ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

Đồ án 2: Image Processing

Toán ứng dụng và thống kê

< Lab01 >

Huỳnh Sỉ Kha - 21127734
Lớp: 21CLC02

Giảng viên:

Vũ Quốc Hoàng
Nguyễn Văn Quang Huy
Lê Thanh Tùng
Phan Thị Phương Uyên

1st August 2023

Mục lục

1 Giới thiệu thuật toán xử lý hình ảnh: . . . 3
2 Một số kỹ thuật sử dụng trong đồ án . . . 3
2.1 Slicing và Indexing . . . 3
2.2 Broadcasting . . . 3
3 Các thuật toán xử lý hình ảnh trong đồ án . . . 4
3.1 Công cụ và thư viện hỗ trợ . . . 4
3.2 Thay đổi độ sáng cho ảnh: . . . 4
3.3 Thay đổi độ tương phản cho ảnh . . . 6
3.4 Lật ảnh . . . 8
3.5 Chuyển sang ảnh đơn sắc . . . 10
3.6 Chuyển sang ảnh Sepia . . . 11
3.7 Làm mờ hình ảnh . . . 13
3.8 Làm sắc nét hình ảnh . . . 15
3.9 Cắt ảnh ở trung tâm theo kích thước . . . 16
3.10 Cắt ảnh tròn theo kích thước . . . 18
3.11 Cắt ảnh elip . . . 20
4 Các hàm chức năng trong chương trình: . . . 24
4.1 Lấy thông tin của ảnh . . . 24
4.2 Danh sách các kênh màu của hình ảnh . . . 24
4.3 Đọc ảnh và ép kiểu sang Numpy array . . . 24
4.4 Lưu ảnh . . . 25
4.5 Hiển thị ảnh . . . 25
4.6 Khởi tạo chú thích cho ảnh . . . 26
4.7 Gọi hàm thực hiện chức năng . . . 26
4.8 Hàm main . . . 27
5 Tài liệu tham khảo: . . . 28

Thứ tự	**Thuật toán**	**Thời gian (ms)**	**Mức độ hoàn thành**
1	Điều chỉnh độ sáng	26	Hoàn thành yêu cầu Cân bằng màu sắc giúp tăng giảm độ sáng với màu sắc đẹp hơn.
2	Điều chỉnh độ tương phản	5	Hoàn thành yêu cầu.
3	Lật ảnh	1	Hoàn thành yêu cầu Tối ưu thời gian.
4	Chuyển sang ảnh xám	3	Hoàn thành yêu cầu, giữ lại được độ bóng của bức ảnh.
5	Chuyển sang ảnh Sepia	11	Hoàn thành yêu cầu
6	Làm mờ	242	Có thể tuỳ chỉnh mức độ làm mờ, độ mịn. Giảm đáng kể thời gian thực thi.
7	Làm sắc nét	251	Có thể tuỳ chỉnh mức độ làm sắc nét, độ mịn. Giảm đáng kể thời gian thực thi.
8	Cắt ảnh ở trung tâm	0	Hoàn thành yêu cầu
9	Cắt ảnh elip	9	Ảnh kết quả thoả yêu cầu đối với hình vuông
10	Hàm main		Hoàn thành yêu cầu

1 Giới thiệu thuật toán xử lý hình ảnh:

Thuật toán xử lý hình ảnh là một tập hợp các phương pháp và kỹ thuật được sử dụng để xử lý và phân tích hình ảnh số (digital image) trong môi trường số học. Hình ảnh số là một tập hợp các điểm ảnh (pixels), mỗi điểm ảnh chứa thông tin về màu sắc tại một vị trí cụ thể trong hình ảnh.
Các bước chính trong quá trình xử lý hình ảnh bao gồm:

1. Thu thập hình ảnh: Đây là bước thu thập dữ liệu hình ảnh từ nguồn đầu vào, như máy ảnh, camera, hoặc các bộ cảm biến.
2. Tiền xử lý (Preprocessing): Bước tiền xử lý là quá trình làm sạch, cân bằng màu trong hình ảnh trước khi áp dụng các thuật toán chỉnh sửa. Bước này giúp hạn chế sai lệch về màu sắc so với ảnh gốc sau khi qua chỉnh sửa.
3. Xử lý và phân tích (Processing and Analysis): Đây là giai đoạn áp dụng các thuật toán xử lý hình ảnh để thực hiện các nhiệm vụ cụ thể, chẳng hạn như xoay, điều chỉnh độ sáng, độ tương phản, chuyển sang chế độ màu đơn sắc, ...
4. Kết quả (Output): Kết quả của quá trình xử lý hình ảnh là một hình ảnh mới, đã được chỉnh sửa theo nhu cầu của người dùng.

