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 2

Image Processing

Môn học: Toán ứng dụng và thống kê cho CNTT

MTH00051_22CLC09

Sinh viên:
Nguyễn Hồ Đăng Duy
22127085
22CLC09

Giảng viên hướng dẫn:
Nguyễn Ngọc Toàn
Nguyễn Văn Quang Huy

Thành phố Hồ Chí Minh, tháng 7 năm 2024

Mục lục

1 Thông tin sinh viên

- Họ và tên: Nguyễn Hồ Đăng Duy
- MSSV: 22127085
- Lớp: 22CLC09

2 Ý tưởng thực hiện

2.1 Thay đổi độ sáng

Để thay đổi độ sáng của ảnh, hàm *adjust_ brightness* sẽ cộng giá trị **brightness** (có thể là số dương hoặc số âm) vào từng phần tử của mảng ảnh. Sau đó, hàm sẽ đảm bảo rằng các giá trị pixel không vượt ra khỏi phạm vi từ 0 đến 255 (vì đây là phạm vi giá trị hợp lệ cho pixel trong không gian màu 8-bit). Kết quả cuối cùng là một mảng hình ảnh đã được điều chỉnh độ sáng.

2.2 Thay đổi độ tương phản

- Hàm adjust_contrast được sử dụng để điều chỉnh độ tương phản của một hình ảnh. Hàm sẽ sử dụng một giá trị **mean** là giá trị trung bình của toán bộ các pixel trong hình ảnh để làm tham chiếu điều chỉnh độ tương phản.
- Sau đó sử dụng công thức để điều chỉnh độ tương phản sao cho cách biệt so với giá trị trung bình được tăng hoặc giảm tùy thuộc vào hệ số **contrast** (và không được vượt ra khỏi phạm vi từ 0 đến 255).
- Băng cách điều chỉnh giá trị pixel tương đối với trung bình của ảnh, ta đảm bảo được rằng các thay đổi sẽ được phân bổ đồng đều hơn giữa các vùng tối và vùng sáng. Hiệu quả hơn việc nhân trực tiếp **image*contrast** vì cách này có thể dẫn đến cắt giá trị và mất chi tiết và không thực sự mô phỏng chính xác sự thay đổi độ tương phản.

2.3 Lật ảnh ngang

Các pixel của hình ảnh sẽ được đảo ngược từ trái sang phải, tương tự như việc xoay một tấm gương ngang qua giữa hình ảnh.

2.4 Lật ảnh dọc

Đối với một hình ảnh, lật dọc có nghĩa là chuyển đổi thứ tự của các hàng từ trên xuống dưới. Điều này có thể được thực hiện bằng cách đảo ngược thứ tự các hàng của mảng đại diện cho hình ảnh.

2.5 RGB thành ảnh xám

- Ảnh RGB sẽ có 3 kênh màu (đỏ, xanh lục và xanh lam) trong khi ảnh xám chỉ có một kênh duy nhất để thể hiện mức độ sáng của mỗi pixel.
- Các kênh màu đỏ, xanh lục và xanh lam có mức độ ảnh hưởng khác nhau đến độ sáng tổng thể của ảnh. Cụ thể, kênh xanh lục (Green) thường ảnh hưởng lớn nhất đến độ sáng nhận thức của con người. Do đó công thức chuyển đổi phổ biến là sử dụng trọng số cho mỗi kênh: [1]
  - Red (R): 0.3
  - Green (G): 0.59
  - Blue (B): 0.11
- Sử dụng trọng số để kết hợp ba kênh màu thành một giá trị độ xám, phản ánh độ sáng tổng thể của pixel trong ảnh gốc.

2.6 RGB thành ảnh sepia

Để chuyển đổi một hình ảnh từ không gian màu RGB sang ảnh có tông màu sepia, ta áp dụng một công thức tuyến tính để phối hợp lại các kênh màu đỏ, xanh lục và xanh lam, nhằm tạo ra một hình ảnh với sắc thái nâu đỏ đặc trưng. [2]

2.7 Làm mờ ảnh

Để áp dụng hiệu ứng làm mờ (blur), ta có thể dùng phép tích chập (convolution) với một kernel làm mờ. Hiệu ứng này giúp làm giảm chi tiết và độ sắc nét của hình ảnh. [3]

