TRƯỜNG ĐH KHOA HỌC TỰ NHIÊN, ĐHQG HCM

BÁO CÁO

PROJECT 2 – IMAGE PROCESSING

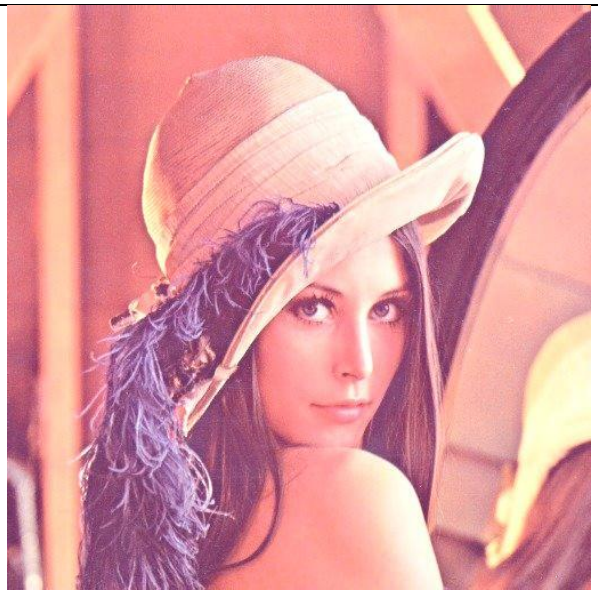
Sinh viên: Nguyễn Quốc Thắng

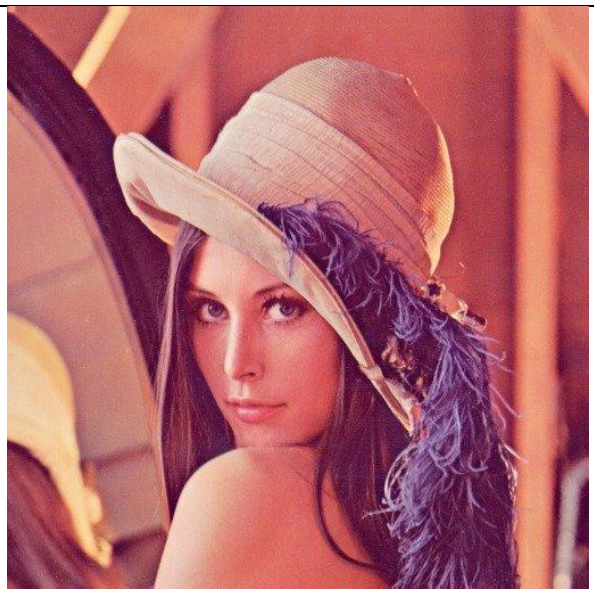
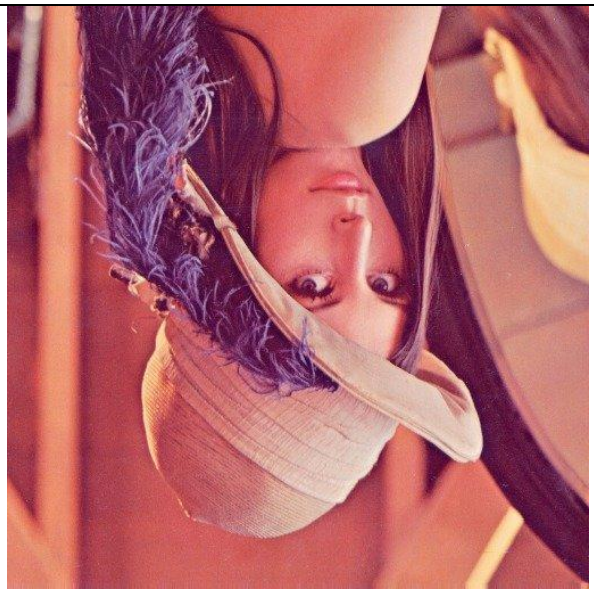
MSSV: 22127385