2 Một số kỹ thuật sử dụng trong đồ án

2.1 Slicing và Indexing

Slicing là kỹ thuật cho phép trích xuất một phần của chuỗi, danh sách, hay mảng bằng cách chỉ định một phạm vi thông qua các chỉ số.
Cú pháp `array[start:stop:step]`. Trong đó:

- **'array'**: Có thể là một chuỗi, danh sách (list), hoặc mảng NumPy mà người lập trình cần trích xuất.
- **'start'**: Chỉ số, vị trí của phần tử đầu tiên muốn lấy. Nếu không được xác định, mặc định là 0.
- **'stop'**: Chỉ số, vị trí của phần tử cuối cùng (không bao gồm) muốn lấy. Nếu không được xác định, mặc định là độ dài của mảng.
- **'step'**: Bước nhảy giữa các chỉ số. Nếu không được xác định, mặc định là 1.

2.2 Broadcasting

Broadcasting là một tính năng hữu ích của thư viện NumPy trong Python. Broadcasting cho phép thực hiện các phép toán (cộng, trừ, nhân, chia, ...) giữa các mảng có kích thước và hình dạng khác nhau mà không cần thực hiện việc mở

rộng (expand) kích thước của các mảng.

Một số quy tắc của kỹ thuật broadcasting:

1. Nếu hai mảng có số chiều (dimensions) khác nhau, thì mảng có số chiều ít hơn sẽ được mở rộng bằng việc thêm các chiều có kích thước 1 ở phía trước (ví dụ: từ (5,) thành (1, 5)).
2. Nếu hai mảng có số chiều bằng nhau, nhưng kích thước của các chiều không giống nhau, thì mảng có kích thước bằng 1 sẽ được mở rộng để có kích thước giống nhau với mảng còn lại.
3. Nếu sau khi mở rộng các mảng, các kích thước của các chiều vẫn không tương thích, lúc này sẽ xảy ra lỗi.

3 Các thuật toán xử lý hình ảnh trong đồ án

3.1 Công cụ và thư viện hỗ trợ

a Thư viện hỗ trợ

- numpy: Thực hiện các phép tính trên ma trận.
- PIL: Đọc, lưu ảnh.
- matplotlib: Hiển thị ảnh kết quả.
- time: Đo thời gian chạy các hàm

b Công cụ hỗ trợ

- Thiết bị chạy chương trình: Asus ROG Zephyrus G14 (8GB, 512 GB).
- Hệ điều hành: Window 10.
- Hình ảnh kiểm thử: Lena Forsén (512x512).

3.2 Thay đổi độ sáng cho ảnh:

```
def change_brightness(image, alpha=5, gamma=0.5)
```

a Tham số truyền vào

- **'image'** (ndarray): Hình ảnh truyền vào ở dạng mảng nhiều chiều.
- **'alpha'** (int): Hệ số điều chỉnh độ sáng. Giá trị dương làm tăng độ sáng cho ảnh, ngược lại, giá trị âm làm ảnh tối hơn. Giá trị mặc định là 5.
- **'gamma'** (int): Hệ số cân bằng màu sắc của ảnh. Giá trị mặc định là 0,5.

b Tham số trả về

- Hàm trả về một hình ảnh mới (ndarray) sau khi đã thay đổi độ sáng và cân bằng màu sắc.

c Ý tưởng thực hiện

- Các điểm ảnh được biểu diễn bằng một số tự nhiên có giá trị từ $0 \rightarrow 255$. Trong đó, 0 đại diện cho màu đen, 255 đại diện màu trắng.
- Nghĩa là, giá trị màu càng gần 255 thì càng sáng, ngược lại, càng gần 0 thì càng tối. Thay đổi độ sáng nghĩa là thay đổi giá trị màu về gần 1 trong 2 cực sao cho phù hợp mong muốn của người dùng.

