

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



**ĐỒ ÁN THỰC HÀNH
TOÁN ỨNG DỤNG VÀ THỐNG KÊ – MTH00051**

ĐỒ ÁN THỰC HÀNH SỐ 02: IMAGE PROCESSING

THẦY (CÔ) HƯỚNG DẪN

CÔ. PHAN THỊ PHƯƠNG UYÊN
THẦY. NGUYỄN NGỌC TOÀN
THẦY. VŨ QUỐC HOÀNG
THẦY. NGUYỄN VĂN QUANG HUY

—o0o—

THÔNG TIN SINH VIÊN THỰC HIỆN

HỌ VÀ TÊN: LÊ PHƯỚC PHÁT

LỚP: 22CLC10

MSSV: 22127322

EMAIL: lpphat22@clc.fitus.edu.vn

THÀNH PHỐ HỒ CHÍ MINH, THÁNG 07 NĂM 2024

MỤC LỤC

I.	Đánh giá mức độ hoàn thành	6
II.	Ý tưởng thực hiện	9
1.	Thay đổi độ sáng và tương phản cho hình ảnh	9
a.	Thay đổi độ sáng hình ảnh	9
b.	Thay đổi độ tương phản hình ảnh	9
2.	Lật hình ảnh theo 2 chiều	10
3.	Chuyển đổi ảnh RGB thành ảnh xám / sepia	10
a.	Chuyển đổi ảnh RGB thành ảnh xám	10
b.	Chuyển đổi ảnh RGB thành ảnh sepia	12
4.	Làm mờ ảnh hay sắc nét ảnh	13
a.	Làm mờ ảnh	13
b.	Làm sắc nét ảnh	13
6.	Cắt hình ảnh theo khung	13
a.	Khung hình tròn	14
b.	Khung 2 hình elip chéo nhau	14
7.	Thu nhỏ/phóng to hình ảnh theo tỷ lệ cho trước (zoom in / zoom out)	15
III.	Mô tả các hàm chức năng quan trọng	15
1.	Import các thư viện liên quan	15
2.	Hàm đọc ảnh read_img (...)	15
3.	Hàm lưu ảnh save_img (...)	16
4.	Hàm hiển thị ảnh show_img (...)	16
5.	Hàm thay đổi độ sáng của ảnh adjust_brightness (...)	17
6.	Hàm thay đổi độ tương phản của hình ảnh adjust_contrast (...)	18
7.	Hàm lật ảnh theo hướng ngang – dọc flip_image (...)	18
8.	Hàm chuyển ảnh RGB sang ảnh xám rgb_to_grayscale (...)	19
9.	Hàm chuyển ảnh RGB sang ảnh sepia rgb_to_sepia (...)	20
10.	Hàm tính tích chập theo kernel cho trước apply_kernel (...)	20
11.	Hàm làm mờ ảnh blur_image (...)	21
12.	Hàm làm sắc nét ảnh sharpen_image (...)	22

Project 1. Image Processing	Toán Ứng Dụng & Thống Kê	Course Code: MTH00051
13.	Hàm cắt ảnh theo kích thước (cắt ở trung tâm) crop_center (...)	22
14.	Hàm cắt ảnh theo khung hình tròn crop_circle (...)	23
15.	Hàm cắt ảnh theo khung hai hình elip chéo nhau crop_elliptical_cross (...)	24
16.	Hàm phóng to hình ảnh zoom_in(...)	24
17.	Hàm thu nhỏ hình ảnh zoom_out (...)	25
18.	Hàm main.....	25

NHẬN XÉT CỦA GIÁO VIÊN


Ngày ..., tháng ..., năm 2024
Giáo viên nhận xét và chấm điểm

LỜI CAM KẾT

Tôi cam đoan rằng nghiên cứu này là do tôi thực hiện, dưới sự giám sát và hướng dẫn của các thầy cô bộ môn **Toán Ứng Dụng và Thống Kê – MTH00051**: cô **Phan Thị Phương Uyên** và thầy **Nguyễn Ngọc Toàn**. Kết quả của nghiên cứu này là hợp pháp và chưa được công bố ở bất kỳ hình thức nào trước đây. Tất cả các tài liệu được sử dụng trong nghiên cứu này được tôi thu thập bằng cách tự mình và từ các nguồn khác nhau, và được liệt kê đầy đủ trong phần tài liệu tham khảo. Ngoài ra, chúng tôi cũng sử dụng kết quả của một số tác giả và tổ chức khác. Tất cả đều được trích dẫn đúng đắn. Trong trường hợp có phạm bản quyền, chúng tôi chịu trách nhiệm cho hành động đó. Do đó, **Trường Đại học Khoa học Tự nhiên TP.HCM** không chịu trách nhiệm về bất kỳ vi phạm bản quyền nào được thực hiện trong nghiên cứu của tôi.


NỘI DUNG BÁO CÁO

I. 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 2 elip chéo nhau	100 %	
8	Hàm main	100 %	X

9	Phóng to / Thu nhỏ 2x	100 %	
---	-----------------------	-------	---

II. Ý tưởng thực hiện

1. Thay đổi độ sáng và tương phản cho hình ảnh

Ta có công thức thay đổi độ sáng và độ tương phản cho hình ảnh như sau:

$$g(i, j) = \alpha f(i, j) + \beta$$

Trong đó:

- $g(i, j)$: là điểm ảnh (pixel) đầu ra sau khi được điều chỉnh nằm trong phạm vi từ $[0; 255]$.
- $f(i, j)$: là điểm ảnh (pixel) đầu vào của tấm hình và phạm vi giá trị của các pixel sẽ nằm trong đoạn $[0; 255]$.
- α : là hệ số đo độ tương phản mà ta thay đổi (mặc định của một tấm ảnh khi đọc vào là $\alpha = 1$).
- β : là hệ số độ sáng mà ta muốn điều chỉnh và giá trị của hệ số này sẽ nằm trong đoạn $[-255; 255]$

a. Thay đổi độ sáng hình ảnh

Do một điểm ảnh $f(i, j)$ sẽ mang giá trị trong đoạn $[0; 255]$ với $f(i, j) = 0$ thì điểm ảnh (pixel) đó sẽ thể hiện màu đen hoàn toàn và $f(i, j) = 255$ thì điểm ảnh sẽ hiện một màu trắng hoàn toàn. Nói cách khác, nếu giá trị điểm ảnh (pixel) càng cao, màu điểm ảnh sẽ càng tiến về màu sáng. Nếu ta cộng giá trị điểm ảnh cho một giá trị thì các màu trong từng điểm pixel sẽ bị thay đổi trong một khoảng nhất định, cụ thể là:

- Nếu $0 < \beta < 255$, thì các màu trong từng điểm ảnh sẽ sáng hơn.
- Nếu $-255 < \beta < 0$, thì các màu trong từng điểm ảnh sẽ bị giảm độ sáng và trở nên dần tối hơn.
- Nếu $\beta = 0$, thì các màu trong từng điểm ảnh sẽ bị không thay đổi giá trị, tức là sẽ giữ nguyên độ sáng hiện trạng.

b. Thay đổi độ tương phản hình ảnh

Đối với độ tương phản, khi càng tăng độ tương phản, tức là tăng khoảng cách giá trị giữa các màu. Để làm được điều này, một cách đơn giản là ta có thể nhân toàn bộ ảnh cho một giá trị số thực $\alpha > 0$ để làm tăng khoảng cách giá trị màu giữa các điểm ảnh. Cụ thể là:

- Nếu $\alpha > 1$, ta sẽ có ảnh đã được tăng độ tương phản. Ví dụ, ta có hai điểm pixel lần lượt là $a = 1$ và $b = 3$ thì khoảng cách hiện tại của chúng là $\delta = 2$. Nếu cùng nhân cả hai giá trị a, b cho $\alpha = 2$ thì khoảng cách lúc sau của chúng sẽ tăng thành $\delta = 4$.
- Nếu $0 < \alpha < 1$, ảnh sẽ được giảm độ tương phản. Ví dụ ta có hai điểm pixel lần lượt là $a = 1$ và $b = 3$ thì khoảng cách hiện tại của chúng là $\delta = 2$. Nếu cùng nhân cả hai giá trị điểm ảnh trên cho $\alpha = \frac{1}{2}$ thì khoảng cách của chúng sẽ giảm thành $\delta = 1$.