2.7.1 Nguyên tắc làm mờ:

- Kernel: Kernel là một ma trận nhỏ được sử dụng để tính toán trung bình của các giá trị pixel xung quanh. Mỗi giá trị trong kernel đóng vai trò như một trọng số để lấy trung bình của một vùng lân cận trong ảnh.
- Thêm padding: Giúp đảm bảo rằng các pixel ở viền của ảnh cũng được xử lý một cách hợp lý mà không làm mất thông tin.
- Phép tích chập: Phép tích chập (convolution) là quá trình áp dụng kernel lên từng vùng của hình ảnh để tạo ra hiệu ứng làm mờ. Quá trình này được thực hiện bằng cách di chuyển kernel quablur từng pixel của hình ảnh và tính toán trung bình có trọng số của các giá trị pixel xung quanh.

2.7.2 Lí do chọn box blur:

- **Đơn giản và hiệu quả:** Box blur sử dụng một kernel trung bình (ma trận với tất cả các giá trị bằng nhau) để tính trung bình cộng của các pixel lân cận. Do đó có độ phức tạp tính toán thấp hơn so với một số phương pháp làm mờ khác như Gaussian blur, giúp tiết kiệm tài nguyên và thời gian xử lý.

- **Làm mờ đồng đều:** Box blur tạo ra hiệu ứng làm mờ đồng nhất trên toàn bộ ảnh. Mỗi pixel trong vùng lân cận đóng góp một phần bằng nhau vào giá trị pixel mới, dẫn đến một hiệu ứng làm mờ mềm mại, làm giảm các chi tiết sắc nét và làm giảm nhiễu.
- **Khả năng điều chỉnh linh hoạt:** Có thể thay đổi kích thước kernel dễ dàng trong Box blur, cho phép điều chỉnh mức độ làm mờ bằng cách thay đổi kích thước của ma trận kernel.

2.8 Làm sắc nét ảnh

Tương tự như làm mờ hình ảnh, làm sắc nét hình ảnh cũng sử dụng một **kernel làm nét** để tăng cường độ tương phản giữa các pixel lân cận, giúp hình ảnh trở nên sắc nét hơn. Kernel làm nét thực hiện điều này bằng cách làm nổi bật các điểm thay đổi cường độ mạnh mẽ trong ảnh, thường xuất hiện tại các đường biên hoặc các chi tiết nhỏ.

2.9 Cắt ảnh theo kích thước

Để có thể cắt ảnh tại trung tâm theo kích thước **size** nhập từ bàn phím, ta cần xác định được tọa độ của các điểm **left, top, right, bottom** của hình ban đầu và cắt hình mới dựa trên các tọa độ đódó

2.10 Cắt ảnh theo khung tròn

Để cắt ảnh theo khung tròn, trước tiên cần thiết kế một mặt nạ (mask) hình tròn. Ảnh kết quả sẽ chỉ giữ lại các pixel nằm trong vùng hình tròn với tâm là trung tâm của ảnh, các pixel nằm ngoài vùng này sẽ được đặt thành 0 (màu đen).

$$mask = (x - center\_x)^2 + (y - center\_y)^2 \leqslant (row/2)^2$$

2.11 Cắt ảnh theo khung elip

2.11.1 Ý tưởng

Hàm cắt ảnh theo khung elip được thiết kế để cắt ảnh dựa trên mask là 2 hình dạng elip giao nhau có cùng tâm nhưng có hướng khác nhau. Ảnh kết quả chỉ giữ lại các pixel nằm trong vùng 2 hình elip chéo nhau, có tâm là trung tâm của ảnh, các pixel nằm ngoài vùng này sẽ được đặt thành 0 (màu đen).

2.11.2 Xây dựng công thức mask

Phương trình tổng quát: Một elip trong hệ tọa độ Cartesian với tâm tại điểm (h,k) và có trục chính song song với trục tọa độ có phương trình là:

$$\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1$$

Trong đó:

- a là độ dài trục chính.

- b là độ dài trục phụ.
- (x,y) là tọa độ của một điểm trên elip.
- (h,k) là tọa độ của tâm elip.