d Mô tả hàm

1. `255 * (image / 255) ** gamma`: Cân bằng màu sắc trước khi thực hiện phép toán trên ma trận.
2. `np.clip(array, a_min, a_max)`: Thực hiện các phép toán trên ma trận với giá trị nằm trong một khoảng đã được xác định.
   - **'array'** (ndarray): Biến array được truyền vào hình ảnh đã qua cân bằng màu sắc và được cộng hoặc trừ đi một lượng **'alpha'**.
   - **'a_min'** (int): Giá trị a_min biểu thị cho giá trị nhỏ nhất trong mảng kết quả. Các giá trị trong mảng nhỏ hơn a_min sẽ được quy về giá trị này.
   - **'a_max'** (int): Giá trị a_max biểu thị cho giá trị lớn nhất trong mảng kết quả. Các giá trị trong mảng lớn hơn a_max sẽ được quy về giá trị này.
3. **Kết quả** `new_image.astype(np.uint8)`: Đảm bảo kiểu dữ liệu của hình ảnh sau khi thay đổi độ sáng là số nguyên không dấu 8-bit trong NumPy.

e Kết quả

- Tăng sáng cho ảnh với $\sigma = 50$ và $\gamma = 1.2$.

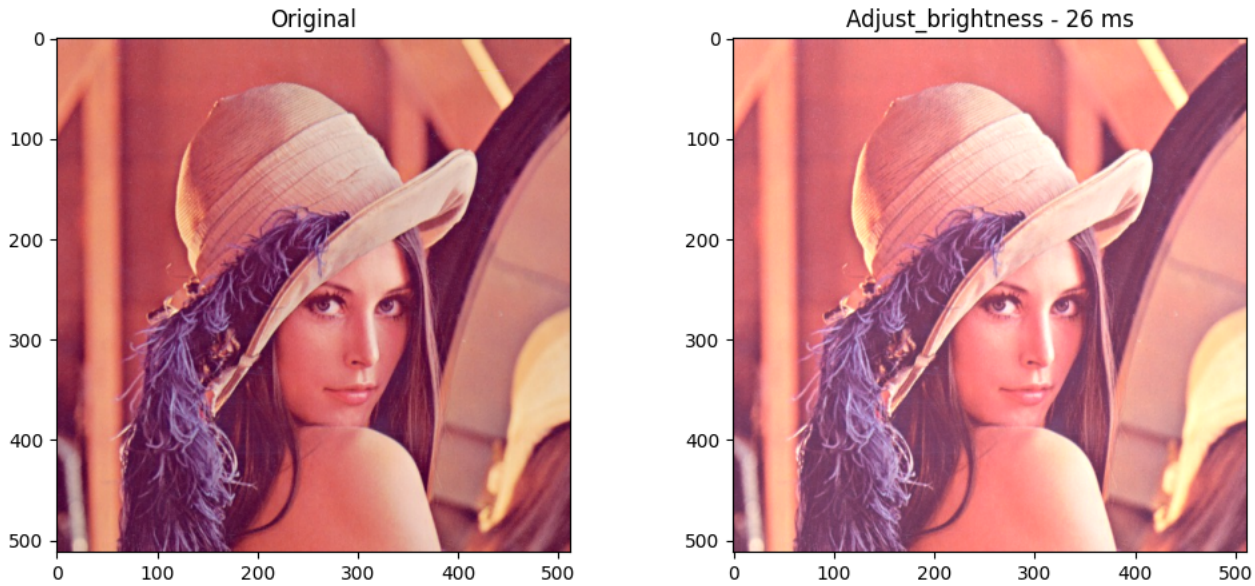

Figure 1: Tăng sáng với $\sigma = 50$ và $\gamma = 1.2$

f Nhận xét

- Việc cân bằng màu sắc trước khi thực hiện thay đổi độ sáng sẽ giúp ảnh sáng hoặc tối hơn nhưng vẫn giữ được các sắc độ của bức ảnh, đồng thời cũng hạn chế việc sai lệch màu sắc khi tăng sáng với σ quá lớn.
- Tính chất phi tuyến tính: Hàm **'adjust_brightness'** thay đổi độ sáng của hình ảnh bằng cách áp dụng các biến đổi phi tuyến tính trên từng điểm ảnh. Điều này có thể dẫn đến hiện tượng mất thông tin hoặc hiện tượng mờ hình ảnh khi độ sáng thay đổi quá lớn, đặc biệt là nếu giá trị alpha và gamma được sử dụng quá cao hoặc quá thấp.