2. Lật hình ảnh theo 2 chiều

Giả sử hình ảnh là một ma trận A có chiều rộng là w và chiều cao là h . Mỗi giá trị của từng điểm ảnh được biểu diễn dưới từng giá trị tọa độ $A[i, j]$ với i là chỉ số hàng và j là chỉ số cột.

a. Lật hình ảnh ngang (horizontal flip)

Khi hình ảnh bị lật ngang, chúng ta sẽ giữ nguyên từng hàng của ma trận điểm ảnh và lấy ngược từng cột của ma trận điểm ảnh. Tức là, mỗi điểm ảnh $A[i, j]$ sẽ được chuyển tới vị trí mới $B[i, w - 1 - j]$ trong ma trận kết quả B .

$$B[i, j] = A[i, w - 1 - j]$$

b. Lật hình ảnh dọc (vertical flip)

Tương tự, khi hình ảnh bị lật dọc, chúng ta sẽ giữ nguyên từng cột của ma trận điểm ảnh và lấy ngược từng dòng của ma trận điểm ảnh. Tức là, mỗi điểm ảnh $A[i, j]$ sẽ được chuyển tới vị trí mới $B[h - 1 - i, j]$ trong ma trận kết quả B .

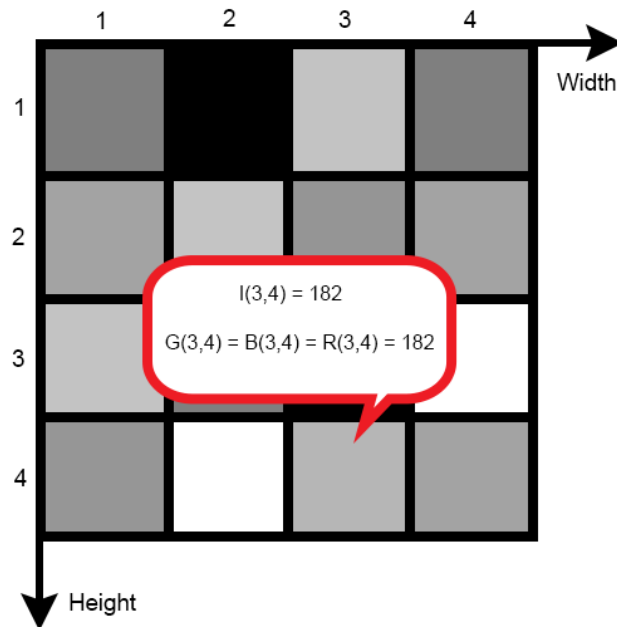
$$B[i, j] = A[h - 1 - i, j]$$

3. Chuyển đổi ảnh RGB thành ảnh xám / sepia

a. Chuyển đổi ảnh RGB thành ảnh xám

- Ảnh xám (grayscale) hay còn gọi là ảnh đơn sắc (monochromatic) là ảnh chỉ mang thông tin về độ sáng và tối (độ trắng và đen) của từng điểm ảnh, là một hệ thống màu có mô hình màu đơn giản nhất với 256 cấp độ xám biến thiên từ màu đen đến màu trắng.
- Ảnh xám được sử dụng cả trong công nghiệp in lẫn dùng trong việc thể hiện ảnh lên các thiết bị số.
- Mỗi giá trị điểm ảnh trong ma trận mang giá trị từ 0 đến 255.

- Trong không gian màu RGB, để có 1 ảnh xám cần phải có giá trị kênh màu $\text{Red}(i, j) = \text{Green}(i, j) = \text{Blue}(i, j)$ với i, j lần lượt là tọa độ của mỗi điểm ảnh.



Đối với chức năng này thì ta sẽ có thể sử dụng hai phương pháp để chuyển ảnh màu RGB sang ảnh xám grayscale:

- **Phương pháp 1: phương pháp trung bình.**

- Ta sẽ tách từng kênh màu (hoặc giá trị của từng kênh màu tại một điểm ảnh đang xét) của ảnh RGB ban đầu:
 - **Red (x, y)** là giá trị điểm ảnh hay vector điểm ảnh đại diện cho kênh màu đỏ.
 - **Green (x, y)** là giá trị điểm ảnh hay vector điểm ảnh đại diện cho kênh màu lục.
 - **Blue (x, y)** là giá trị điểm ảnh hay vector điểm ảnh đại diện cho kênh màu xanh dương.
- Tính trung bình giá trị điểm ảnh thông qua 3 kênh màu thì ta sẽ được một giá trị điểm ảnh mới theo công thức:

$$f(x, y) = \frac{1}{3} \times (\text{Red}(x, y) + \text{Green}(x, y) + \text{Blue}(x, y))$$

- Tuy nhiên ảnh vừa mới tạo có một kênh, ta phải chồng lên ảnh vừa mới tạo hai lần tạo thành một cái tensor tương tự như mô hình ảnh RGB.

- **Phương pháp 2: phương pháp trọng số (weighted)**

Ta sẽ sử dụng phương pháp tính giá trị cường độ sáng của từng điểm ảnh từ ảnh RGB theo mảng trọng số “weight” gồm các giá trị $[0.299, 0.587, 0.114]$ và áp dụng công thức:

$$I(x, y) = 0.299 \times \text{Red}(x, y) + 0.587 \times \text{Green}(x, y) + 0.114 \times \text{Blue}(x, y)$$

Trong đó:

- $I(x, y)$: cường độ sáng tại điểm ảnh (x, y) của ảnh xám.
- $\text{Red}(x, y)$: giá trị kênh màu Red (đỏ) tại điểm ảnh (x, y) của ảnh màu RGB.
- $\text{Green}(x, y)$: giá trị của kênh màu Green (lục) tại điểm ảnh (x, y) của ảnh màu RGB.
- $\text{Blue}(x, y)$: giá trị kênh màu Blue (xanh) tại điểm ảnh (x, y) của ảnh màu RGB.