Lớp: 22CLC01

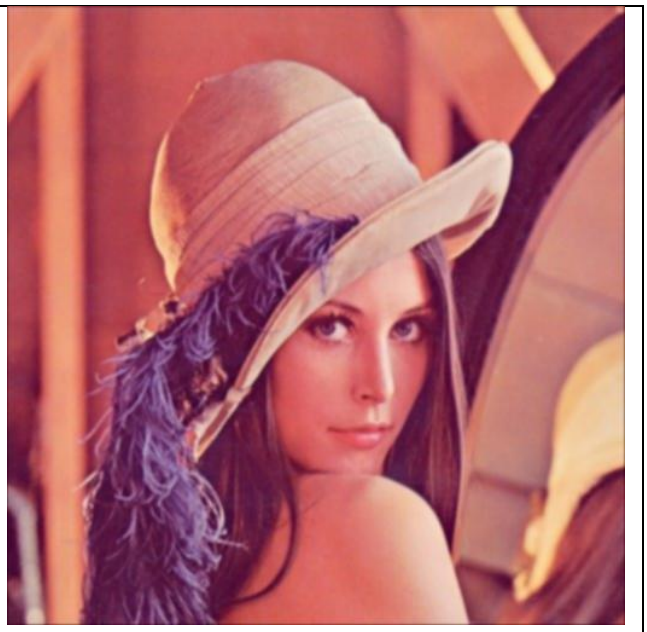
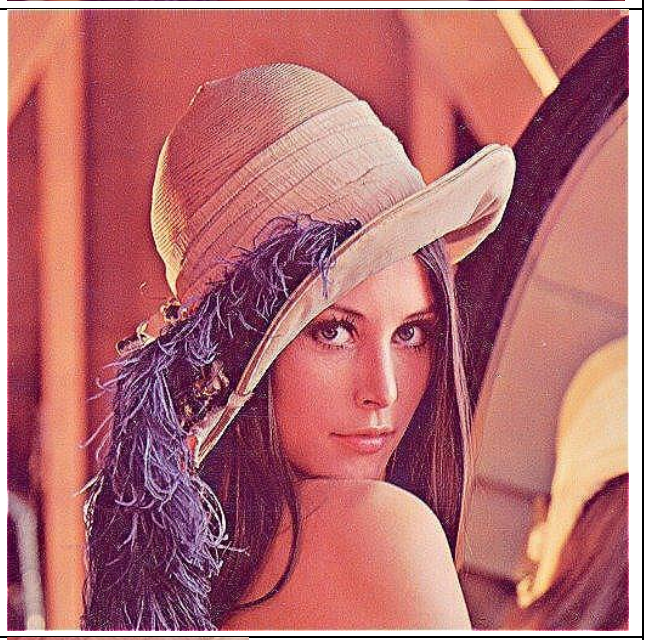
MỤC LỤC

1. Đánh giá mức độ hoàn thành:

STT	Chức năng/ Hàm	Mức độ hoàn thành (%)	Ảnh kết quả
1	Thay đổi độ sáng	100	
2	Thay đổi độ tương phản	100	

3.1	Lật ảnh ngang	100	
3.2	Lật ảnh dọc	100	

4.1	RGB thành ảnh xám	100	
4.2	RGB thành ảnh sepia	100	

5.1	Làm mờ ảnh	100	
5.2	Làm sắc nét ảnh	100	
6	Cắt ảnh theo kích thước	100	

7.1	Cắt ảnh theo khung tròn	100	
7.2	Cắt ảnh theo khung elip	100	
8	Hàm main	100	
9	Phóng to/ Thu nhỏ 2x	100	

2. Hàm Thay đổi độ sáng:

- **Hàm:** `def adjust_brightness(img_2d, brightness):`
- **Ý tưởng:** cộng giá trị độ sáng muốn tăng vào từng phần tử của mảng biểu diễn ảnh.
- **Gồm 2 bước chính:**
  - Chuyển đổi dữ liệu và cộng vào giá trị độ sáng (brightness).
    - **np.uint8(img_2d):** Chuyển đổi mảng img_2d sang kiểu dữ liệu uint8.
  - Giới hạn giá trị pixel trong khoảng từ 0 đến 255 với **np.clip().**

3. Hàm thay đổi độ tương phản:

- **Hàm:** `def adjust_contrast(img_2d, contrast)`
- **Ý tưởng:** Độ tương phản càng cao thì sự khác biệt giữa các giá trị pixel càng lướn, làm cho hình ảnh trở nên rõ nét hơn.
- **Công thức để điều chỉnh độ tương phản hàm:**
  - Tính toán hệ số f: hệ số điều chỉnh, giúp tăng hoặc giảm độ tương phản của hình ảnh

    f = 259 * (contrast + 255) / (255 * (259 - contrast))
  - Điều chỉnh độ tương phản ảnh: mỗi giá trị pixel trong ảnh được nhân với hệ số f và trừ đi 128*f. Việc trừ đi 128*f giúp dịch chuyển giá trị pixel về trung tâm của dãy giá trị (0-255), bảo đảm rằng độ tương phản được điều chỉnh một cách cân bằng.