3.3 Thay đổi độ tương phản cho ảnh

```
def change_contrast(image, alpha=1.5)
```

a Tham số truyền vào

- **'image'** (ndarray): Hình ảnh truyền vào ở dạng mảng nhiều chiều.
- **'alpha'** (int): Hệ số điều chỉnh độ tương phản. $alpha > 1$ làm tăng độ tương phản, ngược lại, $alpha < 1$ làm giảm độ tương phản.

b Tham số trả về

- Hàm trả về một hình ảnh mới (ndarray) sau khi đã thay đổi độ tương phản.

c Ý tưởng thực hiện

- Độ tương phản hay độ chênh lệch về giá trị giữa các điểm ảnh. Thay đổi độ tương phản nghĩa là làm tăng lên hoặc giảm xuống độ chênh lệch này.
- Lấy bức ảnh ban đầu nhân cho một lượng sao cho độ chênh lệch màu bị thay đổi sẽ làm thay đổi độ tương phản của bức ảnh.

d Mô tả hàm

1. `np.clip(array, a_min, a_max)`: Thực hiện các phép toán trên ma trận với giá trị nằm trong một khoảng đã được xác định.
   - **'array'** (ndarray): Biến array được truyền vào hình ảnh đã được nhân với một lượng **'alpha'**.
   - **'a_min'** (int): Giá trị a_min biểu thị cho giá trị nhỏ nhất trong mảng kết quả. Các giá trị trong mảng nhỏ hơn a_min sẽ được quy về giá trị này.
   - **'a_max'** (int): Giá trị a_max biểu thị cho giá trị lớn nhất trong mảng kết quả. Các giá trị trong mảng lớn hơn a_max sẽ được quy về giá trị này.
2. **Kết quả** `new_image.astype(np.uint8)`: Đảm bảo kiểu dữ liệu của hình ảnh sau khi thay đổi độ tương phản là số nguyên không dấu 8-bit trong NumPy.

e Kết quả

- Tăng độ tương phản cho ảnh với $\alpha = 1{,}5$.

