TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

MÔN HỌC: TOÁN ỨNG DỤNG VÀ THỐNG KÊ

ĐỒ ÁN 2: IMAGE PROCESSING

Giáo viên hướng dẫn

Thầy Nguyễn Văn Quang Huy

Thầy Trần Hà Sơn

Thầy Nguyễn Đình Thúc

Thầy Nguyễn Ngọc Toàn

Sinh viên thực hiện

22127374

Lê Thanh Tâm

Thành phố Hồ Chí Minh, tháng 7 năm 2024

MỤC LỤC

I. TÓM TẮT ĐỒ ÁN **2**
II. CÁC HÀM CHỨC NĂNG **3**
1. Các hàm xử lý ảnh: **3**
1.1. Hàm đọc ảnh def read_img(img_path): 3
1.2. Hàm xuất ảnh def show_img(img): 3
1.3. Hàm lưu ảnh def save_img(img, path_file): 3
2. Các hàm biến đổi ảnh: **3**
2.1. Hàm thay đổi độ sáng của ảnh def brightness(img, alpha): 3
2.2. Hàm thay đổi độ tương phản của ảnh def contrast(img, beta): 4
2.3. Hàm lật ảnh theo chiều ngang def flip_horizontal_img(img): 5
2.4. Hàm lật ảnh theo chiều dọc def flip_vertical_img(img): 5
2.5. Hàm chuyển đổi ảnh RGB thành ảnh xám def grayscale_img(img): 6
2.6. Hàm chuyển đổi ảnh RGB thành ảnh sepia def sepia_img(img): 6
2.7. Hàm áp dụng hiệu ứng mờ cho ảnh def blur_img(img, alpha): 7
2.8. Hàm áp dụng hiệu ứng mờ cho ảnh def gaussian_blur_3x3_img(img): 8
2.9. Hàm áp dụng hiệu ứng mờ cho ảnh def gaussian_blur_5x5_img(img): 9
2.10. Hàm áp dụng hiệu ứng sắc nét cho ảnh def sharpen_img(img): 9
2.11. Hàm cắt ảnh theo khung hình tròn def crop_circle_img(img): 10
2.12. Hàm cắt ảnh theo khung hình elip def crop_ellipse_img(img): 11
2.13. Hàm phóng to ảnh def zoom_in_img(img, zoom_factor): 12
2.14. Hàm thu nhỏ ảnh def zoom_out_img(img, zoom_factor): 14
3. Các hàm phụ/hỗ trợ: **15**
3.1. Hàm áp dụng kernel lên ảnh def apply_kernel(img, kernel): 15
3.2. Hàm cắt ảnh ở trung tâm ảnh def crop_center_img(img, new_height, new_width): 16
3.3. Hàm tạo mặt nạ hình elip def create_ellipse_mask(shape, center, axes, angle): 16
III. BẢNG ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH VÀ HÌNH ẢNH KẾT QUẢ CHO TỪNG CHỨC NĂNG **18**
IV. THAM KHẢO **21**

I. TÓM TẮT ĐỒ ÁN

Đồ án này tập trung vào việc phát triển một bộ các hàm xử lý ảnh cơ bản sử dụng thư viện NumPy trong Python, nhằm thực hiện các phép biến đổi hình học và xử lý ảnh như phóng to, thu nhỏ, cắt ảnh theo hình tròn và hình elip, cũng như áp dụng các bộ lọc làm mờ và làm sắc nét. Các hàm này được thiết kế để hoạt động hiệu quả với ảnh dưới dạng mảng NumPy, không phụ thuộc vào các thư viện xử lý ảnh phổ biến khác như OpenCV hoặc PIL. Các hàm phóng to và thu nhỏ hình ảnh giúp thay đổi kích thước của ảnh một cách linh hoạt, trong khi các hàm cắt ảnh theo hình tròn và hình elip cho phép tạo ra các vùng ảnh đặc biệt với các hình dạng cụ thể, ví dụ như tạo avatar hoặc làm nổi bật các phần quan trọng của ảnh. Bên cạnh đó, các hàm áp dụng bộ lọc làm mờ và làm sắc nét giúp cải thiện chất lượng ảnh, làm mờ các chi tiết không mong muốn hoặc làm sắc nét các chi tiết cần thiết. Mỗi hàm được xây dựng với mục tiêu rõ ràng và cụ thể, giúp tăng cường khả năng xử lý và phân tích ảnh trong các ứng dụng thực tế như tạo thumbnail, chuẩn bị ảnh cho các thuật toán nhận diện khuôn mặt, và tạo hiệu ứng đặc biệt trên ảnh. Đồ án này không chỉ mang lại kiến thức về xử lý ảnh cơ bản mà còn cung cấp các công cụ thực tiễn cho việc phát triển các ứng dụng xử lý ảnh nâng cao.

II. CÁC HÀM CHỨC NĂNG

1. Các hàm xử lý ảnh:

1.1. Hàm đọc ảnh *def read_img(img_path)*:

Đây là hàm đọc ảnh từ đường dẫn cung cấp và trả về mảng NumPy biểu diễn dữ liệu của ảnh.

1.2. Hàm xuất ảnh *def show_img(img):*

Đây là hàm có nhiệm vụ hiển thị một hình ảnh từ một mảng dữ liệu ảnh. Ta sẽ hiển thị ảnh bằng cách sử dụng **plt.imshow(img)** trong thư viện matplotlib.