    img_2d = img_2d * f - 128 * f
  - Giới hạn giá trị pixel: **np.clip()**

4. Hàm Lật ảnh ngang – Lật ảnh dọc:

- **Hàm:** `def flip_horizontally(img_2d)` và `def flip_vertically(img_2d)`
- **Ý tưởng:** sử dụng hàm flip trong thư viện Munpy để thực hiện việc lật ảnh.
- **Các bước thực hiện:**
  - Lật ảnh theo chiều ngang: **np.flip(img_2d, axis=1)**
  - Lật ảnh theo chiều dọc: **np.flip(img_2d, axis=0)**

5. Hàm RGB thành ảnh xám:

- **Hàm**: `def convert_to_grayscale(img_2d)`
- **Ý tưởng:** Hình ảnh màu thường được biểu diễn dới dạng mảng với 3 kênh màu (đỏ, xanh lá, xanh dương). Để chuyển đổi sang ảnh xám, chúng ta sử dụng công thức trọng số để tính giá trị xám từ ba kênh màu.
- **Các bước thực hiện:**
  - **Bước 1.** Tạo mảng trọng số: Sử dụng mạng trogn số **[0.2989, 0.5870, 0.1140]** để tính giá trị xám từ 3 kênh màu.
  - **Bước 2.** Tính giá trị xám cho từng điểm ảnh bằng cách nhân mảng 2D của ảnh với mảng trọng số (**@**).
  - **Bước 3.** Giới hạn giá trị: **np.clip().**

- **Bước 4.** Chuyển đổi kiểu dữ liệu: **np.uint8().**

6. Hàm RGB thành ảnh sepia:

- **Hàm:** `def convert_to_sepia(img_2d)`
- **Ý tưởng:** Sử dụng một bộ lọc sepia để tính toán giá trị màu mới từ ba kênh màu gốc.
- **Các bước thực hiện:**
  - **Bước 1.** Tạo mảng bộ lọc sepia

```
sepia_filter = np.array([[0.393, 0.769, 0.189],
                         [0.349, 0.686, 0.168],
                         [0.272, 0.534, 0.131]])
```

  - **Bước 2.** Tính giá trị màu mới cho từng điểm ảnh bằng cách nhân ma trận ảnh 2D với ma trận chuyển vị của mảng bộ lọc sepia.
  - **Bước 3.** Giới hạn giá trị: **np.clip()** và chuyển đổi dữ liệu **np.uint8.**

7. Hàm Làm mờ ảnh:

- **Hàm**: `def blur_image(img_2d)`
- **Ý tưởng**: thực hiện làm mờ một hình ảnh bằng cách sử dụng một **kernel Gaussian**. **Kernel Gaussian** là một ma trận được sử dụng để làm mờ hình ảnh bằng cách tính trung bình trọng số của các pixel xung quanh.
- **Các bước thực hiện:**
  - **Bước 1.** Tạo Kernel Gaussian.

```
kernel = np.array([[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]]) / 256
```

  - **Bước 2.** Tạo ma trận tạm thời có kích thước lướn hơn hình ảnh gốc để chứa các giá trị pixel mở rộng.
  - **Bước 3.** Áp dụng Kernel Gausian
    - Duyệt qua từng pixel của hình ảnh gốc.
    - Tính giá trị mới cho mỗi pixel bằng cách nhân kernel với vùng lân cận 5x5 của pixel đó trong ma trận tạm thời và tính tổng các giá trị.
  - **Bước 4.** Giới hạn giá trị pixel.

8. Hàm Làm sắc nét ảnh:

- **Hàm:** `def sharpen_image(img_2d)`
- **Ý tưởng:** sử dụng một ma trận kernel để thực hiện phép lọc không gian, làm tăng độ tương phản của các cạnh trong hình ảnh.
- **Các bước thực hiện:**
  - **Bước 1.** Tạo kernel: Kernel là một ma trận nhỏ được sử dụng để áp dụng các phép biến đổi lên hình ảnh. Kernel được thiết kế để làm sắc nét hình ảnh.

```
kernel = np.array([[[0], [-1], [0]],
                   [[-1], [5], [-1]],
                   [[0], [-1], [0]]])
```

  - **Bước 2.** Tạo một ma trận tạm thời lớn hơn hình ảnh gốc để dễ dàng áp dụng kernel mà không gặp vấn đề với các biên của hình ảnh.
  - **Bước 3.** Duyệt qua từng pixel của hình ảnh gốc và áp dụng kernel để tính toán giá trị mới cho pixel đó.
  - **Bước 4.** Giới hạn giá trị pixel.

