

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**  
**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**  
**KHOA CÔNG NGHỆ THÔNG TIN**



BÁO CÁO ĐỒ ÁN THỰC HÀNH

**MÔN: TOÁN ỨNG DỤNG VÀ THỐNG KÊ**  
**(MTH00057 – 21CLC01)**

**LAB02: IMAGE PROCESSING**

**GIẢNG VIÊN LÝ THUYẾT**

**NGUYỄN ĐÌNH THÚC**

**GIẢNG VIÊN THỰC HÀNH**

**NGUYỄN VĂN QUANG HUY**

**NGÔ ĐÌNH HY**

**SINH VIÊN THỰC HIỆN**

**21127621 – ÂU DƯƠNG KHANG**

## **MỤC LỤC**

<b>1) Tổng quan</b>	3
<b>2) Cài đặt</b>	3
<b>3) Danh sách testcase</b>	7
<b>4) Đánh giá</b>	15
<b>5) Nguồn tham khảo</b>	15

## 1) Tổng quan

- Chương trình được thử nghiệm trên môi trường text editor Visual Studio Code với phiên bản Python 3.11.0.
- Đánh giá mức độ hoàn thành các chức năng:

<u>Chức năng</u>	<u>Mức độ hoàn thành</u>
Thay đổi độ sáng của ảnh	100%
Thay đổi độ tương phản của 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%
Cắt ảnh theo khung tròn	100%
Cắt ảnh theo khung 2 ellipse chéo nhau	100%

## 2) Cài đặt

- Chương trình có sử dụng các thư viện dưới đây:

```
1 from PIL import Image
2 import numpy as np
3 import matplotlib.pyplot as plt
```

- Hàm phụ trợ:
  - + `def advanced(value):`: điều chỉnh số đưa về khoảng [0, 255].
  - + `def save_image(img, name, option, mode=0):`: xử lý lưu hình ảnh theo tên các chức năng chính.
  - + `def choose_option(img, name, option):`: xử lý các tùy chọn mà người dùng đưa vào.
  - + `def main:`: giao diện chính của chương trình, đưa ra menu, đưa hình ảnh vào và xử lý các chức năng theo tùy chọn người dùng.

- Hàm chức năng chính:

- + Chức năng điều chỉnh độ sáng: `def change_brightness(img, img_new, br):`

Lấy ra các giá trị của kênh màu, với mỗi kênh màu ta thay đổi giá trị bằng cách cho kênh màu cũ cộng với độ sáng ta mong muốn thay đổi, sau đó điều

chỉnh các chỉ số của kênh màu bằng hàm phụ trợ `advanced`(value) đưa về khoảng giá trị [0,255], từ đó ta có hình ảnh sau khi thay đổi độ sáng.

- INPUT: hình ảnh muốn chỉnh độ sáng, độ sáng muốn thay đổi.
- OUTPUT: hình ảnh sau khi chỉnh độ sáng.

+ Chức năng thay đổi độ tương phản: `def change_contrast(img, img_new, co)`:

Lấy ra các giá trị của kênh màu, với mỗi kênh màu ta thay đổi giá trị bằng cách cho kênh màu cũ nhân với độ tương phản ta mong muốn thay đổi, sau đó điều chỉnh các chỉ số của kênh màu bằng hàm phụ trợ `advanced`(value) đưa về khoảng giá trị [0,255], từ đó ta có hình ảnh sau khi thay đổi độ tương phản.

- INPUT: hình ảnh muốn chỉnh độ tương phản, độ tương phản muốn thay đổi.
- OUTPUT: hình ảnh sau khi chỉnh độ tương phản.

+ Chức năng lật ảnh (ngang - dọc):

- Chức năng lật ảnh ngang: `def horizontal_flip(img)`: sử dụng chức năng có sẵn trong thư viện numpy để lật ngang ma trận chứa giá trị của các điểm ảnh là `np.fliplr`.

- INPUT: ma trận chứa giá trị các điểm ảnh của hình.
- OUTPUT: hình được lật ngang.

- Chức năng lật ảnh dọc: `def horizontal_flip(img)`: sử dụng chức năng có sẵn trong thư viện numpy để lật dọc ma trận chứa giá trị của các điểm ảnh là `np.flipud`.

- INPUT: ma trận chứa giá trị các điểm ảnh của hình.
- OUTPUT: hình được lật dọc.