1.3. Hàm lưu ảnh *def save_img(img, path_file):*

Đây là hàm có nhiệm vụ lưu một hình ảnh vào đường dẫn được cung cấp. Hàm sẽ lưu hình ảnh vào đường dẫn được chỉ định. Trước hết ta cần sử dụng ***Image.fromarray(img)*** để chuyển đổi mảng dữ liệu ảnh thành một đối tượng ảnh của thư viện **PIL và sau đó dùng** img.save(img_path) để lưu đối tượng ảnh vào đường dẫn ***path_file.***

2. Các hàm biến đổi ảnh:

2.1. Hàm thay đổi độ sáng của ảnh *def brightness(img, alpha):*

- **Ý tưởng thực hiện:**

  Đọc ảnh: Đảm bảo rằng ảnh đầu vào (img) là một mảng numpy hoặc một đối tượng ảnh mà bạn có thể thao tác.

  Điều chỉnh độ sáng: Sử dụng hệ số ***alpha*** để điều chỉnh độ sáng của ảnh. alpha là một giá trị số thực, trong đó:

  - **alpha > 0** sẽ làm **sáng** ảnh.
  - **alpha < 0** sẽ làm **tối** ảnh.

  Giới hạn giá trị pixel: Đảm bảo rằng giá trị pixel sau khi điều chỉnh nằm trong khoảng [0, 255].

- **Mô tả chức năng:**

  Hàm **brightness_img(img, alpha)** điều chỉnh độ sáng của một ảnh bằng cách cộng thêm một giá trị alpha vào mỗi pixel của ảnh. Sau đó, giá trị pixel được giới hạn trong khoảng [0, 255] để đảm bảo rằng không có giá trị pixel nào vượt quá giới hạn này. Cuối cùng, hàm trả về ảnh đã được điều chỉnh dưới dạng mảng numpy với kiểu dữ liệu **uint8**.

- **Cài đặt:**
    - ***img = img + alpha:*** Cộng thêm giá trị alpha vào mỗi pixel của ảnh.
    - ***np.clip(img, 0, 255):*** Giới hạn giá trị pixel trong khoảng [0, 255].
    - ***img.astype(np.uint8):*** Chuyển đổi kiểu dữ liệu của ảnh thành **uint8** để đảm bảo rằng ảnh có thể được hiển thị đúng cách.

2.2. Hàm thay đổi độ tương phản của ảnh *def contrast(img, beta):*

- **Ý tưởng thực hiện:**

  Đọc ảnh: Đảm bảo rằng ảnh đầu vào (img) là một mảng numpy hoặc một đối tượng ảnh mà bạn có thể thao tác.

  Điều chỉnh độ tương phản: Sử dụng hệ số beta để điều chỉnh độ tương phản của ảnh. beta là một giá trị số thực, trong đó:

  - beta = 1 giữ nguyên độ tương phản của ảnh.
  - 0 <= beta < 1 giảm độ tương phản của ảnh.
  - > 1 tăng độ tương phản của ảnh.

  Điều chỉnh giá trị pixel: Mỗi pixel trong ảnh được nhân với hệ số beta để điều chỉnh độ tương phản.

  Giới hạn giá trị pixel: Đảm bảo rằng giá trị pixel sau khi điều chỉnh nằm trong khoảng [0, 255].

  Trả về ảnh đã điều chỉnh: Trả về ảnh sau khi đã điều chỉnh độ tương phản.

- **Mô tả chức năng:**

  Hàm **contrast_img(img, beta)** điều chỉnh độ tương phản của một ảnh bằng cách sử dụng hệ số beta. Mỗi pixel trong ảnh được nhân với hệ số beta để điều chỉnh độ tương phản. Giá trị pixel sau khi điều chỉnh được giới hạn trong khoảng [0, 255] để đảm bảo rằng không có giá trị pixel nào vượt quá giới hạn này. Cuối cùng, hàm trả về ảnh đã được điều chỉnh dưới dạng mảng numpy với kiểu dữ liệu **uint8**.

- **Cài đặt:**
    - **img * beta:** Nhân các giá trị pixel của ảnh với hệ số beta để điều chỉnh độ tương phản.
    - **np.clip(img, 0, 255):** Giới hạn giá trị pixel trong khoảng [0, 255] để đảm bảo giá trị hợp lệ.
    - **img.astype(np.uint8):** Chuyển đổi kiểu dữ liệu của ảnh đã điều chỉnh thành uint8 để đảm bảo rằng ảnh có thể được hiển thị đúng cách.

2.3. Hàm lật ảnh theo chiều ngang *def flip_horizontal_img(img):*

- **Ý tưởng thực hiện:**

  Đọc ảnh: Đảm bảo rằng ảnh đầu vào (img) là một mảng numpy hoặc một đối tượng ảnh mà bạn có thể thao tác.

  Lật ảnh theo chiều ngang: Sử dụng hàm np.flip của numpy để lật ảnh theo chiều ngang. Cụ thể, sử dụng **axis=1** để lật ảnh theo trục ngang, nghĩa là các cột của ma trận sẽ đảo ngược với nhau.

  Trả về ảnh đã lật: Trả về ảnh sau khi đã lật theo chiều ngang.

- **Mô tả chức năng:**

  Hàm **flip_horizontal_img(img)** lật một ảnh theo chiều ngang. Hàm sử dụng hàm np.flip của numpy với tham số **axis=1** để lật ảnh theo trục ngang. Kết quả là ảnh đầu vào sẽ được lật ngược lại theo chiều ngang và trả về dưới dạng mảng numpy.

- **Cài đặt:**
  - **np.flip(img, axis=1):** Lật ảnh theo chiều ngang bằng cách sử dụng hàm np.flip với **axis=1**, nghĩa là lật ảnh theo trục ngang.
  - Hàm trả về ảnh đã được lật dưới dạng mảng numpy.

2.4. Hàm lật ảnh theo chiều dọc *def flip_vertical_img(img):*

- **Ý tưởng thực hiện:**

  Đọc ảnh: Đảm bảo rằng ảnh đầu vào (img) là một mảng numpy hoặc một đối tượng ảnh mà bạn có thể thao tác.