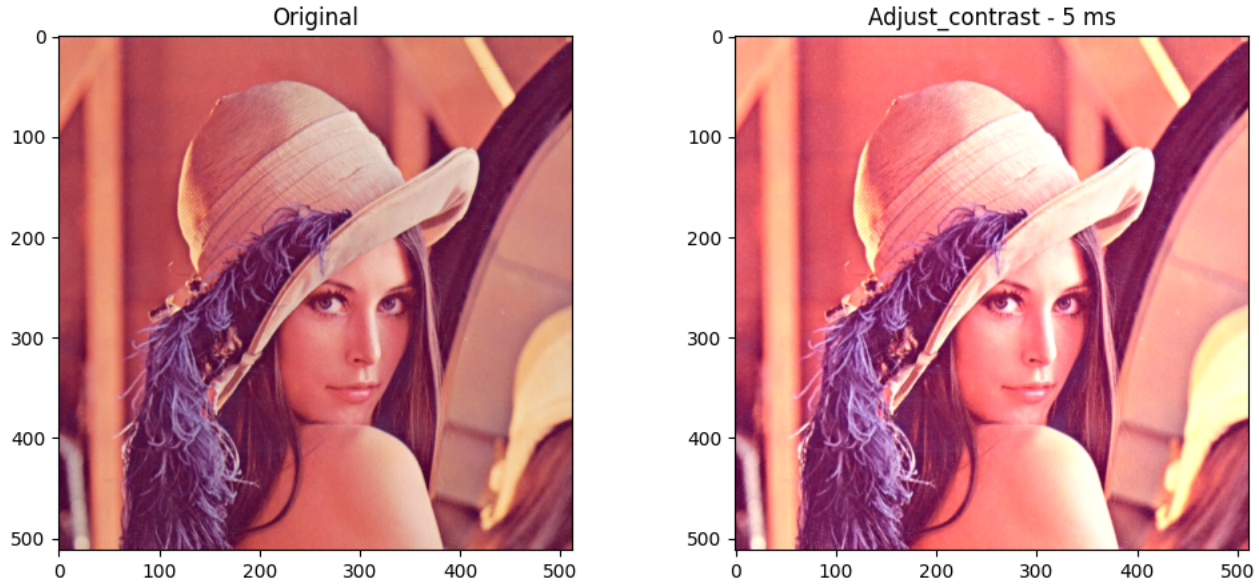

Figure 2: Tăng độ tương phản cho ảnh với $\alpha = 1{,}5$

f Nhận xét

- Độ tương phản tăng: Nếu alpha > 1, hàm sẽ làm tăng độ tương phản của hình ảnh. Các pixel có giá trị ban đầu tối thiểu hơn 127.5 (giá trị trung bình giữa 0 và 255) sẽ tăng đáng kể, trong khi các pixel có giá trị ban đầu nhỏ hơn 127.5 sẽ tăng ít hơn. Điều này làm tăng sự tách biệt giữa các vùng sáng và tối của hình ảnh.
- Ảnh không thể vượt quá [0, 255]: Việc giới hạn giá trị các điểm ảnh trong khoảng từ 0 đến 255 sẽ khiến ảnh cuối cùng trở nên bị giới hạn bởi phạm vi này. Điều này có thể dẫn đến hiện tượng mất thông tin hoặc hiện tượng tràn số (overflow) nếu giá trị alpha được sử dụng quá cao, và hiện tượng thiếu sáng nếu giá trị alpha được sử dụng quá thấp.

3.4 Lật ảnh

```
def rotate_image(image, rotate_type='vertical')
```

a Tham số truyền vào

- **'image'** (ndarray): Hình ảnh truyền vào ở dạng mảng nhiều chiều.
- **'rotate_type'** (sting): Chuỗi xác định kiểu xoay hình ảnh. Nếu là **'vertical'** thì hàm sẽ xoay hình ảnh theo chiều dọc. Nếu là **'horizontal'** thì hàm sẽ xoay hình ảnh theo chiều ngang.

b Tham số trả về

- Hàm trả về một hình ảnh mới (ndarray) sau khi đã xoay theo chiều dọc hoặc chiều ngang, tuỳ thuộc vào giá trị của biến **'rotate_type'**.

c Ý tưởng thực hiện

- Theo lý thuyết, lấy dòng hoặc cột điểm ảnh nằm ở vị trí giữa làm trục đối xứng, hoán đổi vị trí giữa dòng, cột đầu tiên với dòng, cột điểm ảnh cuối cùng, tiếp tục thực hiện với các dòng cột tiếp theo cho tới khi chạm tới trục giữa.
- Theo đồ án, sử dụng kỹ thuật slicing để tạo ra mảng (hình ảnh) với bước nhảy là -1 nghĩa là đọc ngược từ vị trí cuối đến đầu tiên. Kết quả thu được hình ảnh được xoay theo chiều dọc hoặc ngang so với ảnh gốc.

d Mô tả hàm

1. **'rotate_type = vertical'**: Thực hiện xoay ảnh theo chiều dọc.
   - **slicing '[:, ::-1]'**:

  - Chỉ số dòng: dấu (':') bên trái dấu phẩy sẽ giữ lại tất cả các dòng của mảng.
  - Chỉ số cột: dấu (**'::-1'**) bên phải dấu phẩy sẽ giữ lại tất cả các cột đồng thời đảo ngược thứ tự của chúng trong mảng.

2. **'rotate_type = horizontal'**: Thực hiện xoay ảnh theo chiều ngang.
   - Kỹ thuật **slicing '[::-1]'**: Giữ lại toàn bộ cột và dòng của mảng ban đầu, tuy nhiên các dòng sẽ được lấy theo thứ tự ngược lại.
3. **Kết quả** : Ảnh trả về được xoay ngang hoặc dọc so với ảnh ban đầu.

e Kết quả

- Lật ảnh theo trục dọc.