Lưu ý: trong đồ án này, em sẽ sử dụng **phương pháp 2 – phương pháp tính ma trận trọng số “weight”** để chuyển đổi ảnh màu RGB sang ảnh xám (grayscale).

b. *Chuyển đổi ảnh RGB thành ảnh sepia*

- Đầu tiên, ta sẽ tách từng kênh màu (hoặc giá trị của từng kênh màu tại một điểm ảnh đang xét) của ảnh RGB ban đầu:
 - $\text{Red}(x, y)$ là giá trị điểm ảnh hay vector điểm ảnh đại diện cho kênh màu đỏ.
 - $\text{Green}(x, y)$ là giá trị điểm ảnh hay vector điểm ảnh đại diện cho kênh màu lục.
 - $\text{Blue}(x, y)$ là giá trị điểm ảnh hay vector điểm ảnh đại diện cho kênh màu xanh dương.
- Sau đó, ta sẽ áp dụng công thức chuyển đổi RGB sang ảnh sepia:
 - Đối với kênh đỏ (R), ta sẽ dùng bộ ma trận trọng số $[0.393, 0.769, 0.189]$ với trọng số lớn (0.393) cho kênh đỏ gốc và trọng số lớn cho kênh lục (0.769) để tạo ra tông màu ấm của sepia.

$$tR(x, y) = 0.393 \times \text{Red}(x, y) + 0.769 \times \text{Green}(x, y) + 0.189 \times \text{Blue}(x, y)$$

- Đối với kênh lục (G), ta sử dụng trọng số hơi thấp hơn cho kênh đỏ (0.349) và kênh xanh lá cây (0.686), cùng với trọng số nhỏ hơn cho kênh xanh dương (0.168).

$$tG(x, y) = 0.349 \times \text{Red}(x, y) + 0.686 \times \text{Green}(x, y) + 0.168 \times \text{Blue}(x, y)$$

- Và đối với kênh lam (B), ta sẽ dùng trọng số nhỏ nhất cho tất cả các kênh, tạo ra tông màu nâu đặc trưng.

$$tB(x, y) = 0.272 \times \text{Red}(x, y) + 0.534 \times \text{Green}(x, y) + 0.131 \times \text{Blue}(x, y)$$

- Tiếp theo, ta sẽ cắt giá trị và thiết lập giá trị mới cho từng điểm ảnh sepia:
 - Nếu $tR(x, y) > 255$ thì $\text{Red}(x, y) = 255$, ngược lại thì $\text{Red}(x, y) = tR(x, y)$.
 - Nếu $tG(x, y) > 255$ thì $\text{Green}(x, y) = 255$, ngược lại thì $\text{Green}(x, y) = tG(x, y)$.
 - Nếu $tB(x, y) > 255$ thì $\text{Blue}(x, y) = 255$, ngược lại thì $\text{Blue}(x, y) = tB(x, y)$.

4. Làm mờ ảnh hay sắc nét ảnh

a. Làm mờ ảnh

Quy trình thực hiện bao gồm việc đệm hình ảnh để giữ nguyên kích thước sau khi áp dụng kernel, khởi tạo một mảng mới để lưu trữ kết quả, và áp dụng kernel lên từng vùng của hình ảnh. Các giá trị pixel trong hình ảnh kết quả được giới hạn trong khoảng từ 0 đến 255 và chuyển đổi về định dạng uint8 để đảm bảo tính hợp lệ và độ chính xác của hình ảnh đầu ra.

Định nghĩa một kernel làm mờ dưới dạng ma trận 5x5 với tất cả các phần tử bằng 1/25, nhằm trung bình hóa các giá trị pixel trong khu vực 5x5 xung quanh mỗi pixel. Việc làm mờ được thực hiện bằng cách áp dụng kernel này lên hình ảnh đầu vào, giúp làm giảm chi tiết và làm mềm các cạnh của hình ảnh.

b. Làm sắc nét ảnh

Sử dụng một kernel sắc nét định nghĩa dưới dạng ma trận 3x3 với các giá trị cụ thể để tăng cường các cạnh trong hình ảnh. Kernel này được thiết kế để làm nổi bật sự khác biệt giữa các vùng sáng và tối, từ đó cải thiện độ rõ ràng và chi tiết của hình ảnh.

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

5. Cắt hình ảnh theo kích thước cho sẵn (ở trung tâm)

Trước tiên, ta sẽ xác định kích thước hình ảnh. Sau đó, nó tính toán tọa độ của các cạnh trái, trên, phải và dưới của vùng cắt bằng cách lấy phần trung tâm của hình ảnh. Cụ thể, tọa độ cạnh trái và trên được tính bằng cách lấy hiệu của chiều rộng (hoặc chiều cao) gốc và chiều rộng (hoặc chiều cao) mới, chia đôi và lấy phần nguyên. Tọa độ cạnh phải và dưới được tính bằng cách cộng chiều rộng (hoặc chiều cao) mới vào tọa độ cạnh trái (hoặc cạnh trên). Cuối cùng, sử dụng các tọa độ này để cắt và trả về phần ảnh trung tâm với kích thước yêu cầu.

6. Cắt hình ảnh theo khung

a. Khung hình tròn

Để thực hiện việc cắt hình ảnh theo khung hình tròn, trước tiên, chúng ta xác định kích thước của hình ảnh gốc, bao gồm chiều cao và chiều rộng. Sau đó, tạo một lưới tọa độ bằng cách sử dụng các chỉ số hàng và cột cho từng điểm trong hình ảnh. Tọa độ của tâm hình ảnh được tính bằng cách chia chiều cao và chiều rộng của hình ảnh cho hai, lần lượt cho trục dọc và ngang.

Tiếp theo, bán kính của hình tròn được tính bằng cách chọn giá trị nhỏ hơn giữa nửa chiều cao và nửa chiều rộng, đảm bảo rằng hình tròn sẽ hoàn toàn nằm trong giới hạn của hình ảnh. Để tạo mặt nạ hình tròn, chúng ta sử dụng công thức toán học, trong đó mặt nạ Boolean xác định các điểm nằm trong hoặc trên biên của hình tròn với bán kính đã tính và tâm hình ảnh đã xác định.

Cuối cùng, một hình ảnh mới được tạo ra với cùng kích thước như hình ảnh gốc nhưng với tất cả các giá trị màu sắc được khởi tạo là 0 (màu đen). Áp dụng mặt nạ hình tròn để giữ lại các pixel nằm trong hình tròn và loại bỏ các pixel bên ngoài hình tròn bằng cách đặt chúng thành màu đen.

b. Khung 2 hình elip chéo nhau

Ta sẽ lấy chiều cao và chiều rộng của hình ảnh đầu vào và tạo một bản sao của hình ảnh để thực hiện các phép toán mà không làm thay đổi hình ảnh gốc.

Xác định các thông số:

Các thông số cần thiết cho hai elip bao gồm:

- Trung tâm của hình ảnh (x, y) được xác định là điểm giữa của hình ảnh.
- Bán trục chính (s_major) được đặt bằng chiều rộng của hình ảnh.
- Bán trục phụ (s_minor) được tính toán dựa trên chiều rộng và một yếu tố tỷ lệ để tạo ra một elip giao nhau.
- Bán kính (radius) được xác định là giá trị nhỏ nhất giữa bán kính chiều cao và chiều rộng để phù hợp với kích thước hình ảnh.

Tạo hai ma trận 2D y và x để đại diện cho lưới pixel của hình ảnh.

Tính toán khoảng cách từ mỗi pixel đến trung tâm hình ảnh và sử dụng các khoảng cách này để xác định các vùng bên trong elip.

Tạo hai mặt nạ boolean (mask1 và mask2) để xác định các pixel nằm trong hai elip giao nhau. Những pixel nằm trong ít nhất một trong hai elip này sẽ có giá trị True, trong khi các pixel khác sẽ có giá trị False.

Các pixel nằm ngoài vùng được xác định bởi mặt nạ elip sẽ bị gán giá trị 0, có nghĩa là chúng sẽ bị loại bỏ hoặc trở thành đen trong hình ảnh kết quả.

Hình ảnh đã được cắt bằng cách áp dụng mặt nạ elip và chỉ giữ lại các phần của hình ảnh nằm trong khu vực được xác định bởi mặt nạ.