  Lật ảnh theo chiều dọc: Sử dụng hàm np.flip của numpy để lật ảnh theo chiều dọc. Cụ thể, sử dụng **axis=0** để lật ảnh theo trục dọc, nghĩa là các dòng của ma trận sẽ thay đổi vị trí.

  Trả về ảnh đã lật: Trả về ảnh sau khi đã lật theo chiều dọc.

- **Mô tả chức năng:**

  Hàm **flip_vertical_img(img)** lật một ảnh theo chiều dọc. Hàm sử dụng hàm np.flip của numpy với tham số **axis=0** để lật ảnh theo trục dọc. Kết quả là ảnh đầu vào sẽ được lật ngược lại theo chiều dọc và trả về dưới dạng mảng numpy.

- **Cài đặt:**
  - **np.flip(img, axis=0):** Lật ảnh theo chiều dọc bằng cách sử dụng hàm np.flip với **axis=0**, nghĩa là lật ảnh theo trục dọc.
  - Hàm trả về ảnh đã được lật dưới dạng mảng numpy.

2.5. Hàm chuyển đổi ảnh RGB thành ảnh xám *def grayscale_img(img):*

- **Ý tưởng thực hiện:**

  Đọc ảnh: Đảm bảo rằng ảnh đầu vào (img) là một mảng numpy hoặc một đối tượng ảnh mà bạn có thể thao tác.

  Kiểm tra kênh alpha: Kiểm tra xem ảnh có kênh alpha hay không (tức là ảnh có 4 kênh màu: R, G, B, A).

  Loại bỏ kênh alpha: Nếu ảnh có kênh alpha, loại bỏ kênh này để chỉ giữ lại các kênh màu R, G, B.

  Chuyển đổi sang grayscale: Sử dụng công thức chuyển đổi từ RGB sang grayscale bằng cách tính tổng có trọng số của các kênh màu R, G, B. Trọng số được sử dụng là **[0.299, 0.587, 0.114]**[5]**.**

  Trả về ảnh grayscale: Trả về ảnh đã chuyển đổi sang grayscale dưới dạng mảng numpy với kiểu dữ liệu **uint8**.

- **Mô tả chức năng:**

  Hàm **grayscale_img(img)** chuyển đổi một ảnh màu sang ảnh grayscale. Nếu ảnh đầu vào có kênh alpha, hàm sẽ loại bỏ kênh này trước khi thực hiện chuyển đổi. Sau đó, hàm sử dụng công thức chuyển đổi từ RGB sang grayscale bằng cách tính tổng có trọng số của các kênh màu R, G, B. Kết quả là ảnh grayscale được trả về dưới dạng mảng numpy với kiểu dữ liệu **uint8.**

- **Cài đặt:**
  - **if img.shape[-1] == 4:** Kiểm tra xem ảnh có kênh alpha hay không.
  - **img = img[..., :3]:** Loại bỏ kênh alpha nếu có.
  - **gray = np.dot(img[..., :3], [0.299, 0.587, 0.114]):** Sử dụng công thức chuyển đổi từ RGB sang grayscale.
  - **gray.astype(np.uint8):** Chuyển đổi kiểu dữ liệu của ảnh grayscale thành uint8 để đảm bảo rằng ảnh có thể được hiển thị đúng cách.

2.6. Hàm chuyển đổi ảnh RGB thành ảnh sepia *def sepia_img(img):*

- **Ý tưởng thực hiện:**

  Đọc ảnh: Đảm bảo rằng ảnh đầu vào (img) là một mảng numpy hoặc một đối tượng ảnh mà bạn có thể thao tác.

  Chuyển đổi sang grayscale: Sử dụng hàm grayscale_img(img) để chuyển đổi ảnh màu sang ảnh grayscale.

  Áp dụng bộ lọc sepia: Sử dụng công thức để chuyển đổi ảnh grayscale sang ảnh sepia. Công thức này bao gồm việc nhân các giá trị pixel của ảnh grayscale với các hệ số cụ thể để tạo ra ba kênh màu mới (R, G, B).

Giới hạn giá trị pixel: Đảm bảo rằng giá trị pixel sau khi chuyển đổi nằm trong khoảng [0, 255].

Trả về ảnh sepia: Trả về ảnh sau khi đã chuyển đổi sang sepia dưới dạng mảng numpy với kiểu dữ liệu uint8.

- **Mô tả chức năng:**

  Hàm **sepia_img(img)** chuyển đổi một ảnh màu sang ảnh sepia. Đầu tiên, hàm sử dụng hàm ***grayscale_img(img)*** để chuyển đổi ảnh màu sang grayscale. Sau đó, hàm áp dụng bộ lọc sepia bằng cách nhân các giá trị pixel của ảnh grayscale với các hệ số cụ thể để tạo ra ba kênh màu mới (R, G, B). Kết quả là ảnh sepia được trả về dưới dạng mảng numpy với kiểu dữ liệu **uint8**.

- **Cài đặt:**
  - **grayscale_img(img):** Chuyển đổi ảnh màu sang grayscale.
  - **np.stack([...], axis=-1):** Tạo ra ba kênh màu mới (R, G, B) bằng cách nhân các giá trị pixel của ảnh grayscale với các hệ số cụ thể, **axis = -1** là trục cuối cuối cùng trong chiều của mảng trả về.
  - **np.clip(img, 0, 255):** Giới hạn giá trị pixel trong khoảng [0, 255].
  - **img.astype(np.uint8):** Chuyển đổi kiểu dữ liệu của ảnh sepia thành uint8 để đảm bảo rằng ảnh có thể được hiển thị đúng cách.