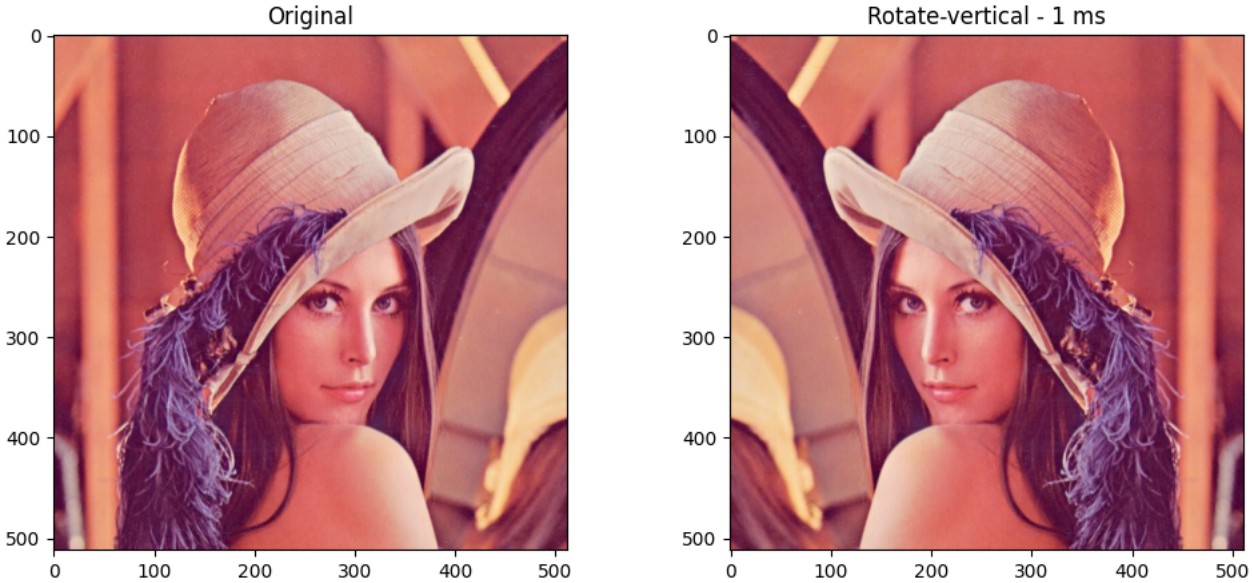

Figure 3: Lật ảnh theo trục dọc

f Nhận xét

- Hiệu suất: Hàm lật ảnh thực hiện các phép gán chỉ mục (indexing) đơn giản trên mảng numpy, do đó có thời gian thực hiện nhanh và hiệu suất tốt.
- Tính đảm bảo: Hàm không sử dụng bất kỳ vòng lặp (loop) hay các phép toán phức tạp nào, do đó có tính đảm bảo cao, không làm sai lệch giá trị màu so với ảnh gốc.

3.5 Chuyển sang ảnh đơn sắc

```
def convert_RGB_to_grayscale(image)
```

a Tham số truyền vào

- **'image'** (ndarray): Hình ảnh truyền vào ở dạng mảng nhiều chiều.

b Tham số trả về

- Hàm trả về một mảng 2 chiều biểu thị cho ảnh đơn sắc (ndarray).

c Ý tưởng thực hiện

- Trong không gian màu RGB, màu đỏ (R) có bước sóng dài hơn so với cả hai màu còn lại, và màu xanh lá cây (G) là màu có bước sóng ngắn hơn, và màu xanh dương (B) nằm ở giữa hai màu này. Do đó, để giảm đóng góp của màu đỏ và tăng đóng góp của màu xanh lá cây, và đặt đóng góp của màu xanh dương giữa hai màu này, ta sử dụng các trọng số tương ứng là 0.3 cho màu đỏ (R), 0.59 cho màu xanh lá cây (G), và 0.11 cho màu xanh dương (B).

d Mô tả hàm

1. Trích xuất các kênh màu từ hình ảnh RGB: Hàm **'slicing_color_channel(image)'** được sử dụng để trích xuất các kênh màu (R, G, B) từ hình ảnh RGB.
2. Tính toán hình ảnh xám: Bằng cách sử dụng trọng số cho mỗi kênh màu. Trong trường hợp này, hàm sử dụng trọng số 0.3 cho kênh màu đỏ (R), 0.59 cho kênh màu xanh lá cây (G), và 0.11 cho kênh màu xanh dương (B).
3. Giới hạn giá trị: Sau khi tính toán, hàm sử dụng **'np.clip()'** để đảm bảo rằng giá trị của hình ảnh xám nằm trong khoảng từ 0 đến 255.

e Kết quả

- Chuyển đổi từ ảnh màu RGB sang ảnh xám.