7. Thu nhỏ/phóng to hình ảnh theo tỷ lệ cho trước (zoom in / zoom out)

Phóng to hình ảnh: Quy trình bắt đầu bằng việc xác định kích thước của hình ảnh gốc, sau đó tính toán kích thước mới dựa trên yếu tố zoom. Một mảng hình ảnh mới được khởi tạo với kích thước lớn hơn, và các pixel trong hình ảnh mới được điền giá trị từ hình ảnh gốc bằng cách ánh xạ tọa độ. Kết quả là hình ảnh được phóng to với kích thước tăng theo yếu tố zoom, nhưng không sử dụng kỹ thuật nội suy, có thể dẫn đến hiện tượng răng cưa hoặc mất chi tiết.

Thu nhỏ hình ảnh: Tương tự như hàm phóng to, hàm này bắt đầu bằng việc lấy kích thước của hình ảnh gốc và tính toán kích thước mới của hình ảnh sau khi thu nhỏ. Một mảng hình ảnh mới với kích thước giảm được khởi tạo, và các pixel được ánh xạ từ hình ảnh gốc vào vị trí tương ứng trong hình ảnh thu nhỏ. Kết quả là hình ảnh được giảm kích thước, giữ lại các chi tiết chính của hình ảnh gốc.

Cả hai đều không sử dụng nội suy pixel, dẫn đến hình ảnh kết quả có thể không mịn mà có thể xuất hiện hiện tượng răng cưa hoặc mất chi tiết.

III. Mô tả các hàm chức năng quan trọng

1. Import các thư viện liên quan

Đầu tiên, ta cần phải cài đặt các thư viện liên quan trong quá trình thực hiện chương trình. Các thư viện cần cài đặt:

- *Thư viện numpy*: dùng để tính toán các phép ma trận, vector, ...
- *Thư viện pillow*: dùng để đọc ảnh và lưu ảnh.
- *Thư viện matplotlib*: dùng để hiển thị hình ảnh.

2. Hàm đọc ảnh read_img(...)

Hàm “read_img” đọc hình ảnh từ đường dẫn được cung cấp và trả về hình ảnh dưới dạng một mảng numpy.

- Tham số đầu vào: đường dẫn ảnh img_path có kiểu string.
- Kết quả đầu ra: ảnh dưới dạng mảng numpy 2D có dạng np.ndarray.
- Mô tả cách hoạt động

Đối với hàm này, chúng ta cần phải kiểm tra điều kiện tiên quyết trước.

- Kiểm tra kiểu đường dẫn ảnh mà ta cần truyền vào có phù hợp hay không, tức là img_path có phải là kiểu string hay không.
- Sau đó ta kiểm tra xem file ảnh mà đường dẫn đưa tới có tồn tại hay không. Sau khi kiểm tra các điều kiện tiên quyết, ta sẽ mở ảnh để đọc ảnh.

- Sử dụng câu lệnh “try” để xử lý các ngoại lệ trong quá trình đọc và xử lý ảnh.
- “Image.open(img_path)”: mở tệp ảnh bằng thư viện Pillow (PIL).
- Sử dụng “with” statement để đảm bảo tệp được đóng một cách đúng đắn sau khi hoàn thành.
- “img” được chuyển đổi thành mảng numpy bằng cách sử dụng “np.array(img)”

Cuối cùng, hàm trả về ảnh dưới dạng một mảng numpy 2D với kích thước (height, width, channels)

3. Hàm lưu ảnh save_img(...)

Hàm “save_img” này được thiết kế để lưu một hình ảnh vào đường dẫn chỉ định. Hàm này bao gồm các bước kiểm tra và xử lý để đảm bảo rằng dữ liệu đầu vào hợp lệ và hỗ trợ các định dạng hình ảnh cơ bản như ảnh xám và ảnh RGB.

- Tham số đầu vào: hàm này nhận hai tham số
 - “img”; hình ảnh dưới dạng mảng numpy 2D với shape là (height, width, channels).
 - “img_path”: đường dẫn nơi hình ảnh sẽ được lưu có kiểu string.
- Kết quả đầu ra: None.
- Mô tả cách hoạt động

Đầu tiên, ta sẽ kiểm tra các tham số đầu vào có thỏa điều kiện hay không ?

- Kiểm tra “img có phải là một mảng numpy (np.ndarray) không. Nếu không, ném ra ngoại lệ “ValueError” với thông báo lỗi.
- Kiểm tra xem img_path có phải là một chuỗi không. Nếu không, ném ra ngoại lệ “ValueError” với thông báo lỗi.

Tiếp theo, ta sẽ xử lý hình ảnh trên định dạng bằng cách chuyển đổi mảng numpy của hình ảnh sang đối tượng hình ảnh PIL để dùng matplotlib hiển thị hình ảnh.

- Nếu “img” là một mảng 2D (“ndim == 2”), nó được xem là ảnh xám. Ta sẽ sử dụng “Image.fromarray” để chuyển đổi mảng numpy thành đối tượng hình ảnh PIL với chế độ “L”.
- Nếu “img” là một mảng 3D với 3 kênh (“ndim == 3” và “img.shape[2] == 3”), nó được coi là ảnh RGB. Sử dụng “Image.fromarray” để chuyển đổi mảng NumPy thành đối tượng hình ảnh PIL với chế độ “RGB”.
- Nếu “img” không khớp với định dạng nào trong hai định dạng trên, ném ra ngoại lệ ValueError với thông báo lỗi.

Cuối cùng, ta sẽ lưu đối tượng hình ảnh PIL vào đường dẫn “img_path”.

4. Hàm hiển thị ảnh show_img (...)

Hàm “show_img” có tác dụng hiển thị một hình ảnh từ một mảng Numpy.

- Tham số đầu vào: “img” hình ảnh dưới dạng mảng numpy.s
- Kết quả đầu ra: None
- Mô tả cách hoạt động

Đầu tiên, ta sẽ kiểm tra ảnh “img” có phải là một mảng NumPy không. Nếu không, ném ra ngoại lệ “ValueError” với thông báo lỗi.

Sau đó, ta sẽ hiển thị hình ảnh:

- “**plt.figure(figsize=(8, 8))**”: tạo một đối tượng figure với kích thước 8x8 inch.
- “**plt.imshow(img)**”: hiển thị hình ảnh img. Hình ảnh này có thể có dạng (height, width) cho ảnh xám hoặc (height, width, channels) cho ảnh màu.
- “**plt.axis("off")**”: tắt hiển thị trục để hình ảnh hiển thị không bị ảnh hưởng bởi trục tọa độ
- “**plt.show()**”: hiển thị biểu đồ chứa hình ảnh.

5. Hàm thay đổi độ sáng của ảnh `adjust_brightness(...)`

Để xử lý việc điều chỉnh độ sáng tối của hình ảnh, chúng ta cần sử dụng một số hàm của numpy, do np. ndarray được hỗ trợ có thể broadcasting, nên chúng ta chỉ cần cộng ma trận ảnh cho một số nguyên bias biểu thị độ sáng tối của hình ảnh thì numpy đã có thể tự hiểu là cộng toàn bộ các giá trị cho bias đó.

Hàm “adjust_brightness” này sẽ giúp điều chỉnh độ sáng / tối của hình ảnh bằng cách trên.