2.7. Hàm áp dụng hiệu ứng mờ cho ảnh *def blur_img(img, alpha):*

- **Ý tưởng thực hiện[2]:**

  Điều chỉnh kích thước kernel: Dựa trên giá trị alpha, tính toán kích thước của kernel. Kích thước kernel càng lớn thì hiệu ứng làm mờ càng mạnh.

  Đảm bảo kích thước kernel là số lẻ: Nếu kích thước kernel là số chẵn, tăng thêm 1 để đảm bảo nó là số lẻ (điều này cần thiết cho việc áp dụng kernel).

  Tạo kernel: Tạo kernel làm mờ bằng cách chia đều giá trị cho tất cả các phần tử trong kernel.

  Áp dụng kernel: Sử dụng hàm apply_kernel để áp dụng kernel lên ảnh và tạo hiệu ứng làm mờ.

  Trả về ảnh đã làm mờ: Trả về ảnh đã được làm mờ dưới dạng **uint8.**

- **Mô tả chức năng:**

  Hàm **blur_img(img, alpha)** áp dụng hiệu ứng làm mờ lên một ảnh đầu vào. Mức độ làm mờ được điều chỉnh bởi tham số alpha, với giá trị từ **0 đến 1**. Giá trị alpha càng nhỏ thì hiệu ứng làm mờ càng mạnh.

- **Cài đặt:**
  - **kernel_size = int(3 / alpha):** Tính toán kích thước kernel dựa trên giá trị alpha. Giá trị alpha càng nhỏ thì kích thước kernel càng lớn.
  - **if kernel_size % 2 == 0: kernel_size += 1:** Đảm bảo kích thước kernel là số lẻ.
  - **kernel = np.ones((kernel_size, kernel_size)) / (kernel_size * kernel_size):** Tạo kernel làm mờ bằng cách chia đều giá trị cho tất cả các phần tử trong kernel.
  - **return apply_kernel(img, kernel).astype(np.uint8):** Áp dụng kernel lên ảnh và trả về ảnh đã được làm mờ dưới dạng uint8.

2.8. Hàm áp dụng hiệu ứng mờ cho ảnh *def gaussian_blur_3x3_img(img)*:

- **Ý tưởng thực hiện[2]:**

  Đọc ảnh: Đảm bảo rằng ảnh đầu vào (img) là một mảng numpy hoặc một đối tượng ảnh mà bạn có thể thao tác.

  Tạo kernel Gaussian blur: Tạo một kernel Gaussian 3x3 với các giá trị cụ thể để làm mờ ảnh. Kernel này có trọng số cao hơn ở trung tâm và giảm dần ra các cạnh, giúp làm mờ ảnh một cách mượt mà hơn.

  Áp dụng kernel: Sử dụng hàm apply_kernel để áp dụng kernel Gaussian lên ảnh, tạo hiệu ứng làm mờ.

  Trả về ảnh đã làm mờ: Trả về ảnh sau khi đã áp dụng hiệu ứng làm mờ dưới dạng mảng numpy với kiểu dữ liệu uint8.

- **Mô tả chức năng:**

  Hàm **gaussian_blur_3x3_img(img)** áp dụng hiệu ứng làm mờ Gaussian (Gaussian blur) lên một ảnh. Hàm nhận vào một ảnh và tạo một kernel Gaussian 3x3 với các giá trị cụ thể để làm mờ ảnh. Kernel này có trọng số cao hơn ở trung tâm và giảm dần ra các cạnh, giúp làm mờ ảnh một cách mượt mà hơn. Kết quả là ảnh đã được làm mờ được trả về dưới dạng mảng numpy với kiểu dữ liệu **uint8**.

- **Cài đặt:**
  - **kernel = np.array([[1, 2, 1], [2, 4, 2], [1, 2, 1]]) / 16.0:** Tạo kernel Gaussian 3x3 với các giá trị cụ thể và chuẩn hóa tổng các giá trị bằng 1.
  - **apply_kernel(img, kernel):** Áp dụng kernel Gaussian lên ảnh để tạo hiệu ứng làm mờ.
  - **astype(np.uint8):** Chuyển đổi kiểu dữ liệu của ảnh đã làm mờ thành uint8 để đảm bảo rằng ảnh có thể được hiển thị đúng cách.

2.9. Hàm áp dụng hiệu ứng mờ cho ảnh *def gaussian_blur_5x5_img(img):*

- **Ý tưởng thực hiện[2]:**

  Đọc ảnh: Đảm bảo rằng ảnh đầu vào (img) là một mảng numpy hoặc một đối tượng ảnh mà bạn có thể thao tác.

  Tạo kernel Gaussian blur: Tạo một kernel Gaussian 5x5 với các giá trị cụ thể để làm mờ ảnh. Kernel này có trọng số cao hơn ở trung tâm và giảm dần ra các cạnh, giúp làm mờ ảnh một cách mượt mà hơn.

  Áp dụng kernel: Sử dụng hàm **apply_kernel** để áp dụng kernel Gaussian lên ảnh, tạo hiệu ứng làm mờ.