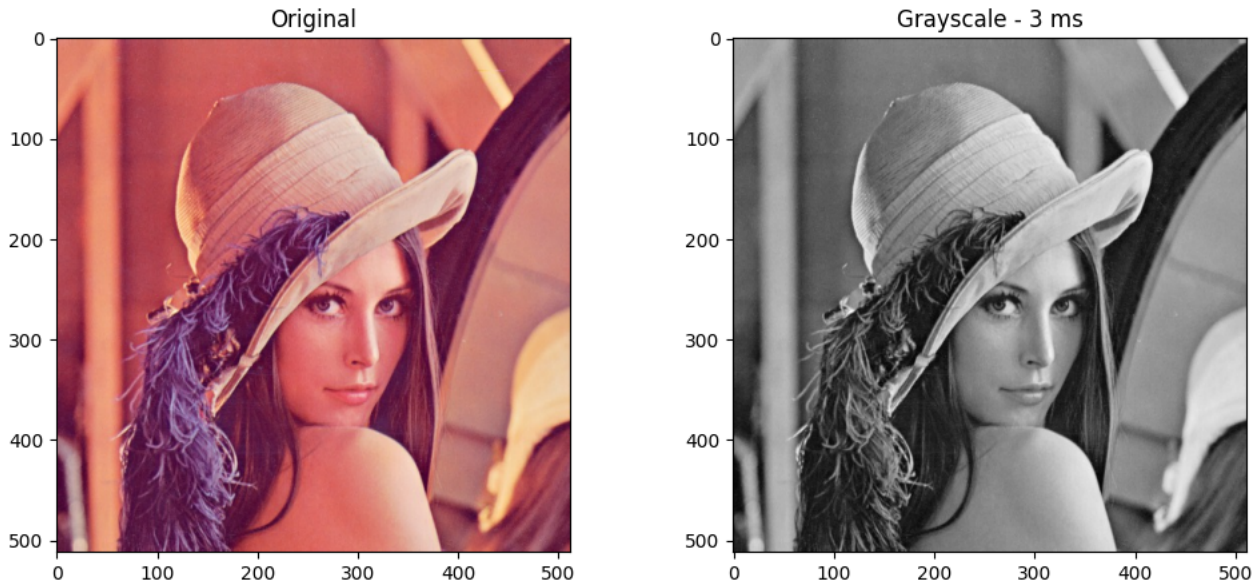

Figure 4: Chuyển đổi ảnh màu RGB thành ảnh xám

f Nhận xét

- Phương pháp có trọng số này giúp giảm thiểu hiện tượng mất mát màu sắc chi tiết và tạo ra ảnh xám với mức độ màu sắc hài hòa hơn so với phương pháp trung bình (average method) thông thường.
- Hình ảnh xám kết quả sẽ giữ lại một phần thông tin về màu sắc của hình ảnh gốc, nhưng các màu sắc trong ảnh xám được giảm độ bóng và sự chênh lệch màu sắc so với hình ảnh gốc.

3.6 Chuyển sang ảnh Sepia

```
def convert_RGB_to_Sepia(image)
```

a Tham số truyền vào

- **'image'** (ndarray): Hình ảnh truyền vào ở dạng mảng nhiều chiều.

b Tham số trả về

- Hàm trả về một mảng đại diện cho ảnh Sepia (ndarray).

c Ý tưởng thực hiện

- Sử dụng công thức sau để tạo ra ảnh màu Sepia (R, G, B đại diện cho giá trị của màu đỏ, xanh lá, xanh dương trong ảnh gốc):
  1. Màu đỏ mới = 0.393R + 0.769G + 0.189B
  2. Màu xanh lá mới = 0.349R + 0.686G + 0.168B