- Tham số đầu vào: hàm này sẽ nhận hai tham số
 - “image”: mảng numpy 2D đại diện cho hình ảnh. Mảng numpy này nên là dạng 8-bit gồm các điểm ảnh pixels có giá trị trong đoạn [0, 255].
 - “bias”: giá trị số nguyên để điều chỉnh độ sáng tối của hình ảnh nằm trong đoạn [-255, 255].
 - bias > 0: ảnh sẽ tăng độ sáng.
 - bias < 0: ảnh sẽ giảm độ sáng.
 - Bias = 0: ảnh không tăng độ sáng.
- Kết quả đầu ra: ảnh mảng numpy có cùng kích thước với hình ảnh đầu vào, với độ sáng đã được điều chỉnh. Hình ảnh đầu ra cũng ở định dạng 8-bit, với các giá trị pixel được cắt giữa 0 và 255.
- Mô tả cách hoạt động

Đầu tiên ta sẽ chuyển đổi mảng hình ảnh từ kiểu dữ liệu “uint8” (8-bit) sang “int16” để tránh tràn số khi điều chỉnh độ sáng.

Sau đó, ta sẽ điều chỉnh độ sáng và cắt giá trị của mỗi điểm pixel trong mảng numpy.

- Ta sẽ tăng giá trị của mỗi điểm ảnh trong mảng với giá trị bias bằng cách broadcasting giá trị bias thành ma trận có cùng kích thước với mảng numpy “image” và thực hiện phép element-wise với mỗi điểm ảnh pixels như trên.
- Tiếp theo, ta sẽ dùng hàm np.clip() để cắt giá trị mỗi điểm ảnh nhằm giới hạn giá trị đó nằm trong đoạn [0; 255] bởi vì giá trị của mỗi điểm ảnh trong mảng numpy chỉ từ 0 đến 255.

Cuối cùng, ta sẽ chuyển đổi mảng numpy sau khi đã thực hiện việc điều chỉnh từ kiểu dữ liệu “int16” trở lại “uint8” để đảm bảo hình ảnh đầu ra ở định dạng 8-bit.

6. Hàm thay đổi độ tương phản của hình ảnh `adjust_contrast(...)`

Đối với độ tương phản, để xử lý được việc này chúng ta cũng sẽ sử dụng một số hàm từ numpy và thực hiện phép broadcasting cho giá trị bias và thực hiện phép element wise với mảng numpy đầu vào.

Hàm “`adjust_contrast`” này điều chỉnh độ tương phản của ảnh đầu vào.

- Tham số đầu vào: hàm này sẽ nhận hai tham số
 - “image”: mảng numpy 2D đại diện cho hình ảnh. Mảng numpy này nên là dạng 8-bit gồm các điểm ảnh pixels có giá trị trong đoạn [0, 255].
 - “bias”: giá trị số thực kiểu float dương để điều chỉnh độ tương phản.
 - $0 < \text{bias} < 1$: ảnh sẽ giảm độ tương phản theo tỷ lệ bias.
 - $\text{bias} > 1$: ảnh sẽ tăng độ tương phản theo tỷ lệ bias.
 - $\text{bias} = 1$: ảnh sẽ giữ nguyên độ tương phản.

- Kết quả đầu ra: mảng NumPy có cùng kích thước với hình ảnh đầu vào, với độ tương phản đã được điều chỉnh. Hình ảnh đầu ra cũng ở định dạng 8-bit, với các giá trị pixel được cắt giữa 0 và 255.

- Mô tả cách hoạt động

Đầu tiên, ta sẽ chuyển đổi mảng hình ảnh từ kiểu dữ liệu uint8 (8-bit) sang int16 để tránh tràn số khi điều chỉnh độ tương phản.

Sau đó, ta sẽ điều chỉnh độ tương phản và cắt giá trị mỗi điểm ảnh:

- Ta sẽ nhân giá trị “bias” với mỗi giá trị trong pixel trong mảng bằng phép broadcasting và element-wise.
- Tiếp theo, ta sẽ dùng hàm np.clip() để cắt giá trị mỗi điểm ảnh nhằm giới hạn giá trị đó nằm trong đoạn [0; 255] bởi vì giá trị của mỗi điểm ảnh trong mảng numpy chỉ từ 0 đến 255.

Cuối cùng, chuyển đổi mảng từ kiểu dữ liệu int16 trở lại uint8 để đảm bảo hình ảnh đầu ra ở định dạng 8-bit.

7. Hàm lật ảnh theo hướng ngang – dọc `flip_image(...)`

Hàm “`flip_image`” này dùng để lật hình ảnh theo chiều ngang hoặc chiều dọc.

- Tham số đầu vào: hàm này sẽ nhận hai tham số

- “image”: mảng NumPy đại diện cho hình ảnh. Hình ảnh có thể là ảnh grayscale hoặc RGB (với 3 kênh màu).
- “mode”: chuỗi xác định chế độ lật, có thể là "horizontal" để lật theo chiều ngang hoặc "vertical" để lật theo chiều dọc.
- Kết quả đầu ra: mảng NumPy có cùng kích thước với hình ảnh đầu vào, hình ảnh được lật theo chế độ đã chỉ định.
- Mô tả cách hoạt động
Đầu tiên, ta sẽ chọn mode là “vertical” hay là “horizontal”.
 - Nếu mode là "horizontal", lật hình ảnh theo chiều ngang bằng cách sử dụng np.flip với axis=1.
 - Nếu mode là "vertical", lật hình ảnh theo chiều dọc bằng cách sử dụng np.flip với axis=0.
 - Nếu mode không phải là "horizontal" hoặc "vertical", ném ra ngoại lệ ValueError.

Cuối cùng, trả về mảng hình ảnh đã được lật theo chế độ đã chỉ định.

8. Hàm chuyển ảnh RGB sang ảnh xám rgb_to_grayscale (...)

Hàm “rgb_to_grayscale” chuyển đổi một hình ảnh rgb thành ảnh xám bằng cách áp dụng công thức chuẩn độ sáng.

- Tham số đầu vào: “image” mảng numpy đại diện cho hình ảnh RGB, với hình dạng (height, width, 3) và các giá trị pixels nằm trong đoạn [0, 255].
- Kết quả đầu ra: mảng NumPy đại diện cho hình ảnh grayscale, có cùng chiều rộng và chiều cao với hình ảnh đầu vào. Hình ảnh đầu ra ở định dạng 8-bit và có một kênh với các giá trị từ 0 đến 255.
- Mô tả cách hoạt động

Đầu tiên ta sẽ kiểm tra hình ảnh đầu vào để đảm bảo hình đầu vào là hình RGB với 3 kênh màu.

Sau đó, ta sẽ trích xuất các kênh màu đỏ, lục lam từ hình ảnh RGB ban đầu bằng phép slicing để chỉ lấy đúng 3 kênh màu đầu tiên và bỏ qua các kênh màu còn lại với mỗi kênh màu bấy giờ có kích thước như hình ảnh ban đầu.

Tiếp theo, ta sẽ thực hiện việc nhân mỗi kênh màu với trọng số tương ứng để tính toán giá trị ảnh xám theo công thức chuẩn cho độ sáng. Ta thực hiện hiện phép broadcasting và element-wised để nhân từng trọng số tương ứng với mỗi kênh màu vào từng giá trị của điểm ảnh.

Tiếp tục cộng các ma trận trọng số tương ứng lại với nhau để cho ra cường độ sáng của mỗi điểm ảnh và được các giá trị điểm ảnh của ảnh xám.