  Trả về ảnh đã làm mờ: Trả về ảnh sau khi đã áp dụng hiệu ứng làm mờ dưới dạng mảng numpy với kiểu dữ liệu **uint8.**

- **Mô tả chức năng:**

  Hàm **gaussian_blur_5x5_img(img)** áp dụng hiệu ứng làm mờ Gaussian (Gaussian blur) lên một ảnh. Hàm nhận vào một ảnh và tạo một kernel Gaussian 5x5 với các giá trị cụ thể để làm mờ ảnh. Kernel này có trọng số cao hơn ở trung tâm và giảm dần ra các cạnh, giúp làm mờ ảnh một cách mượt mà hơn. Kết quả là ảnh đã được làm mờ được trả về dưới dạng mảng numpy với kiểu dữ liệu **uint8.**

- **Cài đặt:**
  - **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]]) / 256.0:** Tạo kernel Gaussian 5x5 với các giá trị cụ thể và chuẩn hóa tổng các giá trị bằng 1.
  - **apply_kernel(img, kernel):** Áp dụng kernel Gaussian lên ảnh để tạo hiệu ứng làm mờ.
  - **astype(np.uint8):** Chuyển đổi kiểu dữ liệu của ảnh đã làm mờ thành uint8 để đảm bảo rằng ảnh có thể được hiển thị đúng cách.

2.10. Hàm áp dụng hiệu ứng sắc nét cho ảnh *def sharpen_img(img):*

- **Ý tưởng thực hiện[2]:**

  Đọc ảnh: Đảm bảo rằng ảnh đầu vào (img) là một mảng numpy hoặc một đối tượng ảnh mà bạn có thể thao tác.

  Tạo kernel làm sắc nét: Tạo một kernel 3x3 với các giá trị cụ thể để làm sắc nét ảnh. Kernel này sẽ tăng cường các cạnh và chi tiết trong ảnh.

  Áp dụng kernel: Sử dụng hàm **apply_kernel** để áp dụng kernel làm sắc nét lên ảnh.

Đảm bảo giá trị pixel hợp lệ: Sử dụng np.clip để đảm bảo rằng các giá trị pixel nằm trong khoảng [0, 255].

Trả về ảnh đã làm sắc nét: Trả về ảnh sau khi đã áp dụng hiệu ứng làm sắc nét dưới dạng mảng numpy với kiểu dữ liệu **uint8.**

- **Mô tả chức năng:**

  Hàm **sharpen_img(img)** áp dụng hiệu ứng **làm sắc nét (sharpening)** lên một ảnh. Hàm nhận vào một ảnh và tạo một **kernel** 3x3 với các giá trị cụ thể để làm sắc nét ảnh. Kernel này sẽ tăng cường các cạnh và chi tiết trong ảnh. Sau khi áp dụng kernel, hàm đảm bảo rằng các giá trị pixel nằm trong khoảng [0, 255] và trả về ảnh đã được làm sắc nét dưới dạng mảng numpy với kiểu dữ liệu **uint8.**

- **Cài đặt:**
  - **kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]]):** Tạo kernel làm sắc nét 3x3 với các giá trị cụ thể để tăng cường các cạnh và chi tiết trong ảnh.
  - **apply_kernel(img, kernel):** Áp dụng kernel làm sắc nét lên ảnh.
  - **np.clip(sharpened, 0, 255):** Đảm bảo rằng các giá trị pixel nằm trong khoảng [0, 255].
  - **astype(np.uint8):** Chuyển đổi kiểu dữ liệu của ảnh đã làm sắc nét thành uint8 để đảm bảo rằng ảnh có thể được hiển thị đúng cách.

2.11. Hàm cắt ảnh theo khung hình tròn *def crop_circle_img(img):*

- **Ý tưởng thực hiện:**

  Đọc ảnh: Đảm bảo rằng ảnh đầu vào (img) là một mảng numpy hoặc một đối tượng ảnh mà bạn có thể thao tác.

  Lấy kích thước ảnh: Lấy chiều cao và chiều rộng của ảnh.

  Tạo mặt nạ hình tròn:

  - Xác định tọa độ trung tâm của ảnh.
  - Tính bán kính của hình tròn sao cho nó vừa với ảnh.
  - Tạo một lưới tọa độ và tính khoảng cách từ mỗi điểm đến trung tâm.
  - Tạo mặt nạ hình tròn dựa trên khoảng cách này.

  Áp dụng mặt nạ lên ảnh:

  - Nếu ảnh có 3 kênh màu (RGB), áp dụng mặt nạ lên từng kênh.
  - Nếu ảnh là ảnh xám (grayscale), áp dụng mặt nạ trực tiếp.

  Trả về ảnh đã được cắt hình tròn: Trả về ảnh sau khi đã áp dụng mặt nạ hình tròn.

- **Mô tả chức năng:**

  Hàm **crop_circle_img(img)** cắt một hình tròn ở trung tâm của một ảnh. Hàm nhận vào một ảnh và tạo một mặt nạ hình tròn với bán kính vừa với ảnh. Sau đó, hàm áp dụng mặt nạ này lên ảnh để giữ lại phần hình tròn ở trung tâm và loại bỏ phần còn lại. Kết quả là ảnh đã được cắt hình tròn được trả về dưới dạng mảng numpy.

- **Cài đặt:**
  - **height, width = img.shape[:2]:** Lấy chiều cao và chiều rộng của ảnh.
  - **center_y, center_x = height // 2, width // 2:** Xác định tọa độ trung tâm của ảnh.
  - **radius = min(center_y, center_x):** Tính bán kính của hình tròn sao cho nó vừa với ảnh.
  - **Y, X = np.ogrid[:height, :width]:** Tạo một lưới tọa độ.
  - **dist_from_center = np.sqrt((X - center_x)**2 + (Y - center_y)**2):** Tính khoảng cách từ mỗi điểm đến trung tâm.
  - **mask = dist_from_center <= radius:** Tạo mặt nạ hình tròn dựa trên khoảng cách này.
  - **circular_img[..., i] = img[..., i] * mask:** Áp dụng mặt nạ lên từng kênh màu (RGB).
  - **circular_img = img * mask:** Áp dụng mặt nạ trực tiếp nếu ảnh là ảnh xám (grayscale).
  - **astype(np.uint8):** Chuyển đổi kiểu dữ liệu của ảnh đã cắt thành uint8 để đảm bảo rằng ảnh có thể được hiển thị đúng cách.