+ Chức năng chuyển đổi màu ảnh thành màu xám/sepia

- Chức năng đổi màu xám: `def gray_scale(img, img_new)`: để có thể chuyển đổi ảnh thành màu xám ta thay đổi giá trị của kênh màu các điểm ảnh theo công thức:

$$\text{imgGray} = 0.2989 * R + 0.5870 * G + 0.1140 * B.$$

áp dụng cho tất cả các cột màu.

- INPUT: ma trận chứa giá trị các điểm ảnh của hình.
  - OUTPUT: hình được thay đổi thành màu xám.
- Chức năng lật ảnh dọc: `def sepia(img, img_new)`: để có thể chuyển đổi ảnh thành màu xám ta thay đổi giá trị của kênh màu các điểm ảnh theo công thức:

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

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

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

áp dụng lần lượt cho các cột màu.

- INPUT: ma trận chứa giá trị các điểm ảnh của hình.
- OUTPUT: hình được thay đổi thành màu sepia.

+ Chức năng làm mờ/rõ hình ảnh:

- Chức năng làm mờ: `def blur(img, kernel)`: sử dụng phép tích chập với ma trận kernel để làm mờ ảnh. Phép tích chập tại 1 điểm trung tâm tính bằng cách lấy tổng của tích từng giá trị kernel với ma trận con có cùng kích cỡ với nó. Thay vì duyệt từng giá trị và thực hiện phép chập ở từng vị trí trong mảng hình, ta dịch ma trận theo dòng và cột sau đó nhân với từng giá trị của kernel, sau đó lấy tổng các ma trận vừa tính, ta sẽ được ma trận kết quả sau khi thực hiện phép tích chập.
  - INPUT: ma trận chứa giá trị các điểm ảnh của hình, ma trận kernel cho việc làm mờ.
  - OUTPUT: hình được làm mờ.
- Chức năng làm rõ: `def sharpen(img, kernel)`: tương tự cách xử lý làm mờ ảnh ở trên, tuy nhiên khi này ma trận kernel được đưa vào có giá trị khác phù hợp với việc làm rõ ảnh.
  - INPUT: ma trận chứa giá trị các điểm ảnh của hình, ma trận kernel cho việc làm rõ.
  - OUTPUT: hình được làm rõ.

+ Chức năng cắt ảnh (ở trung tâm): `def crop(img, size)`: hiệu chỉnh kích cỡ ảnh theo dữ liệu nhập vào. Đầu tiên, tạo ảnh mới có kích cỡ theo input, bằng việc xác định tâm từ đó ta in hình từ trung tâm ra sao cho phù hợp với kích cỡ muốn hiệu chỉnh.

- INPUT: ma trận chứa giá trị các điểm ảnh của hình, kích cỡ hình ảnh sau khi cắt.
- OUTPUT: hình được cắt theo kích cỡ đã cho.

+ Chức năng cắt ảnh thành hình tròn: `def circle_crop(img)`: xác định tâm ảnh và bán kính ảnh. Ta có thể tính khoảng cách từ tâm đến các điểm ảnh, nếu khoảng cách đó bé hơn bán kính nghĩa là nó nằm trong đường tròn, nếu lớn hơn có nghĩa là nằm ngoài đường tròn, vì thế ta sẽ đặt tất cả giá trị kênh màu những điểm nằm ngoài đường tròn bằng 0 (màu đen). Từ đó, ta sẽ có hình được cắt theo hình tròn.

- INPUT: ma trận chứa giá trị các điểm ảnh của hình, bán kính hình tròn.
- OUTPUT: hình được cắt thành hình tròn.

+ Chức năng cắt ảnh thành 2 hình ellipse chéo nhau:

`def double_ellipse_crop(img)`: Tương tự như cách giải quyết chức năng cách ảnh thành hình tròn, nhưng chúng ta sẽ sử dụng phương trình ellipse để giải bài toán:

Cho elip có tiêu điểm  $F_1$  và  $F_2$  chọn hệ trục tọa độ  $Oxy$  sao cho  $F_1(-c; 0)$  và  $F_2(c; 0)$ . Khi đó người ta chứng minh được

$$M(x; y) \in \text{elip} \Rightarrow \frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \quad (1)$$

trong đó:  $b^2 = a^2 - c^2$

Đầu tiên, ta cần tìm  $a, b$  bằng cách lấy kích cỡ chiều dọc của hình chia cho 2 và 3 để tìm  $a, b$  của hình ellipse đầu, sau đó chúng ta đảo  $a$  và  $b$  để có  $a$  và  $b$  mới cho hình ellipse thứ 2. Tiếp đến ta tìm tới tâm của bức hình, tính khoảng cách giữa các điểm với tâm, từ đó ta xoay hình theo góc angle đã nhập từ trước. Cuối cùng, dựa vào hình đã xoay ta kiểm tra với phương trình chính tắc. Nếu đúng, ta sẽ đặt giá trị các kênh màu bằng với các kênh màu của hình nằm trong ellipse và ngược lại với các kênh màu của hình nằm ngoài hình ellipse cho bằng 0.