Một số công thức tịnh tiến và quay được sử dụng: Để tìm công thức elip nằm chéo trong hình vuông, ta dùng các phép biến đổi hệ tọa độ sau:

- Phép quay tâm O, góc quay α trong hệ tọa độ Oxy biến điểm M(x;y) thành điểm M'(x';y') [6]. Khi đó $\begin{cases}$
- Phép tịnh tiến: Trong mặt phẳng tọa độ Oxy cho vector $\vec{v} = (a; b)$. Với mỗi điểm M(x;y) ta có M'(x';y') là ảnh của M qua phép tịnh tiến theo $\vec{v}$. [7] Khi đó $M\vec{M}' = \vec{v} \Leftrightarrow \begin{cases} x' = x + a \\ y' = y + b \end{cases}$

Ứng dụng trong phương trình của mask: Áp dụng cả 2 công thức đã nêu trên vào phương trình hình elip, trong đó:

- Tọa độ tâm của hình là **(center_x, center_y)** nên phương trình elip gốc sẽ tịnh tiến theo vector tạo từ tâm mới và tâm cũ.
- Hình elip trong phương trình mask nghiêng một góc $\alpha = 45$ độ so với trục tọa độ.
- Có vô số hình elip thỏa điều kiện nằm trong và tiếp xúc với hình vuông. Do đó cần có một biến **size** để điều chỉnh tỷ lệ của các bán trục, với giá trị từ 0 đến 1.

Từ các ý tưởng đó, ta xây dựng được công thức cho 1 elip:

$$\frac{(x + y - \text{center\_x} - \text{center\_y})^2}{(\text{size} \times \text{row}^2)} + \frac{(x - y - \text{center\_x} + \text{center\_y})^2}{((1 - \text{size}) \times \text{row}^2)} = 1$$

Elip thứ hai được tạo bằng cách hoán đổi tỷ lệ các bán trục của elip 1

$$\frac{(x + y - \text{center\_x} - \text{center\_y})^2}{((1 - \text{size}) \times \text{row}^2)} + \frac{(x - y - \text{center\_x} + \text{center\_y})^2}{(\text{size} \times \text{row}^2)} = 1$$

2.12 Phóng to/thu nhỏ 2x

Hàm **zoom_img** có mục tiêu chính là thay đổi kích thước của một hình ảnh theo tỷ lệ phần trăm được chỉ định bằng cách sử dụng phương pháp nội suy điểm gần nhất (nearest-neighbor interpolation). Bằng cách thay đổi kích thước của hình ảnh dựa trên tỷ lệ phần trăm đầu vào, hàm có thể phóng to hoặc thu nhỏ hình ảnh. Điều này được thực hiện bằng cách ánh xạ từng điểm ảnh trong hình ảnh mới đến điểm ảnh gần nhất trong hình ảnh gốc. [8]

3 Mô tả các hàm

3.1 Các thư viện cần thiết

Các thư viện được sử dụng trong đồ án là: numpy, PIL và matplotlib

- numpy: Cho phép làm việc hiệu quả với ma trận và mảng. Đặc biệt là dữ liệu ma trận và mảng lớn vì có tốc độ xử lý nhanh, cấu trúc hàm đơn giản.
- Image từ PIL: Được sử dụng để đọc ảnh.
- matplotlib.pyplot: Được sử dụng để hiển thị ảnh.

3.2 read_img(img_path)

Hàm **read_img** được thiết kế để có thể đọc hình ảnh từ **img_path** được truyền vào và đổi nó thành một mảng numpy 2D bằng lệnh *img_2d = np.array(im)* giúp dễ dàng thao tác trong các tác vụ xử lý hình ảnh về sau.

3.3 show_img(img_2d)

Hàm **show_img** dùng để hiển thị một hình ảnh từ mảng numpy 2D sử dụng thư viện matplotlib. Vì bài toán đặt ra là xử lý hình ảnh nên có thể dùng **plt.axis('off')** để tắt các trục tọa độ xung quanh hình ảnh, giúp hình ảnh hiển thị rõ ràng hơn.

3.4 save_img(img_2d, img_path)

Hàm **save_img** được dùng để lưu một hình ảnh từ mảng numpy 2D vào một đường dẫn cho output file:

1. Kiểm tra xem đường dẫn **img_path** có hợp lệ hay chưa. Nếu chưa thì chỉnh sửa để đảm bảo có thể lưu được hình ảnh.
2. Chuyển đổi mảng numpy **img_2d** ban đầu thành đối tượng hình ảnh của PIL bằng hàm *Image.fromarray()*.
3. Yêu cầu người dùng chọn kiểu tệp: Hàm có hỗ trợ người dùng lưu ảnh dưới các dạng jpg, jpeg, png hoặc pdf. Người dùng nhập kiểu tệp muốn lưu, nếu kiểu tệp không hợp lệ thì hàm sẽ in ra thông báo lỗi.
4. Lưu hình ảnh: Đường dẫn **img_path** được cập nhật để bao gồm tên file đã quy định sẵn và kiểu tệp vừa được chọn. Sau đó hình ảnh sẽ được lưu vào đường dẫn này bằng hàm *save()*