2.12. Hàm cắt ảnh theo khung hình elip *def crop_ellipse_img(img):*

- **Ý tưởng thực hiện[1]:**

  Đọc ảnh: Đảm bảo rằng ảnh đầu vào (img) là một mảng numpy hoặc một đối tượng ảnh mà bạn có thể thao tác.

  Lấy kích thước ảnh: Lấy chiều cao và chiều rộng của ảnh.

  Tạo mặt nạ cho hai hình elip:

  - Xác định tọa độ trung tâm của ảnh.
  - Xác định trục chính và trục phụ của hai hình elip.
  - Xác định góc nghiêng của hai hình elip.
  - Sử dụng hàm **create_ellipse_mask** để tạo mặt nạ cho từng hình elip.
  - Kết hợp **hai** mặt nạ bằng phép OR.

  Áp dụng mặt nạ lên ảnh:

  - Nếu ảnh có 3 kênh màu (RGB), áp dụng mặt nạ lên từng kênh.

- Nếu ảnh là ảnh xám (grayscale), áp dụng mặt nạ trực tiếp.

Trả về ảnh đã được cắt hình elip: Trả về ảnh sau khi đã áp dụng mặt nạ hình elip.

- **Mô tả chức năng:**

  Hàm **crop_ellipse_img(img)** cắt hai hình elip ở trung tâm của một ảnh. Hàm nhận vào một ảnh và tạo hai mặt nạ hình elip với các trục và góc nghiêng khác nhau. Sau đó, hàm kết hợp hai mặt nạ này và áp dụng lên ảnh để giữ lại phần hình elip ở trung tâm và loại bỏ phần còn lại. Kết quả là ảnh đã được cắt hình elip được trả về dưới dạng mảng numpy.

- **Cài đặt:**
  - **create_ellipse_mask:** Hàm này tạo mặt nạ hình elip dựa trên kích thước, tọa độ trung tâm, trục và góc nghiêng.
  - **height, width = img.shape[:2]:** Lấy chiều cao và chiều rộng của ảnh.
  - **center = (width // 2, height // 2):** Xác định tọa độ trung tâm của ảnh.
  - **axes1 = (width // 2, height // 2), axes2 = (width // 2, height // 2):** Xác định trục chính và trục phụ của hai hình elip.
  - **angle1 = 30, angle2 = 60:** Xác định góc nghiêng của hai hình elip.
  - **mask1 = create_ellipse_mask((height, width), center, axes1, angle1), mask2 = create_ellipse_mask((height, width), center, axes2, angle2):** Tạo mặt nạ cho từng hình elip.
  - **combined_mask = mask1 | mask2:** Kết hợp hai mặt nạ bằng phép OR.
  - **elliptical_img[..., i] = img[..., i] * combined_mask:** Áp dụng mặt nạ lên từng kênh màu (RGB).
  - **elliptical_img = img * combined_mask:** Áp dụng mặt nạ trực tiếp nếu ảnh là ảnh xám (grayscale).

2.13. Hàm phóng to ảnh *def zoom_in_img(img, zoom_factor):*

- **Ý tưởng thực hiện[7]:**

  Đọc ảnh và xác định kích thước: Lấy chiều cao và chiều rộng của ảnh đầu vào.

  Tính toán kích thước mới: Dựa trên hệ số phóng to (zoom_factor), tính toán chiều cao và chiều rộng mới của ảnh.

  Tạo mảng lưu trữ ảnh phóng to: Khởi tạo một mảng numpy với kích thước mới để lưu trữ ảnh đã phóng to.

  Tạo các tọa độ gốc: Tạo các tọa độ gốc tương ứng cho ảnh đã phóng to bằng cách chia tỷ lệ.

  Tạo các tọa độ nguyên và trọng số nội suy:

- Tính toán các tọa độ nguyên (floor) và trần (ceil) cho các hàng và cột.
- Tính toán trọng số nội suy cho các hàng và cột.

Áp dụng nội suy bilinear:

- Lặp qua từng kênh màu của ảnh.
- Lấy giá trị của 4 điểm xung quanh cho mỗi pixel.
- Tính toán giá trị pixel của ảnh phóng to bằng cách sử dụng nội suy bilinear.

Trả về ảnh đã phóng to: Trả về ảnh đã phóng to sau khi đã áp dụng nội suy và giới hạn giá trị pixel trong khoảng [0, 255].

- **Mô tả chức năng:**

  Hàm **zoom_in_img(img, zoom_factor)** phóng to một ảnh bằng cách sử dụng nội suy bilinear với NumPy. Hàm nhận vào một ảnh và hệ số phóng to, sau đó tính toán kích thước mới của ảnh và áp dụng nội suy bilinear để tạo ra ảnh phóng to. Kết quả là một ảnh đã được phóng to với kích thước mới và giá trị pixel được giới hạn trong khoảng [0, 255].