9. Hàm Cắt theo kích thước ảnh:

- **Hàm:** `def crop_image(img_2d, width, height)`
- **Ý tưởng:** cắt một hình ảnh 2D đến kích thước cụ thể (width và height) bằng cách cắt từ trung tâm của hình ảnh. Điều này giúp giữ lại phần trung tâm của hình ảnh và loại bỏ các phần rìa.
- **Các bước thực hiện:**
  - **Bước 1.** Lấy kích thước hình ảnh gốc. Dùng .shape
  - **Bước 2.** Tính toán tọa độ bắt đầu để cắt hình ảnh từ trung tâm **(start_x, start_y**). Sử dụng phép chia nguyên // để tính.
  - **Bước 3**. Cắt hình ảnh.

```
img_2d = img_2d[start_y:start_y+height, start_x:start_x+width]
```

    - **start_y:start_y+height** xác định phạm vi các hàng (chiều dọc) sẽ được giữ lại trong hình ảnh cắt. Bắt đầu từ hàng start_y và lấy height hàng tiếp theo.
    - **start_x:start_x+width** xác định phạm vi các cột (chiều ngang) sẽ được giữ lại trong hình ảnh cắt. Bắt đầu từ cột start_x và lấy width cột tiếp theo.

10. Hàm Cắt ảnh theo khung tròn:

- **Hàm:** `def crop_to_circle(img_2d)`
- **Ý tưởng:** cắt một hình ảnh thành hình tròn, với tâm của hình tròn nằm ở trung tâm của hình ảnh. Các phần bên ngoài hình tròn sẽ được đặt giá trị bằng 0 (màu đen).
- **Các bước thực hiện:**
  - **Bước 1.** Cắt ảnh thành hình vuông. Sử dụng hàm **crop_image** để cắt ảnh thành hình vuông với kích thước bằng chiều cao của ảnh gốc.
  - **Bước 2.** Xác định các thông số cần thiết:
    - Lấy chiều cao và chiều rộng của ảnh đã cắt.
    - Tính tọa độ tâm của ảnh **(center_x, center_y).**
    - Tính bán kính của hình tròn **(radius)** bằng giá trị nhỏ hơn giữa **center_x** và **center_y.**
  - **Bước 3.** Tạo mặt nạ (mask):
    - Sử **dụng np.ogrid[:height, :width]** để tạo lưới tọa độ y và x.
    - **mask = (x - center_x) ** 2 + (y - center_y) ** 2 > radius ** 2** : tạo mặt nạ với các điểm nằm ngoài hình tròn (các điểm có khoảng cách từ tâm lớn hơn bán kính).
  - **Bước 4.** Áp dụng mặt nạ:
    - Đặt giá trị của các điểm nằm ngoài hình tròn bằng 0.

11. Hàm Cắt ảnh theo khung elip:

- **Hàm:** `def crop_to_2diagonal_ellipses(img_2d, in_width, in_height)`
- **Ý tưởng:** cắt một ảnh thành hai hình elip chéo nhau, nằm ở trung tâm của ảnh. Các bước thực hiện bao gồm cắt ảnh theo kích thước đầu vào, tạo hai mặt nạ elip, kết hợp hai mặt nạ này và áp dụng mặt nạ kết hợp lên ảnh để tạo ra ảnh kết quả.

  Sử dụng Phương pháp thiết lập phương trình elip và công thức quay hệ tọa độ, tìm mối liên hệ giữa chiều dài của cạnh và độ dài trục của elip.

- **Các bước thực hiện:**
  - **Bước 1.** Cắt ảnh theo kích thước đầu vào:

- Sử dụng hàm **crop_image** để cắt ảnh img_2d theo kích thước **in_width** và **in_height.**

- **Bước 2.** Tạo mặt nạ elip:
  - Định nghĩa hàm **create_elipse_mask** để tạo mặt nạ elip với các tham số:
    - **shape**: Kích thước của ảnh.
    - **a, b:** Bán trục lớn và bán trục nhỏ của elip.
    - **center**: Tọa độ trung tâm của elip.
    - **angle:** Góc xoay của elip.