3.5 adjust_brightness(image, brightness)

Hàm **adjust_brightness** dùng để điều chỉnh độ sáng của ảnh:

1. **Chuyển đổi hình ảnh sang mảng numpy:** Sử dụng **np.array()** để chắc chắn rằng dữ liệu đầu vào là một mảng numpy. Điều này giúp thực hiện các phép toán ma trận một cách hiệu quả.

2. **Điều chỉnh độ sáng:** Cộng giá trị độ sáng **brightness** vào từng phần tử trong mảng hình ảnh. Để đảm bảo không xảy ra lỗi tràn số khi cộng giá trị, chúng ta sử dụng **np.int16** cho giá trị điều chỉnh.

3. **Cắt giá trị pixel:** Sử dụng **np.clip()** để đảm bảo rằng mọi giá trị trong mảng đều nằm trong phạm vi từ **0 đến 255**. Điều này ngăn chặn các giá trị pixel không hợp lệ mà có thể xảy ra sau khi cộng giá trị độ sáng.

4. **Chuyển đổi lại về kiểu dữ liệu phù hợp:** Cuối cùng, chuyển đổi mảng đã được điều chỉnh về kiểu **np.uint8**, bởi vì đây là kiểu dữ liệu phổ biến để biểu diễn giá trị pixel trong hình ảnh.

3.6 adjust_contrast(image,contrast)

Hàm adjust_contrast dùng để điều chỉnh độ tương phản của ảnh:

1. **Chuyển đổi ảnh sang kiểu float32:** Để đảm bảo rằng các phép toán nhân chia không bị tràn số, hình ảnh được chuyển đổi sang kiểu **float32**.

2. **Điều chỉnh hệ số tương phản:**

   - Nếu **contrast** bằng 0, hàm trả về hình ảnh gốc không thay đổi.
   - Nếu **contrast** lớn hơn 0, hệ số tương phản được tính là nghịch đảo của giá trị đầu vào (**contrast = 1 / contrast**). Điều này giúp tăng độ tương phản khi giá trị **contrast** nhỏ hơn 1 và giảm độ tương phản khi giá trị lớn hơn 1.
   - Nếu **contrast** nhỏ hơn 0, hàm sử dụng giá trị tuyệt đối của **contrast** để tạo hiệu ứng đảo ngược màu sắc.

3. **Tính giá trị trung bình của hình ảnh:** Tính giá trị trung bình (**mean**) của toàn bộ các pixel trong hình ảnh để làm chuẩn cho việc điều chỉnh tương phản.

4. **Điều chỉnh độ tương phản:** Mỗi pixel trong hình ảnh được điều chỉnh dựa trên công thức **(image - mean) * contrast + mean**. Công thức này giúp tăng hoặc giảm độ tương phản tùy thuộc vào giá trị của **contrast.**

5. **Cắt giá trị pixel:** Sử dụng **np.clip()** để giới hạn giá trị của các pixel trong khoảng từ 0 đến 255, đảm bảo không có giá trị pixel nào vượt quá phạm vi hợp lệ.

6. **Chuyển đổi lại về kiểu uint8:** Cuối cùng, hình ảnh được chuyển đổi lại về kiểu **uint8** để đảm bảo định dạng hợp lệ cho pixel trong hình ảnh.

3.7 flip_horizontal(image)

Hàm **flip_horizontal** có nhiệm vụ lật ngược một hình ảnh theo chiều ngang.

1. **Chuyển đổi hình ảnh thành mảng numpy:** Sử dụng **np.array(image)** để đảm bảo rằng đầu vào là một mảng numpy, cho phép thực hiện thao tác trên các phần tử của hình ảnh một cách hiệu quả.
2. **Kiểm tra số chiều của hình ảnh:**
   - **Ảnh 3D (Ảnh màu):** Nếu hình ảnh có 3 chiều (ví dụ: hình ảnh màu RGB), nó có dạng **(height, width, channels)**. Để lật ngang, hàm sử dụng **img[:, ::-1, :]**, tức là giữ nguyên thứ tự của chiều **height** và **channels**, nhưng đảo ngược chiều **width** bằng cách sử dụng chỉ số **::-1**.
   - **Ảnh 2D (Ảnh xám):** Nếu hình ảnh có 2 chiều (ảnh xám), nó có dạng **(height, width)**. Cũng sử dụng **img[:, ::-1]** để đảo ngược chiều **width**, lật ảnh theo chiều ngang.
3. **Chuyển đổi lại về kiểu uint8:** Cuối cùng, hình ảnh được chuyển đổi lại về kiểu **uint8** để đảm bảo định dạng hợp lệ cho pixel trong hình ảnh.