- **Cài đặt:**
  - **height, width = img.shape[:2]:** Lấy chiều cao và chiều rộng của ảnh.
  - **new_height, new_width = int(height * zoom_factor), int(width * zoom_factor):** Tính toán chiều cao và chiều rộng mới dựa trên hệ số phóng to.
  - **zoomed_img = np.zeros((new_height, new_width, img.shape[2]), dty**pe=img.dtype): Tạo mảng để lưu trữ ảnh đã phóng to.
  - **row_scale = height / new_height, col_scale = width / new_width:** Tính toán tỷ lệ hàng và cột.
  - **rows = np.arange(new_height) * row_scale, cols = np.arange(new_width) * col_scale:** Tạo các tọa độ gốc tương ứng cho ảnh đã phóng to.
  - **rows_floor = np.floor(rows).astype(int), cols_floor = np.floor(cols).astype(int):** Tạo các tọa độ nguyên.
  - **rows_ceil = np.minimum(rows_floor + 1, height - 1), cols_ceil = np.minimum(cols_floor + 1, width - 1):** Tạo các tọa độ trần.
  - **row_weights = rows - rows_floor, col_weights = cols - cols_floor:** Tính toán trọng số nội suy.
  - **Q11, Q12, Q21, Q22:** Lấy giá trị của 4 điểm xung quanh cho mỗi pixel.
  - **zoomed_img[:, :, k] = ...:** Tính toán giá trị pixel của ảnh phóng to **bằng cách sử dụng nội suy bilinear.**

- **return np.clip(zoomed_img, 0, 255).astype(np.uint8):** Trả về ảnh đã phóng to sau khi đã áp dụng nội suy và giới hạn giá trị pixel trong khoảng [0, 255].

2.14. Hàm thu nhỏ ảnh *def zoom_out_img(img, zoom_factor):*

- **Ý tưởng thực hiện[7]:**

  Đọc ảnh và xác định kích thước: Lấy chiều cao và chiều rộng của ảnh đầu vào.

  Tính toán kích thước mới: Dựa trên hệ số thu nhỏ (zoom_factor), tính toán chiều cao và chiều rộng mới của ảnh.

  Tạo mảng lưu trữ ảnh thu nhỏ: Khởi tạo một mảng numpy với kích thước mới để lưu trữ ảnh đã thu nhỏ.

  Lặp qua từng pixel của ảnh mới:

  - Tìm tọa độ tương ứng trong ảnh gốc cho mỗi pixel của ảnh mới.
  - Trích xuất khối pixel từ ảnh gốc.
  - Tính toán giá trị trung bình của khối pixel cho mỗi kênh màu.

  Trả về ảnh đã thu nhỏ: Trả về ảnh đã thu nhỏ sau khi đã tính toán giá trị trung bình cho mỗi pixel.

- **Mô tả chức năng:**

  Hàm **zoom_out_img(img, zoom_factor)** thu nhỏ một ảnh bằng cách tính toán giá trị trung bình của các khối pixel trong ảnh gốc. Hàm nhận vào một ảnh và hệ số thu nhỏ, sau đó tính toán kích thước mới của ảnh và áp dụng phương pháp tính trung bình để tạo ra ảnh thu nhỏ. Kết quả là một ảnh đã được thu nhỏ với kích thước mới.

- **Cài đặt:**
  - **height, width = img.shape[:2]:** Lấy chiều cao và chiều rộng của ảnh.
  - **new_height, new_width = int(height / zoom_factor), int(width / zoom_factor):** Tính toán chiều cao và chiều rộng mới dựa trên hệ số thu nhỏ.
  - **zoomed_out_img = np.zeros((new_height, new_width, img.shape[2]), dtype=img.dtype):** Tạo mảng để lưu trữ ảnh đã thu nhỏ.
  - **for i in range(new_height), for j in range(new_width):** Lặp qua từng pixel của ảnh mới.
  - **orig_i_start = int(i * zoom_factor), orig_i_end = int((i + 1) * zoom_factor), orig_j_start = int(j * zoom_factor), orig_j_end = int((j + 1) * zoom_factor):** Tìm tọa độ tương ứng trong ảnh gốc.

- **block = img[orig_i_start:orig_i_end, orig_j_start:orig_j_end]:** Trích xuất khối pixel từ ảnh gốc.
- **zoomed_out_img[i, j, k] = np.mean(block[:, :, k]):** Tính toán giá trị trung bình của khối pixel cho mỗi kênh màu.
- **return zoomed_out_img:** Trả về ảnh đã thu nhỏ.

3. Các hàm phụ/hỗ trợ:

3.1. Hàm áp dụng kernel lên ảnh *def apply_kernel(img, kernel):*

- **Ý tưởng thực hiện[2]:**

  Lấy kích thước ảnh và kernel: Xác định chiều cao và chiều rộng của ảnh và kernel.

  Tạo ảnh kết quả: Khởi tạo một mảng numpy với cùng kích thước như ảnh đầu vào để lưu trữ kết quả sau khi áp dụng kernel.

  Pad ảnh: Thêm padding vào ảnh để xử lý các pixel biên, sử dụng phương pháp phản chiếu (reflect).

  Áp dụng kernel:

  - Lặp qua từng pixel của ảnh.
  - Lặp qua từng kênh màu của ảnh.
  - Trích xuất vùng ảnh con tương ứng với kích thước của kernel.
  - Tính tổng tích chập giữa vùng ảnh con và kernel.

  Trả về ảnh kết quả: Trả về ảnh sau khi đã áp dụng kernel.

- **Mô tả chức năng:**

  Hàm ***apply_kernel(img, kernel)*** áp dụng một **kernel (ma trận lọc)** lên một ảnh. Hàm này thực hiện **phép tích chập (convolution)** giữa kernel và từng vùng con của ảnh để tạo ra một ảnh mới. Kết quả là một ảnh đã được lọc theo kernel, có thể được sử dụng cho các tác vụ như làm mờ, làm sắc nét, hoặc phát hiện biên.

- **Cài đặt:**
  - **height, width = img.shape[:2]:** Lấy chiều cao và chiều rộng của ảnh.
  - **k_height, k_width = kernel.shape:** Lấy chiều cao và chiều rộng của kernel.
  - **result = np.zeros_like(img).astype(np.float32):** Tạo mảng để lưu trữ ảnh kết quả.
  - **pad_height = k_height // 2, pad_width = k_width // 2:** Tính toán kích thước padding.
  - **padded_img = np.pad(img, ((pad_height, pad_height), (pad_width, pad_width), (0, 0)), mode='reflect'):** Thêm padding vào ảnh.