- INPUT: ma trận chứa giá trị các điểm ảnh của hình,  $a, b$ .
- OUTPUT: hình được cắt thành hình 2 ellipse chèo nhau.

### **3) Danh sách testcase**

INPUT:

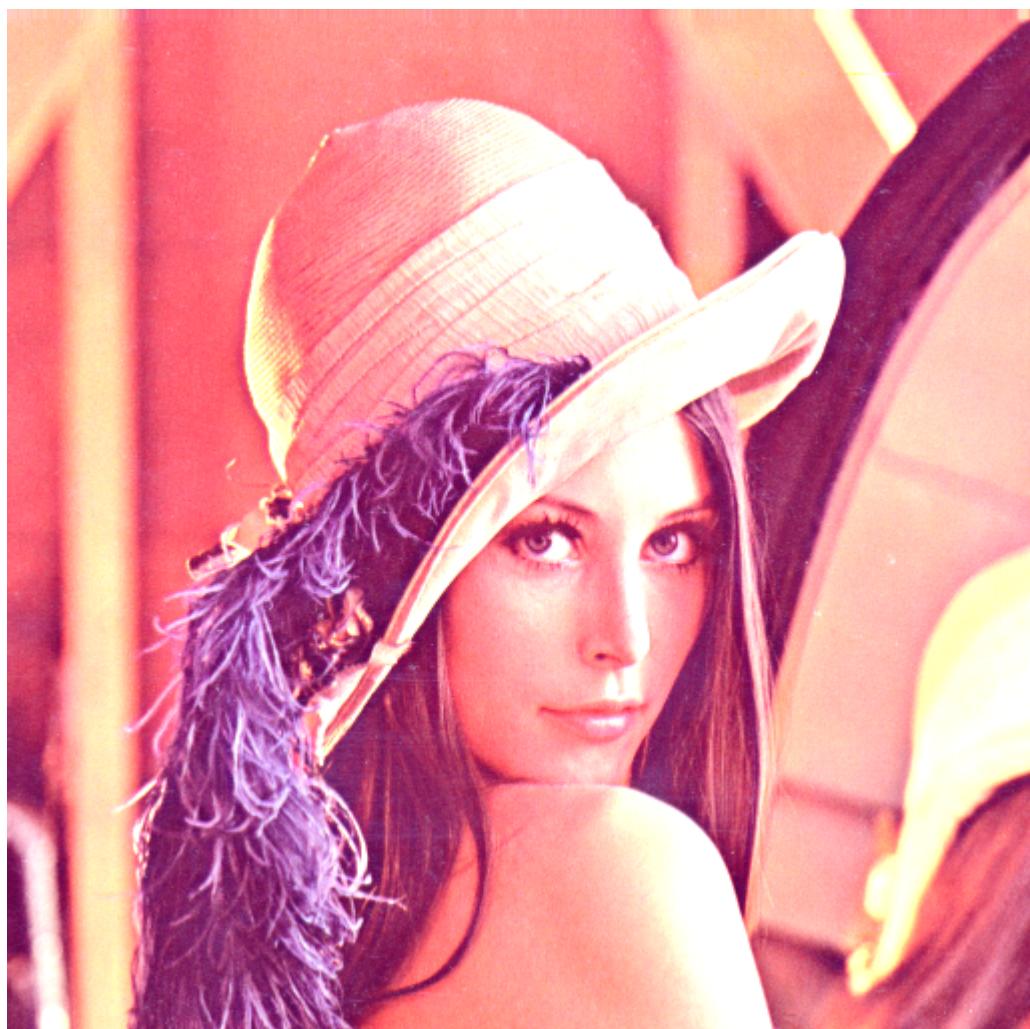


OUTPUT:

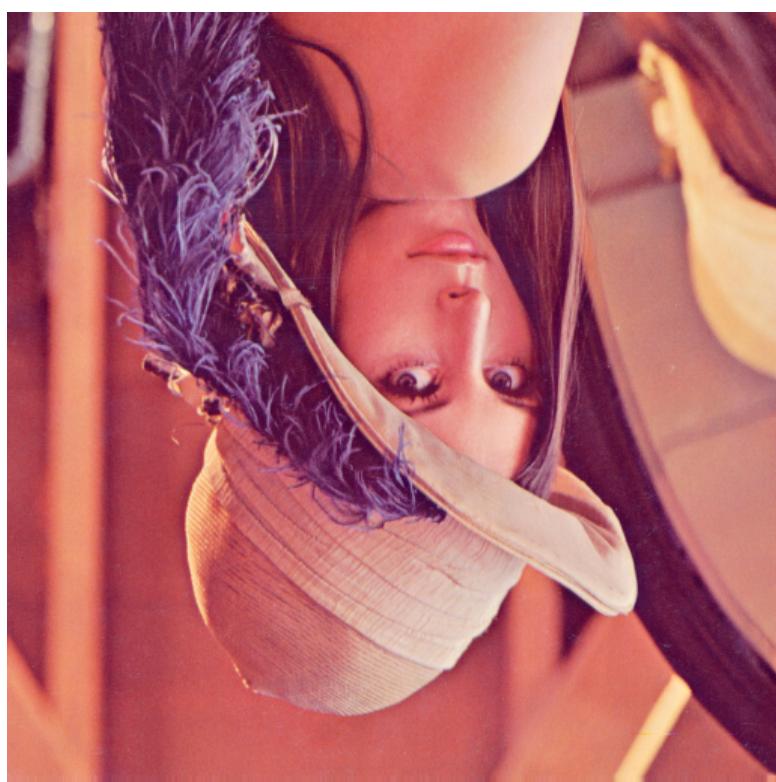
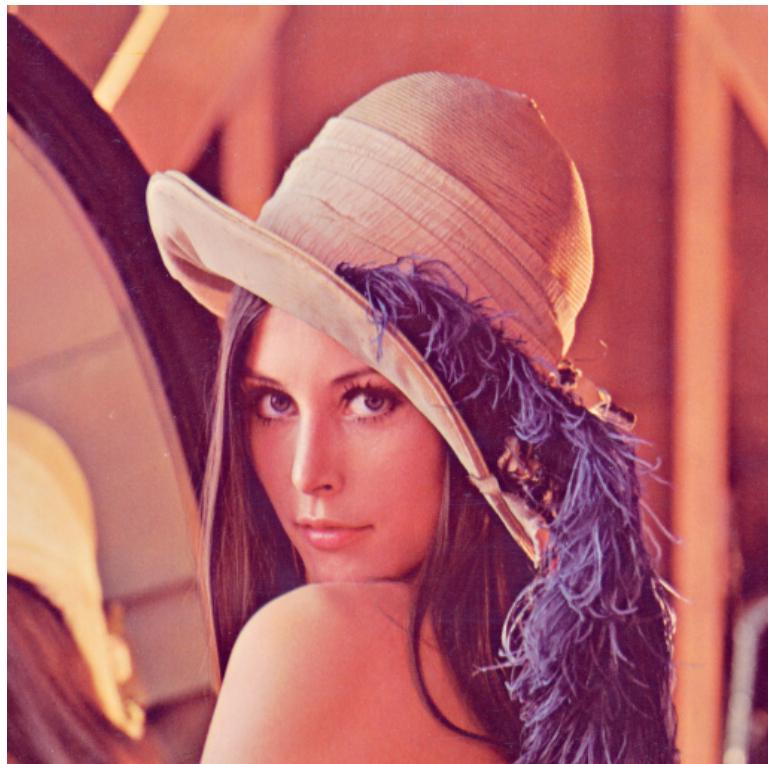
- Thay đổi độ sáng (độ sáng 100)