3.8 flip_vertical(image)

Hàm **flip_vertical** có nhiệm vụ lật ngược hình ảnh theo chiều dọc:

1. **Chuyển đổi hình ảnh thành mảng numpy:** Sử dụng **np.array(image)** để đảm bảo rằng đầu vào là một mảng numpy, cho phép thực hiện thao tác trên các phần tử của hình ảnh một cách hiệu quả.
2. **Kiểm tra số chiều của hình ảnh:**
   - **Ảnh 3D (Ảnh màu):** Nếu hình ảnh có 3 chiều (ví dụ: hình ảnh màu RGB), nó có dạng **(height, width, channels)**. Sử dụng slicing của numpy để đảo ngược thứ tự các hàng của mảng ảnh **img[::-1, :, :]**
   - **Ảnh 2D (Ảnh xám):** Nếu hình ảnh có 2 chiều (ảnh xám), nó có dạng **(height, width)**. Tương tự với ảnh 3D, dùng **img[::-1,:]** để lật dọc ảnhảnh
3. **Chuyển đổi lại về kiểu uint8:** Cuối cùng, hình ảnh được chuyển đổi lại về kiểu **uint8** để đảm bảo định dạng hợp lệ cho pixel trong hình ảnh.numpy
4. **Trả về kết quả:** Trả về hình ảnh đã lật theo chiều dọc dưới dạng mảng numpy

3.9 rgb_to_gray(image)

1. **Kiểm tra xem hình ảnh có phải là ảnh xám không:** Nếu **image.ndim = 2**, điều này có nghĩa là hình ảnh đã là ảnh xám (không có kênh màu thứ ba). Trong trường hợp này, hàm chỉ cần chuyển đổi hình ảnh về kiểu **uint8** và trả về. Nếu không thì tiếp tục bước 2.
2. **Tách các kênh màu R, G, B từ ảnh:** Tách từng kênh màu từ mảng hình ảnh ban đầu:

```
R = img[:, :, 0]
G = img[:, :, 1]
B = img[:, :, 2]
```

3. **Tính toán giá trị xám dựa trên công thức trọng số:** Công thức tính toán giá trị xám bằng cách kết hợp các kênh màu theo trọng số đã được xác định:

```
gray = (R*0.3+G*0.59+B*0.11)
```

4. **Chuyển đổi lại về kiểu uint8:** Cuối cùng, hình ảnh được chuyển đổi lại về kiểu **uint8** để đảm bảo định dạng hợp lệ cho pixel trong hình ảnh.

5. **Trả về kết quả:** Trả về ảnh xám dưới dạng mảng numpy

3.10 rgb_to_sepia(image)

1. **Kiểm tra xem hình ảnh có phải là ảnh xám không:** Nếu **image.ndim = 2**, điều này có nghĩa là hình ảnh đã là ảnh xám (không có kênh màu thứ ba). Trong trường hợp này, cùng một giá trị được sử dụng cho cả ba kênh màu R, G, B để tạo ảnh sepia. Nếu không phải ảnh xám thì ta tách các kênh màu như bước 2.

2. **Tách các kênh màu R, G, B từ ảnh:** Tách từng kênh màu từ mảng ảnh RGB gốc để có thể áp dụng công thức sepia cho từng kênh:

```
R = img[:, :, 0]
G = img[:, :, 1]
B = img[:, :, 2]
```

3. **Tính toán các kênh màu mới cho ảnh sepia:** Thực hiện chuyển đổi hình ảnh RGB thành hình ảnh sepia bằng cách tính toán giá trị mới cho từng kênh màu với công thức ở phần ý tưởng

```
newRed = R * 0.393 + G * 0.769 + B * 0.189
newGreen = R * 0.349 + G * 0.686 + B * 0.168
newBlue = R * 0.272 + G * 0.534 + B * 0.131
```

4. **Cắt giá trị pixel trong phạm vi 0-255: np.clip(sepia_img, 0, 255)** đảm bảo rằng tất cả các giá trị pixel nằm trong phạm vi hợp lệ từ 0 đến 255, tránh vượt ra ngoài giới hạn khi cộng và nhân với trọng số.