- **for i in range(height), for j in range(width), for k in range(img.shape[2]):** Lặp qua từng pixel và từng kênh màu của ảnh.
- **region = padded_img[i:i + k_height, j:j + k_width, k]:** Trích xuất vùng ảnh con tương ứng.
- **result[i, j, k] = np.sum(region * kernel):** Tính tổng tích chập giữa vùng ảnh con và kernel.
- **return result:** Trả về ảnh kết quả.

3.2. Hàm cắt ảnh ở trung tâm ảnh *def crop_center_img(img, new_height, new_width):*

- **Ý tưởng thực hiện[6]:**

  Lấy kích thước ảnh: Xác định chiều cao và chiều rộng của ảnh đầu vào.

  Tính toán tọa độ bắt đầu: Tính toán tọa độ bắt đầu của vùng cắt ở trung tâm ảnh dựa trên chiều cao và chiều rộng mới.

  Cắt ảnh: Sử dụng tọa độ bắt đầu và kích thước mới để cắt vùng trung tâm của ảnh.

  Trả về ảnh đã cắt: Trả về vùng ảnh đã cắt.

- **Mô tả chức năng:**

  Hàm **crop_center_img(img, new_height, new_width)** cắt một vùng ảnh từ trung tâm của ảnh đầu vào với kích thước xác định bởi **new_height** và **new_width**. Hàm này hữu ích khi cần lấy một phần trung tâm của ảnh để xử lý hoặc hiển thị.

3.3. Hàm tạo mặt nạ hình elip *def create_ellipse_mask(shape, center, axes, angle):*

- **Ý tưởng thực hiện[1]:**

  Chuyển đổi góc: Chuyển đổi góc nghiêng từ độ sang radian.

  Tạo lưới tọa độ: Tạo lưới tọa độ cho ảnh với kích thước xác định bởi shape.

  Tính toán tọa độ tương đối: Tính toán tọa độ tương đối của các điểm ảnh so với tâm của elip.

  Tính toán các giá trị cosine và sine: Tính toán các giá trị cosine và sine của góc nghiêng để sử dụng trong phương trình elip.

  Tạo mặt nạ elip: Sử dụng phương trình elip để xác định các điểm nằm trong elip và tạo mặt nạ.

  Trả về mặt nạ: Trả về mặt nạ hình elip.

- **Mô tả chức năng:**

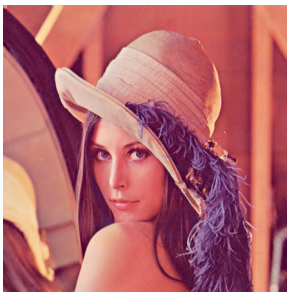
  Hàm **create_ellipse_mask(shape, center, axes, angle)** tạo một mặt nạ hình elip trên một ảnh có kích thước xác định. Mặt nạ này có thể được sử dụng để chọn các vùng cụ thể của ảnh hoặc để áp dụng các phép biến đổi chỉ trên vùng elip.

- **Cài đặt:**
  - **a, b = axes:** Lấy độ dài trục chính và trục phụ của elip.
  - **theta = np.radians(angle):** Chuyển đổi góc nghiêng từ độ sang radian.
  - **Y, X = np.ogrid[:shape[0], :shape[1]]:** Tạo lưới tọa độ cho ảnh.
  - **xc, yc = center:** Lấy tọa độ tâm của elip.
  - **x = X - xc, y = Y - yc:** Tính toán tọa độ tương đối của các điểm ảnh so với tâm của elip.
  - **cos_theta = np.cos(theta), sin_theta = np.sin(theta):** Tính toán các giá trị cosine và sine của góc nghiêng.
  - **ellipse_mask = (((x * cos_theta + y * sin_theta) ** 2) / a ** 2 + ((x * sin_theta - y * cos_theta) ** 2) / b ** 2) <= 1:** Sử dụng phương trình elip để tạo mặt nạ.
  - **return ellipse_mask:** Trả về mặt nạ hình elip.

$$\frac{x^2}{a^2 \sin^2(2\theta)} + \frac{y^2}{b^2 \sin^2(2\theta)} + \frac{2\cos(2\theta)xy}{ab \sin^2(2\theta)} = 1$$

Với $0 < \theta < \pi/2$

III. BẢNG ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH VÀ HÌNH ẢNH KẾT QUẢ CHO TỪNG CHỨC NĂNG

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%	

IV. THAM KHẢO

[1] Ellipse in a rectangle - Mathematics Stack Exchange.

conic sections - Ellipse in a rectangle - Mathematics Stack Exchange

Ngày truy cập 25/07/2024

[2] Kernel, Convolution matrix, or Mask - for **Blurring, Sharpening.**

https://en.wikipedia.org/wiki/Kernel_(image_processing)

Ngày truy cập 20/07/2024

[3] Use **Numpy.flip()** to reverses the order of array elements along the specified axis

https://www.geeksforgeeks.org/numpy-flip-python/

Ngày truy cập 20/07/2024

[4] Use np.clip() to make the pixels in [0,255]

Numpy Clip | How to Use np.clip() Function in Python - Python Pool

Ngày truy cập 20/07/2024

[5] Equation for Converting RGB to Grayscale

RGB to Grayscale Conversion Calculator | Good Calculators

Ngày truy cập 20/07/2024

[6] Crop center of image

https://stackoverflow.com/questions/16646183/crop-an-image-in-the-centre-using-pil

Ngày truy cập 25/07/2024

[7] Resize Image - Zoom in/Zoom out

python - Numpy Resize/Rescale Image - Stack Overflow

Ngày truy cập 25/07/2024