Tiếp theo, ta sẽ dùng hàm np.clip() để cắt giá trị mỗi điểm ảnh nhằm giới hạn giá trị đó nằm trong đoạn [0; 255] bởi vì giá trị của mỗi điểm ảnh trong mảng numpy chỉ từ 0 đến 255.

Cuối cùng, chuyển đổi mảng từ kiểu dữ liệu int16 trở lại uint8 để đảm bảo hình ảnh đầu ra ở định dạng 8-bit.

9. Hàm chuyển ảnh RGB sang ảnh sepia `rgb_to_sepia(...)`

Hàm “`rgb_to_sepia`” chuyển đổi một hình ảnh RGB thành hình ảnh có tông màu sepia, tạo ra hiệu ứng màu nâu ấm đặc trưng của sepia.

- Tham số đầu vào: mảng NumPy đại diện cho hình ảnh RGB với các giá trị từ 0 đến 255.
- Kết quả đầu ra: mảng NumPy đại diện cho hình ảnh sepia, có cùng hình dạng với hình ảnh đầu vào, với 3 kênh màu (R, G, B) và các giá trị từ 0 đến 255.
- Mô tả cách hoạt động

Đầu tiên, ta sẽ trích xuất các giá trị pixels của từng kênh màu R, G, và B từ hình ảnh RGB.

Sau đó, ta sẽ tính toán các giá trị sepia cho từng kênh màu bằng cách sử dụng công thức chuyển đổi:

- Đỏ: $tr = 0.393 \times R + 0.769 \times G + 0.189 \times B$
- Xanh lá: $tg = 0.349 \times R + 0.686 \times G + 0.168 \times B$
- Xanh dương: $tb = 0.272 \times R + 0.534 \times G + 0.131 \times B$

Tiếp theo, tạo mảng sepia bằng cách dùng hàm `np.stack` để kết hợp các kênh màu đã được tính toán thành một hình ảnh sepia đầy đủ. Để làm được điều này, đầu tiên ta sẽ tạo danh sách các mảng 2D [tr, tg, tb] đại diện cho các kênh màu đỏ, xanh lá và xanh dương đã được tính toán sepia. Sau đó, ta sẽ xếp chồng các mảng 2D này lên trục thứ ba là trục channels để tạo ra một mảng 3D với các kênh màu nằm dọc theo trục này có kích thước về độ cao và chiều rộng giống với mảng numpy ban đầu của hình ảnh RGB.

Ta sẽ dùng hàm `np.clip()` để cắt giá trị mỗi điểm ảnh nhằm giới hạn giá trị đó nằm trong đoạn [0; 255] bởi vì giá trị của mỗi điểm ảnh trong mảng numpy chỉ từ 0 đến 255

Cuối cùng, chuyển đổi mảng từ kiểu dữ liệu int16 trở lại uint8 để đảm bảo hình ảnh đầu ra ở định dạng 8-bit.

10. Hàm tính tích chập theo kernel cho trước `apply_kernel(...)`

Hàm `apply_kernel` thực hiện phép toán tích chập (convolution) trên một hình ảnh bằng cách sử dụng một kernel (hoặc bộ lọc).

- Tham số đầu vào: hàm này sẽ nhận 2 tham số đầu vào
 - “image”: làm một mảng numpy đại diện cho hình ảnh RGB với các giá trị từ 0 đến 255.
 - “kernel”: mảng Numpy 2 chiều, có kích thước (k_height, k_width), trong đó k_height là chiều cao của kernel và k_width là chiều rộng của kernel.

- Kết quả đầu ra: một mảng Numpy 3 chiều có cùng kích thước với hình ảnh đầu vào, với các giá trị pixel đã được xử lý qua phép toán tích chập với kernel.
- Mô tả cách hoạt động
Đầu tiên, lấy kích thước của hình ảnh (img_height, img_width) và kernel (k_height, k_width).
Tính kích thước padding cần thêm vào hình ảnh để đảm bảo kết quả tích chập có cùng kích thước với hình ảnh gốc. Padding là một phương pháp thêm các hàng và cột xung quanh biên của hình ảnh để giữ nguyên kích thước sau khi áp dụng kernel. Sử dụng np.pad để thêm padding vào hình ảnh. Padding ở đây là các giá trị 0, thêm vào các cạnh của hình ảnh.
Tạo một mảng new_image với cùng kích thước như hình ảnh gốc để lưu kết quả sau khi áp dụng kernel.
Tiếp theo ta sẽ áp dụng kernel:
 - Lặp qua từng pixel trong hình ảnh (theo chiều cao i và chiều rộng j).
 - Với mỗi pixel, lặp qua từng kênh màu (R, G, B) trong hình ảnh.
 - Tính tổng tích chập cho mỗi kênh màu bằng cách nhân kernel với một vùng con của hình ảnh đã được padding. Vùng con này có cùng kích thước với kernel và có tâm là pixel hiện tại.
 - Giá trị tích chập này sẽ thay thế giá trị pixel hiện tại trong new_image.Cuối cùng ta sẽ sử dụng np.clip để đảm bảo tất cả các giá trị pixel trong new_image nằm trong khoảng từ 0 đến 255 và chuyển đổi new_image về định dạng uint8 để đảm bảo rằng tất cả các giá trị pixel đều là số nguyên 8-bit không dấu.

11. Hàm làm mờ ảnh blur_image (...)

Hàm **blur_image** áp dụng hiệu ứng làm mờ cho hình ảnh bằng cách sử dụng một kernel tích chập (convolutional kernel) để trung bình các giá trị pixel trong một vùng lân cận 5x5.

- Tham số đầu vào: “image” mảng numpy đại diện cho hình ảnh RGB, với hình dạng (height, width, 3) và các giá trị pixels nằm trong đoạn [0, 255].
- Kết quả đầu ra: một mảng numpy có hình dạng giống với hình ảnh đầu vào với hiệu ứng làm mờ được áp dụng
- Mô tả cách hoạt động

Ta sẽ áp dụng phương pháp box blur với $r = 5$ để tạo ra một kernel áp dụng:

- np.ones((5, 5)) tạo ra một ma trận 5x5 với tất cả các giá trị bằng 1.
- Chia mỗi phần tử của ma trận này cho 25 (tổng số phần tử trong ma trận 5x5) tạo ra kernel làm mờ. Mỗi giá trị trong kernel là $1/25$, tức là 0.04.
- Kernel này sẽ trung bình giá trị pixel trong khu vực 5x5 xung quanh mỗi pixel, giúp làm mờ hình ảnh bằng cách giảm sự khác biệt giữa các pixel lân cận.

Sau đó, ta sẽ áp dụng hàm `apply_kernel` để thực hiện phép toán tích chập giữa hình ảnh và kernel.

12. Hàm làm sắc nét ảnh `sharpen_image (...)`

Hàm `sharpen_image` áp dụng hiệu ứng làm sắc nét (sharpening) cho hình ảnh bằng cách sử dụng một kernel tích chập đặc biệt để làm nổi bật các cạnh và chi tiết trong hình ảnh.