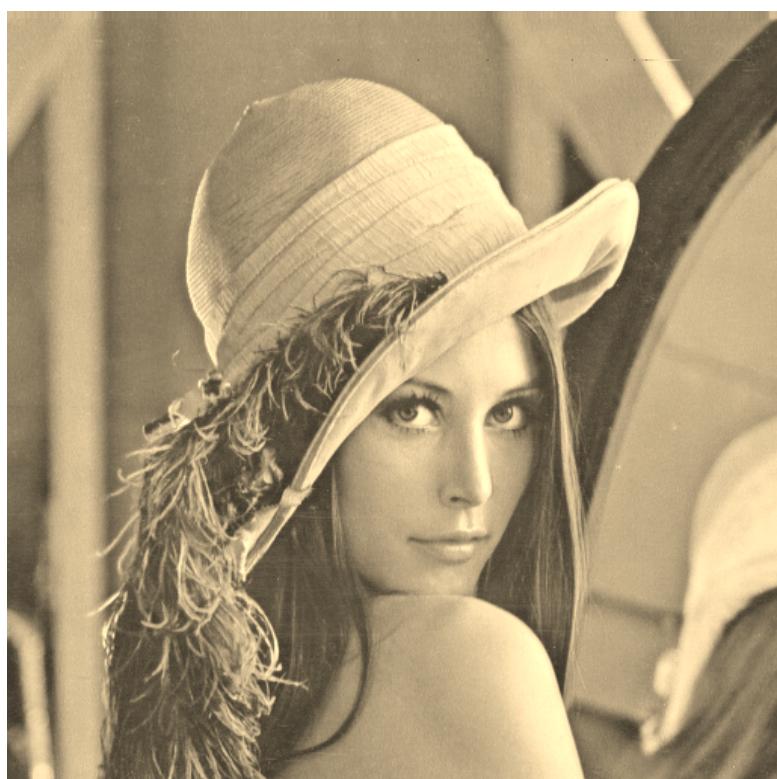
- Thay đổi độ tương phản (độ tương phản 1.5)



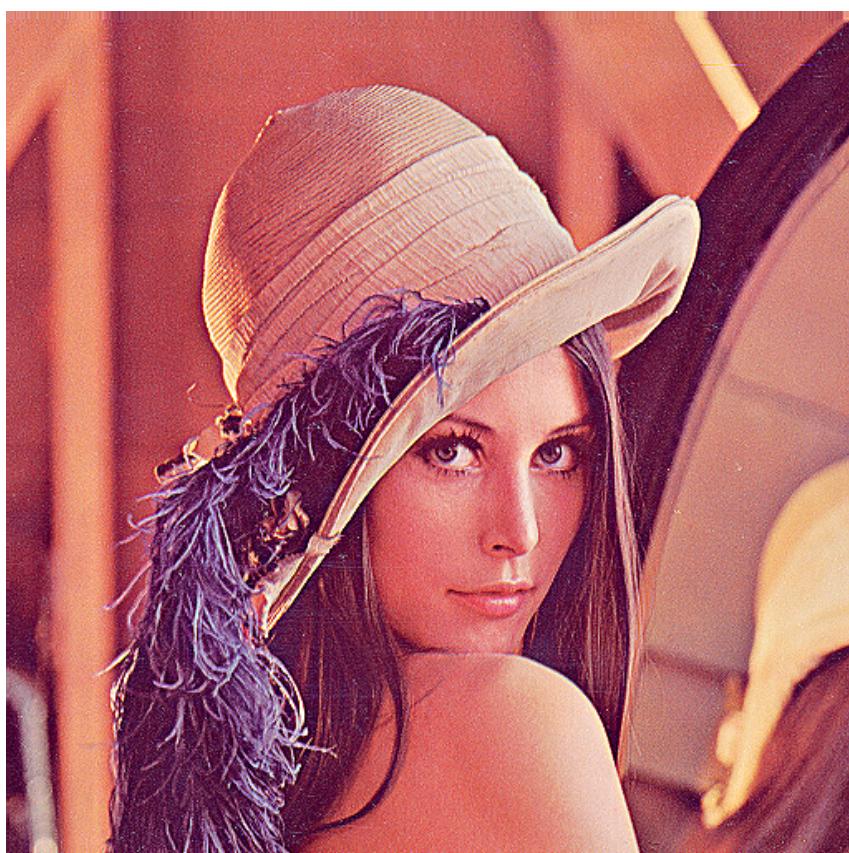
- Lật ngang/đọc hình



- Chuyển thành ảnh xám/sepia



- Làm mờ/rõ ảnh



- Cắt ảnh ở trung tâm (size: 300x300)



- Cắt ảnh thành hình tròn



- Cắt ảnh thành 2 ellipse chéo nhau:



## **4) Đánh giá**

- **Ưu điểm:** Giải thuật cho ra kết quả, không xuất hiện lỗi, thành công thực hiện đầy đủ các chức năng đã yêu cầu, có thể làm trên cả hình chữ nhật và hình vuông.
- **Nhược điểm:** Đối với chức năng làm rõ ảnh (sharpen) vẫn chưa tối ưu thời gian xử lý, còn chậm.

## **5) Nguồn tham khảo**

[Convert Image to Grayscale in Python | Delft Stack](#)

[Xử lý hình ảnh bằng Python \(koodibar.com\)](#)

[image processing - How is a sepia tone created? - Stack Overflow](#)

[github.com - Image processing](#)

[Convolution - Tích chập \(techmaster.vn\)](#)