xây dựng và triển khai thành công nhiều hàm xử lý ảnh cơ bản bằng cách thao tác trực tiếp trên ma trận pixel – một cách tiếp cận giúp hiểu rõ bản chất của xử lý ảnh số. Mặc dù các thuật toán được cài đặt thủ công, đơn giản nhưng lại mang tính ứng dụng cao và thường xuyên xuất hiện trong thực tế, như chuyển ảnh sang thang xám, áp bộ lọc làm mờ/làm nét hay cắt ảnh theo hình dạng đặc biệt như tròn và ellipse.
- Tuy các hàm đều hoạt động đúng và cho kết quả trực quan nhưng thời gian thực thi vẫn còn hạn chế, đặc biệt với ảnh có kích thước lớn. Do đó, việc tối ưu hiệu suất, tái sử dụng mã nguồn hoặc kết hợp thêm các thư viện chuyên dụng là cần thiết cho hướng phát triển của đồ án trong tương lai.

IV. Tài liệu tham khảo

- Geeksforgeeks: https://www.geeksforgeeks.org/java/image-processing-in-java-colored-image-to-sepia-image-conversion/
- Wikipedia:
  - https://en.wikipedia.org/wiki/Grayscale
  - https://en.wikipedia.org/wiki/Kernel_(image_processing)
- Github
  - https://github.com/NgocTien0110/Applied-Mathematics-and-Statistics
  - https://github.com/faithdanghuy/Image-Processing

V. Acknowledgement

- Đồ án có sự giúp đỡ của bạn Tài trong ý tưởng của hàm cắt theo hình tròn, ellip về việc tạo mask và ý tưởng của hàm xử lý constrast.
- Đồ án có sự giúp đỡ của bạn Long trong hàm show_img về trường hợp ảnh grayscale.
- Có sự giúp đỡ của ChatGPT trong việc:
  - Viết report
  - Tìm hiểu các hàm trong các thư việc cần dùng
  - Tìm các kiến thức về toán
  - Fix bug nhanh chóng hơn
  - Tạo ảnh để test
  - Tạo hàm đếm thời gian
  - Tối ưu hóa code bằng các hàm có sẵn để thuật toán chạy nhanh hơn