- Tham số đầu vào: “image” mảng numpy đại diện cho hình ảnh RGB, với hình dạng (height, width, 3) và các giá trị pixels nằm trong đoạn [0, 255].
- Kết quả đầu ra: một mảng numpy có hình dạng giống với hình ảnh đầu vào với hiệu ứng làm sắc nét cạnh được áp dụng
- Mô tả cách hoạt động

Đầu tiên, ta sẽ tạo ra một kernel làm sắc nét ảnh:

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

- Kernel này có tác dụng làm nổi bật các cạnh trong hình ảnh bằng cách khuếch đại các sự khác biệt giữa các pixel lân cận. Cụ thể:
 - [-1, -1, -1] ở các cạnh của ma trận làm giảm giá trị pixel xung quanh pixel trung tâm.
 - “5” tại điểm trung tâm làm tăng giá trị pixel tại vị trí đó.
 - Kết quả là pixel tại trung tâm sẽ có giá trị cao hơn nếu có sự khác biệt rõ rệt với các pixel xung quanh, làm cho các cạnh và chi tiết trở nên rõ ràng hơn.

Sau đó, ta sẽ áp dụng hàm `apply_kernel` để thực hiện phép toán tích chập giữa hình ảnh và kernel.

13. Hàm cắt ảnh theo kích thước (cắt ở trung tâm) `crop_center (...)`

Hàm `crop_center` cắt một hình ảnh từ giữa để có kích thước xác định trước

- Tham số đầu vào: hàm này sẽ nhận 3 tham số
 - “image”: mảng numpy đại diện cho hình ảnh. Mảng này có thể là hình ảnh xám hoặc hình ảnh RGB (3 kênh màu).
 - “new_width”: chiều rộng của hình ảnh sau khi cắt.
 - “new_height”: chiều cao của hình ảnh sau khi cắt.
- Kết quả đầu ra: là mảng NumPy (np.ndarray) hình ảnh với kích thước như hình ảnh ban đầu sau khi được cắt ở trung tâm.

- Mô tả cách hoạt động

Đầu tiên, ta sẽ lấy kích thước từ hình ảnh với height và width là chiều cao và chiều rộng của hình ảnh, lấy từ mảng numpy.

Sau đó, ta sẽ tính toán các tọa độ cần cắt:

- left và top là tọa độ của góc trên bên trái của vùng cắt.

- o right và bottom là tọa độ của góc dưới bên phải của vùng cắt.
- o $(width - new_width) // 2$: Tính khoảng cách từ cạnh trái của hình ảnh đến cạnh trái của vùng cắt.
- o $(height - new_height) // 2$: Tính khoảng cách từ cạnh trên của hình ảnh đến cạnh trên của vùng cắt.
- o $(width + new_width) // 2$: Tính khoảng cách từ cạnh trái của hình ảnh đến cạnh phải của vùng cắt.
- o $(height + new_height) // 2$: Tính khoảng cách từ cạnh trên của hình ảnh đến cạnh dưới của vùng cắt.

Tiếp theo, ta sẽ dùng phép slicing image [top:bottom, left:right] chọn các pixel của hình ảnh cần giữ lại, cụ thể:

- o “top:bottom” chọn các hàng từ “top” đến “bottom – 1”.
- o “left:right” chọn các cột từ “left” đến “right – 1”.

Cuối cùng thì chuyển đổi bức ảnh về dạng uint8 (số nguyên không dấu 8-bit) để có thể biểu diễn được như mọi bức ảnh khác

14. Hàm cắt ảnh theo khung hình tròn crop_circle (...)

Hàm “**crop_circle**” này cắt một hình ảnh thành hình tròn, tạo ra một hiệu ứng khung hình tròn.

- Tham số đầu vào: “image” mảng numpy đại diện cho hình ảnh RGB, với hình dạng (height, width, 3) và các giá trị pixels nằm trong đoạn [0, 255].
- Kết quả đầu ra: mảng NumPy đại diện cho hình ảnh được cắt theo khung hình tròn. Các pixel nằm ngoài hình tròn được đặt thành màu đen (0).
- Mô tả cách hoạt động

Đầu tiên ta sẽ lấy kích thước về chiều cao và chiều rộng của hình ảnh và tạo lưới tọa độ y, x (2 mảng 2D) cho toàn bộ hình ảnh.

Sau đó ta sẽ tính toán tọa độ của tâm hình ảnh bằng cách lấy trung điểm của chiều cao và chiều rộng.

Ta sẽ tính bán kính của hình tròn sao cho nó phù hợp với kích thước nhỏ hơn giữa chiều cao và chiều rộng của hình ảnh. Cụ thể, nếu kích thước ảnh đầu vào là hình vuông thì bán kính sẽ bằng khoảng từ tâm đến các cạnh. Còn nếu ảnh là hình chữ nhật, thì bán kính sẽ bằng độ dài nhỏ nhất từ tâm cho đến các cạnh.

Ta sẽ tạo mặt nạ hình tròn bằng cách tính toán các pixel nằm trong khoảng cách bán kính từ tâm. Các pixel nằm trong hình tròn có giá trị True, còn các pixel ngoài hình tròn có giá trị False. Điều kiện để xác định các điểm nằm trong hình tròn sẽ được xác định là dựa vào phương trình đường tròn, tức là: $(y - center_y)^2 + (x - center_x)^2 \leq radius^2$.

Áp dụng mặt nạ để đặt giá trị của các điểm ảnh nằm bên ngoài vùng hình tròn thành 0 (biến thành màu đen) và giữ nguyên ảnh với các vùng nằm ở trong hình tròn.

Cuối cùng, trả về hình ảnh đã được cắt dưới dạng np.array (result).

15. Hàm cắt ảnh theo khung hai hình elip chéo nhau `crop_elliptical_cross(...)`

Hàm “`crop_elliptical_cross`” được thiết kế để cắt hình ảnh bằng cách áp dụng một mặt nạ gồm hai hình ellipse chồng lên nhau một cách đối xứng.

- Tham số đầu vào: “`image`” mảng numpy đại diện cho hình ảnh RGB, với hình dạng (height, width, 3) và các giá trị pixels nằm trong đoạn [0, 255].
- Kết quả đầu ra: mảng NumPy đại diện cho hình ảnh được cắt theo khung 2 elip chéo nhau. Các pixel nằm ngoài hình elip chéo nhau được đặt thành màu đen (0).
- Mô tả cách hoạt động

Đầu tiên ta sẽ lấy kích thước về chiều cao và chiều rộng của hình ảnh và tạo lưới tọa độ y, x (2 mảng 2D) cho toàn bộ hình ảnh.

Sau đó ta sẽ tính toán tọa độ của tâm hình ảnh bằng cách lấy trung điểm của chiều cao và chiều rộng.

Ta sẽ tính bán kính của hình tròn để chèn hình elip vào sao cho nó phù hợp với kích thước nhỏ hơn giữa chiều cao và chiều rộng của hình ảnh. Cụ thể, nếu kích thước ảnh đầu vào là hình vuông thì bán kính sẽ bằng khoảng từ tâm đến các cạnh. Còn nếu ảnh là hình chữ nhật, thì bán kính sẽ bằng độ dài nhỏ nhất từ tâm cho đến các cạnh.

Ta sẽ tính bán trục chính (s_{major}) và bán trục phụ (s_{minor}) của hình elip.

$$s_{major} = width$$

$$s_{minor} = \frac{width}{\sqrt{2} + 1}$$

Ta sẽ tính $dist1$ và $dist2$ là các giá trị được tính toán để đánh giá khoảng cách của mỗi pixel so với trung tâm hình ảnh theo các hướng khác nhau.

$$dist1 = (x - y_{center}) + (y - x_{center})$$

$$dist2 = (x - y_{center}) - (y - x_{center})$$

Sau đó ta sẽ tính các khoảng cách từ một pixel đến tâm của của từng elip là:

$$mask1 = \frac{-dist2^2}{s_{major} \times \sqrt{2}} + \frac{dist1^2}{s_{minor} \times \sqrt{2}}$$

$$mask2 = \frac{dist1^2}{s_{major} \times \sqrt{2}} + \frac{-dist2^2}{s_{minor} \times \sqrt{2}}$$

Tạo các mặt nạ boolean, trong đó các giá trị True cho các pixel nằm trong các hình ellipse.