5. **Chuyển đổi lại về kiểu uint8:** Cuối cùng, hình ảnh được chuyển đổi lại về kiểu **uint8** để đảm bảo định dạng hợp lệ cho pixel trong hình ảnh.

6. **Trả về kết quả:** Trả về ảnh xám dưới dạng mảng numpy

3.11 blur_image(image, kernel_size)

Hàm **blur_image** có nhiệm vụ làm mờ hình ảnh ban đầu thông qua phép tích chập (convolution) với một kernel làm mờ dựa trên **kernel_size** truyền vào (kernel_size phải là số lẻ để phép tích chập có một điểm trung tâm rõ ràng)

1. **Xác định số channels của ảnh:** Kiểm tra số chiều của mảng ảnh để xác định hình ảnh là RGB hay ảnh xám.
2. **Tạo kernel:** kernel trung bình được tạo là ma trận có các phần tử bằng nhau và tổng các phần tử bằng 1, giúp tính toán trung bình cộng của các pixel xung quanh.
3. **Thêm padding cho ảnh:** Padding là cần thiết để xử lý các pixel biên của hình ảnh, giúp phép tích chập được áp dụng lên toàn bộ ảnh mà không bị cắt bớt.
4. **Áp dụng phép tích chập cho từng kênh màu:** Duyệt qua từng pixel của hình ảnh gốc và áp dụng kernel để tính toán giá trị trung bình có trọng số cho từng vùng xung quanh.kernel
5. **Cắt giá trị pixel:** Sử dụng **np.clip** để giới hạn giá trị pixel trong khoảng từ 0 đến 255, tránh hiện tượng tràn số (overflow) khi chuyển đổi lại về kiểu dữ liệu **uint8**.
6. **Chuyển đổi lại về kiểu uint8:** Sau khi áp dụng làm mờ, chuyển đổi lại về kiểu **uint8** để phù hợp với định dạng của dữ liệu hình ảnh.
7. **Trả về kết quả:** Trả về ảnh đã làm mờ dưới dạng mảng numpy

3.12 sharpen_image(image)

1. **Xác định số channels của ảnh:** Kiểm tra số chiều của mảng ảnh để xác định hình ảnh là RGB hay ảnh xám.
2. **Tạo kernel:** Kernel làm nét là một ma trận 3×3 được sử dụng để thực hiện phép tích chập, nhằm làm nổi bật các đặc điểm sắc nét trong ảnh.
3. **Thêm padding cho ảnh:** Sử dụng **np.pad** để thêm padding cho ảnh, đảm bảo rằng kernel có thể áp dụng cho các pixel ở biên mà không bị mất thông tin.
4. **Tích chập ảnh với kernel:** Với mỗi kênh màu, thực hiện phép tích chập bằng cách nhân từng vùng 3×3 của ảnh với kernel làm nét và tính tổng giá trị để cập nhật giá trị pixel mới. Phép tính được thực hiện cho tất cả các pixel trong ảnh (hoặc từng kênh nếu là ảnh màu RGB).
5. **Cắt giá trị pixel:** Sử dụng **np.clip** để giới hạn giá trị pixel trong khoảng từ 0 đến 255, tránh hiện tượng tràn số (overflow) khi chuyển đổi lại về kiểu dữ liệu **uint8**.
6. **Chuyển đổi lại về kiểu uint8:** Sau khi áp dụng làm mờ, chuyển đổi lại về kiểu **uint8** để phù hợp với định dạng của dữ liệu hình ảnh.
7. **Trả về kết quả:** Trả về ảnh đã làm mờ dưới dạng mảng numpy

3.13 crop_center(image, size)

1. **Lấy kích thước ảnh gốc:** Lấy chiều cao (**height**) và chiều rộng (**width**) của ảnh từ thuộc tính **shape** của mảng.
2. **Tính tọa độ cắt:**

- Tọa độ **left** và **top**: Tọa độ điểm bắt đầu của khung cắt trên trục x (ngang) và trục y (dọc) được tính bằng cách lấy khoảng cách từ biên đến trung tâm ảnh và trừ đi một nửa của kích thước size.

$$left = \frac{width - size}{2}$$

$$top = \frac{height - size}{2}$$

- Tọa độ **right** và **bottom**: Tọa độ điểm kết thúc của khung cắt, được tính bằng cách cộng tọa độ bắt đầu với kích thước size.

$$right = left + size$$

$$bttom = top + size$$

3. **Cắt ảnh:** Sử dụng chỉ số slicing của numpy để cắt ảnh từ tọa độ **top** đến **bottom** và **left** đến **right**. Giúp lấy phần trung tâm của ảnh theo kích thước đã chỉ định.