  Cụ thể: Phương trình tổng quát

```
cos_angle = np.cos(np.radians(angle))
sin_angle = np.sin(np.radians(angle))

elipse_mask = (((x - x_center) * cos_angle + (y - y_center) * sin_angle) ** 2 / a ** 2 +
               ((x - x_center) * sin_angle - (y - y_center) * cos_angle) ** 2 / b ** 2) <= 1
```

Trong đó:

- **x** và **y** là tọa độ của các điểm trong mặt nạ.
- **x_center** và **y_center** là tọa độ trung tâm của elip.
- **a** và **b** lần lượt là bán trục lớn và bán trục nhỏ của elip.
- **Gồm các bước:**
  1. Tạo lưới tọa độ:

     y, x = np.ogrid[:shape[0], :shape[1]]
  2. Tính toán các giá trị sin, cos của góc xoay.
  3. Tính toán phương trình elip.
  4. Trả về mặt nạ.

  - Sử dụng hàm **create_elipse_mask** để tạo hai mặt nạ elip với các tham số khác nhau:
    - **mask1:** Elip đầu tiên với góc xoay 45 độ.
    - **mask2:** Elip thứ hai với góc xoay 135 độ.

Note: Do không thể tìm được mối liên hệ hay công thức liên hệ giữa độ dài trục elip và độ dài cạnh của ảnh nên dựa trên quan sát (**đây không phải là công thức dựa trên sự chứng minh)** đã tìm ra sự chênh lệch về tỉ lệ giữa 2 cái là :

Độ dài trục đứng elip = độ dài cạnh ảnh * 200/512

Độ dài trục nằm elip = độ dài cạnh ảnh * 300/512

- **Bước 3.** Kết hợp hai mặt nạ:
  - Sử dụng **toán tử |** để kết hợp hai mặt nạ mask1 và mask2.
- **Bước 4.** Áp dụng mặt nạ lên ảnh:
  - Tạo bản sao của ảnh gốc **img_2d.**
  - Áp dụng mặt nạ kết hợp lên ảnh sao chép, các pixel không thuộc mặt nạ sẽ được đặt giá trị bằng **0.**

12. Hàm Phóng to/ Thu nhỏ 2x:

- **Hàm:** `def scale_image(img_2d, scale_factor)`
- **Ý tưởng:** tính toán kích thước mới của hình ảnh dựa trên hệ số tỷ lệ và sau đó tạo ra một hình ảnh mới với kích thước đã tính toán, trong đó mỗi pixel của hình ảnh mới được lấy mẫu từ hình ảnh gốc.
- **Các bước thực hiện:**
  - **Bước 1.** Lấy kích thước của hình ảnh gốc: Lấy chiều cao **(img_height)** và chiều rộng **(img_width)** của hình ảnh gốc.
  - **Bước 2.** Tính toán kích thước mới.
    - Dựa trên **hệ số tỷ lệ (scale_factor),** tính toán chiều cao mới **(new_height**) và chiều rộng mới **(new_width)** của hình ảnh.
  - **Bước 3.** Tạo hình ảnh mới. Tạo một mảng numpy mới (scaled_img) với kích thước mới và kiểu dữ liệu là **np.uint8.**
  - **Bước 4.** Lấy mẫu lại hình ảnh: Duyệt qua từng pixel của hình ảnh mới và gán giá trị của nó bằng giá trị của pixel tương ứng trong hình ảnh gốc. Điều này được thực hiện bằng cách chia chỉ số của pixel mới cho hệ số tỷ lệ để tìm chỉ số của pixel tương ứng trong hình ảnh gốc.
  - **Bước 5**. Trả về hình ảnh mới.

13. Một số hàm bổ trợ:

- read_img(img_path): sử dụng **Image.open()** để lấy ảnh về và **np.array(img)** để chuyển về ma trận.

- show_img(img): sử dụng hàm imshow và show của matplotlib

- save_img (img_2d, img_path): sử dụng hàm **fromarray** để chuyển từ ma trận sang ảnh, sau đó dùng hàm save để lưu ảnh.

14. Tài liệu tham khảo:

1. Kernel (image processing) - Wikipedia
2. Sách Deep Learning cơ bản - v2.pdf - Google Drive
3. Công thức về phép quay hay nhất | Toán lớp 11 (vietjack.com)

-HẾT-