Áp Dụng Mặt Nạ và Cắt Hình Ảnh: `ellipses_img[~(mask1 | mask2)] = 0`: Đặt giá trị pixel ngoài các hình ellipse thành 0, giữ lại chỉ những phần bên trong hình ellipse.

Cuối cùng sẽ trả hình ảnh hoàn chỉnh sau khi cắt theo 2 khung elip chéo nhau.

16. Hàm phóng to hình ảnh `zoom_in(...)`

Hàm `zoom_in` tăng kích thước của một hình ảnh theo một tỷ lệ cho trước.

- Tham số đầu vào: hàm này nhận hai tham số
 - “image”: mảng ảnh đầu vào. Có thể là ảnh xám hoặc ảnh RGB (3 kênh).
 - “factor”: tỷ lệ phóng to. Hình ảnh sẽ được tăng kích thước theo tỷ lệ này.
- Kết quả đầu ra: Mảng ảnh đã được phóng to. Kích thước của ảnh sẽ được tăng theo tỷ lệ phóng to.
- Mô tả cách hoạt động
Ta sẽ lấy chiều cao và chiều rộng của hình ảnh gốc bằng `image.shape[:2]`
Kích thước mới của ảnh đã phóng to được tính bằng cách nhân kích thước gốc với tỷ lệ phóng to. Điều này tạo ra `new_height` và `new_width`.
Một mảng rỗng `zoomed_image` được khởi tạo với các giá trị bằng không, có kích thước mới và số kênh giống như ảnh gốc.
Ta sẽ điền giá trị pixel cho ảnh đã phóng to:
 - Vòng lặp lồng nhau đi qua từng pixel của ảnh đã phóng to.
 - Với mỗi pixel (i, j) trong ảnh đã phóng to, giá trị pixel tương ứng từ ảnh gốc được lấy. Tọa độ trong ảnh gốc được tính bằng cách chia tọa độ trong ảnh đã phóng to (i, j) cho tỷ lệ phóng to.

17. Hàm thu nhỏ hình ảnh `zoom_out (...)`

Hàm `zoom_out` giảm kích thước của một hình ảnh theo một tỷ lệ cho trước. Điều này được thực hiện bằng cách thay đổi kích thước của hình ảnh và nội suy giá trị các pixel.

- Tham số đầu vào: hàm này nhận hai tham số
 - “image”: Mảng ảnh đầu vào. Có thể là ảnh xám hoặc ảnh RGB (3 kênh).
 - “factor”: Tỷ lệ thu nhỏ. Hình ảnh sẽ được giảm kích thước theo tỷ lệ này.
- Kết quả đầu ra: Mảng ảnh đã được thu nhỏ. Kích thước của ảnh sẽ được giảm theo tỷ lệ phóng to.
- Mô tả cách hoạt động
Ta sẽ lấy chiều cao và chiều rộng của hình ảnh gốc bằng `image.shape[:2]`.
Kích thước mới của ảnh đã thu nhỏ được tính bằng cách chia kích thước gốc cho tỷ lệ phóng to. Điều này tạo ra `new_height` và `new_width`.
Một mảng rỗng `zoomed_image` được khởi tạo với các giá trị bằng không, có kích thước mới và số kênh giống như ảnh gốc.
Ta sẽ điền giá trị pixel cho ảnh đã thu nhỏ:
 - Vòng lặp lồng nhau đi qua từng pixel của ảnh đã thu nhỏ.
 - Với mỗi pixel (i, j) trong ảnh đã thu nhỏ, giá trị pixel tương ứng từ ảnh gốc được lấy. Tọa độ trong ảnh gốc được tính bằng cách nhân tọa độ trong ảnh đã thu nhỏ (i, j) với tỷ lệ phóng to.

18. Hàm `main`

- Tham số đầu vào: None
- Kết quả đầu ra: None
- Mô tả cách hoạt động
Người dùng sẽ nhập tên hình ảnh theo cú pháp “name_image.jpg”.
Chú ý: tên hình ảnh phải kèm thêm kiểu định dạng

Thử đọc hình ảnh từ tệp với tên người dùng cung cấp. Nếu không tìm thấy tệp, thông báo lỗi sẽ được hiển thị và yêu cầu nhập lại tên tệp.

Nếu có lỗi trong khi đọc hình ảnh, thông báo lỗi sẽ được hiển thị và yêu cầu nhập lại tên tệp.

Chương trình sẽ hiện các lựa chọn như sau:

1. Điều chỉnh độ sáng
2. Điều chỉnh độ tương phản
3. Lật hình ảnh (ngang/dọc)
4. Chuyển đổi hình ảnh RGB sang grayscale/sepia
5. Làm mờ/làm sắc nét hình ảnh
6. Cắt hình ảnh theo kích thước (cắt trung tâm)
7. Cắt hình ảnh theo khung (hình tròn/hình elip)
8. Phóng to/thu nhỏ hình ảnh
9. Áp dụng tất cả các chức năng
10. Thay đổi tệp hình ảnh
11. Thoát khỏi chương trình

Dựa trên sự lựa chọn của người dùng, thực hiện các chức năng xử lý hình ảnh:

- Điều chỉnh độ sáng và độ tương phản: Người dùng nhập giá trị scalar và kết quả được lưu và hiển thị.
- Lật hình ảnh: Người dùng chọn lật ngang hoặc dọc, kết quả được lưu và hiển thị.
- Chuyển đổi màu sắc: Chuyển đổi hình ảnh sang grayscale và sepia, kết quả được lưu và hiển thị.
- Làm mờ/làm sắc nét: Hình ảnh được làm mờ và làm sắc nét, kết quả được lưu và hiển thị.
- Cắt hình ảnh: Người dùng nhập kích thước cho cắt trung tâm hoặc chọn kiểu khung để cắt theo khung, kết quả được lưu và hiển thị.
- Phóng to/thu nhỏ: Người dùng chọn phóng to hoặc thu nhỏ, kết quả được lưu và hiển thị.
- Áp dụng tất cả các chức năng: Kết thúc vòng lặp xử lý hình ảnh.

Nếu người dùng chọn thay đổi tệp hình ảnh, vòng lặp xử lý hình ảnh sẽ kết thúc và yêu cầu người dùng nhập lại tên tệp mới.

Kết thúc chương trình: Nếu người dùng chọn thoát khỏi chương trình, hàm main sẽ kết thúc và chương trình sẽ thoát.

TÀI LIỆU THAM KHẢO

Trong quá trình thực hiện **đồ án thực hành số 2 - Image Processing**, nhằm phục vụ cho quá trình thực hiện được suôn sẻ và tăng hiệu suất cũng như độ chính xác thuật toán, em đã tham khảo và sử dụng một số tài liệu mở và điện tử sau đây:

- **Tài liệu tham khảo về nội dung báo cáo:**

[1] [Kernel \(image processing\)](#) (ngày truy cập: 20/07/2024)

[2] [Kỹ Thuật Grayscale và Nhi Phân Hoá Ảnh \(Adaptive Threshold\)](#) (ngày truy cập: 21/07/2024)

[3] [How to convert a color image into sepia image](#) (ngày truy cập: 22/07/2024)