4. **Trả về kết quả:** Trả về mảng ảnh đã được cắt, giữ nguyên định dạng và kiểu dữ liệu của ảnh gốc.

3.14 circular_crop(image)

1. **Lấy kích thước ảnh và tạo lưới tọa độ:** Lấy chiều cao **row** và chiều rộng **col** của ảnh từ thuộc tính **shape** của mảng. Sau đó sử dụng **np.ogrid** để tạo lưới tọa độ **x** và **y** có cùng kích thước với ảnh. Trong đó **x** chứa tọa độ hàng, **y** chứa tọa độ cột.

2. **Tính tọa độ tâm hình tròn:** Tính tọa độ **center_x** và **center_y** lần lượt là tọa độ trung tâm của hình tròn theo chiều ngang và chiều dọc

3. **Tạo mặt nạ hình tròn (mask)** Tạo một mặt nạ boolean **mask** xác định các pixel nằm trong hình tròn bằng cách so sánh khoảng cách từ mỗi điểm tới tâm với bán kính (radius).

4. **Áp dụng mặt nạ cho ảnh:** Nhân mặt nạ với ảnh để giữ lại các pixel trong hình tròn và đặt các pixel ngoài hình tròn thành 0.

5. **Chuyển đổi kiểu dữ liệu và trả về kết quả:** Kết quả được chuyển đổi về kiểu dữ liệu **uint8** để trả về một ảnh hợp lệ có thể hiển thị.

3.15 elliptical_crop(image, size)

1. **Lấy kích thước ảnh và tạo lưới tọa độ:** Lấy chiều cao **row** và chiều rộng **col** của ảnh từ thuộc tính **shape** của mảng. Sau đó sử dụng **np.ogrid** để tạo lưới tọa độ **x** và **y** có cùng kích thước với ảnh. Trong đó **x** chứa tọa độ hàng, **y** chứa tọa độ cột.

2. **Tính tọa độ tâm elip:** Tính tọa độ **center_x** và **center_y** lần lượt là tọa độ trung tâm của hình elip theo chiều ngang và chiều dọc

3. **Tạo mặt nạ elip (mask)** Tạo 2 mặt nạ boolean **mask_1** và **mask_2** xác định các pixel nằm trong 2 hình elip khác nhau bằng cách sử dụng 2 phương trình elip khác nhau đã được chứng minh. Hai phương trình này mô tả hai elip có các trục bán lớn và bán nhỏ khác nhau. Hệ số **size** và **1-size** cho phép điều chỉnh tỷ lệ giữa các trục.

4. **Kết hợp 2 mặt nạ** Sử dụng toán tử | (hoặc) để kết hợp hai mặt nạ **mask_1** và **mask_2**, tạo thành một mặt nạ tổng hợp mask bao gồm tất cả các pixel nằm trong ít nhất một trong hai elip.

$$mask\_1 = \frac{(x + y - \text{center\_x} - \text{center\_y})^2}{(\text{size} \times \text{row}^2)} + \frac{(x - y - \text{center\_x} + \text{center\_y})^2}{((1 - \text{size}) \times \text{row}^2)} \leqslant 1$$

$$mask\_2 = \frac{(x + y - \text{center\_x} - \text{center\_y})^2}{((1 - \text{size}) \times \text{row}^2)} + \frac{(x - y - \text{center\_x} + \text{center\_y})^2}{(\text{size} \times \text{row}^2)} \leqslant 1$$

5. **Áp dụng mặt nạ cho ảnh:** Nhân mặt nạ với ảnh để giữ lại các pixel trong vùng elip giao nhau và đặt các pixel ngoài vùng này thành 0.

6. **Chuyển đổi kiểu dữ liệu và trả về kết quả:** Kết quả được chuyển đổi về kiểu dữ liệu **uint8** để trả về một ảnh hợp lệ có thể hiển thị.

3.16 zoom_img(image, percent)

1. **Xác định kích thước mới:** Tính toán các kích thước mới **new_height** và **new_width** dựa trên tỷ lệ phần trăm phóng to được chỉ định.

2. **Khởi tạo hình ảnh mới:** Tạo **new_img** dựa trên kích thước vừa tính.

3. **Thực hiện nội suy điểm gần nhất:** Duyệt qua từng điểm ảnh trong **new_img** và tính toán điểm ảnh tương ứng từ hình ảnh gốc bằng cách sử dụng **int(i / percent) và int(j / percent)**. Điều này chọn điểm ảnh gần nhất trong hình ảnh gốc cho mỗi điểm ảnh trong hình ảnh mới.

4. **Hoàn thiện và trả kết quả:** Chuyển đổi các giá trị điểm ảnh trong **new_img** sang kiểu số nguyên 8-bit không dấu (**np.uint8**), đây là định dạng tiêu chuẩn cho dữ liệu điểm ảnh của hình ảnh. Sau đó trả về kết quả là ảnh đã thay đổi kích thước.

4 Kết quả

4.1 Bảng đánh giá mức độ hoàn thành

STT	Chức năng/Hàm	Mức độ hoàn thành
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%

Bảng 1: Bảng đánh giá từng chức năng

4.2 Hình ảnh kết quả

Hình đã được chỉnh tỉ lệ lại để có thể phù hợp với báo cáo. Kích thước gốc của tất cả các hình sau đây là 512x512 (riêng phần zoom thì là 256x256 và 1024x1024).

4.2.1 Ảnh gốc

Hình 1: Ảnh gốc

4.2.2 Thay đổi độ sáng

Hình 2: Tăng độ sáng

Hình 3: Giảm độ sáng

4.2.3 Thay đổi độ tương phản

Hình 4: Thay đổi độ tương phản

4.2.4 Lật ảnh

Hình 5: Lật ảnh ngang

Hình 6: Lật ảnh dọc

4.2.5 RGB thành ảnh xám/sepia

Hình 7: RGB thành ảnh xám

Hình 8: RGB thành ảnh sepia

4.2.6 Làm mờ ảnh

Hình 9: Làm mờ ảnh

4.2.7 Làm sắc nét ảnh

Hình 10: Làm sắc nét ảnh

4.2.8 Cắt ảnh theo kích thước

Hình 11: Cắt ảnh tại trung tâm

4.2.9 Cắt ảnh theo khung tròn/elip

Hình 12: Cắt ảnh theo khung tròn

Hình 13: Cắt ảnh theo khung elip

4.2.10 Phóng tho/thu nhỏ ảnh 2x

Hình 14: Thu nhỏ ảnh 2x

Hình 15: Phóng to ảnh 2x

Tài liệu

[1] John. (2019, April 29). Converting color to grayscale. John D. Cook | Applied Mathematics Consulting. https://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/

[2] How to convert a color image into sepia image - Image Processing Project - dyclassroom. https://dyclassroom.com/image-processing-project/how-to-convert-a-color-image-into-sepia-image

[3] Wikipedia contributors. (2024, June 10). Kernel (image processing). Wikipedia. https://en.wikipedia.org/wiki/Kernel_(image_processing)

[4] Kevinzakka. (n.d.). GitHub - kevinzakka/learn-blur: Learning about various image blurring techniques. GitHub. https://github.com/kevinzakka/learn-blur

[5] MSE2308198 - Lissajous Ellipse. (n.d.). Desmos. https://www.desmos.com/calculator/1gwixpvfn8

[6] vietjack.com. (n.d.-a). Công thức về phép quay hay nhất | Toán lớp 11. www.vietjack.com. https://vietjack.com/tai-lieu-mon-toan/cong-thuc-ve-phep-quay-ctqt11.jsp

[7] vietjack.com. (n.d.). Công thức về phép tịnh tiến hay nhất | Toán lớp 11. www.vietjack.com. https://vietjack.com/tai-lieu-mon-toan/cong-thuc-ve-phep-tinh-tien-ctqt11.jsp

[8] GeeksforGeeks. (2023, January 4). Image Processing without OpenCV Python. GeeksforGeeks. https://www.geeksforgeeks.org/image-processing-without-opencv-python/

Cộng tác viên

Đồ án 2 Image Processing được thực hiện với sự giúp đỡ của:

1. Chat GPT: https://chatgpt.com/
2. GitHub Copilot: https://github.com/features/copilot