3. Màu xanh dương mới = 0.272R + 0.534G + 0.131B

d Mô tả hàm

1. `slicing image[:, :, i]`: Kỹ thuật slicing này sẽ giữ lại toàn bộ giá trị biểu thị màu ứng với i = 0 cho màu đỏ, 1 cho màu xanh lá và 2 cho màu xanh dương.
2. Ứng với mỗi giá trị màu đó, tính các màu R, G, B mới bằng công thức trên c. Kết hợp với hàm '**np.clip()**' để đảm bảo các giá trị màu nằm trong khoảng hợp lệ từ 0 → 255.
3. `np.stack((r, g, b), axis=2)`: Gán các thành phần màu đã được tính toán thành 1 hình ảnh mới. Trong trường hợp này, màu mới được xếp chồng lên nhau theo trục thứ 3 (axis=2).
4. **Kết quả** `new_image.astype(np.uint8)`: Đảm bảo kiểu dữ liệu của hình ảnh sau khi thay đổi chuyển sang đơn sắc là số nguyên không dấu 8-bit trong NumPy.

e Kết quả

- Chuyển đổi từ ảnh màu RGB sang ảnh Sepia.

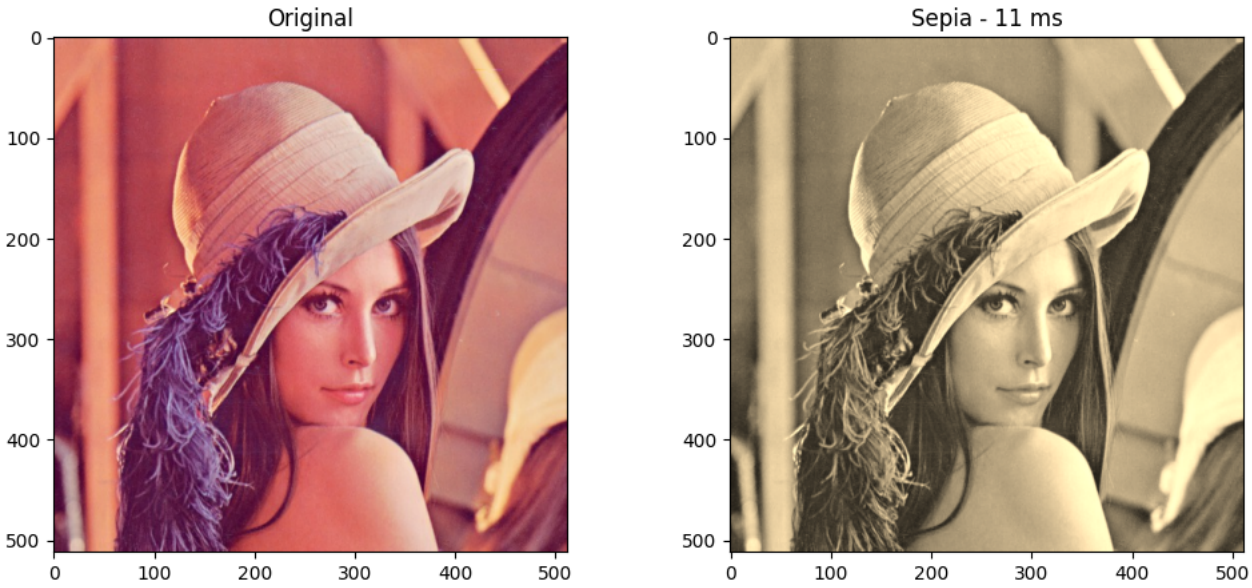

Figure 5: Chuyển đổi ảnh màu RGB thành ảnh Sepia

f Nhận xét

- Hình ảnh kết quả sẽ có hiệu ứng Sepia, tạo ra cảm giác cổ điển, cũ hơn cho hình ảnh. Hiệu ứng Sepia được tạo ra bằng cách thay đổi giá trị màu sắc của hình ảnh gốc.

- Hiệu ứng Sepia tạo ra cảm giác cổ điển và mang tính nghệ thuật, thường được sử dụng để làm đẹp hình ảnh và thêm tính chất lịch sử cho các tác phẩm nghệ thuật.

3.7 Làm mờ hình ảnh

```
def blur_image(image, kernel_size = 10, sigma = 5)
```

a Tham số truyền vào

- **'image'** (ndarray): Hình ảnh truyền vào ở dạng mảng nhiều chiều.
- **'kernel_size'** (int): Kích thước kernel Guassian. Mặc định là 10 nếu người dùng không cung cấp.
- **'sigma'** (int): Tham số sigma đại diện cho độ lệch chuẩn của phân phối Guassian. Mặc định là 5 nếu người dùng không cung cấp.

b Tham số trả về

- Một mảng NumPy biểu diễn ảnh sau khi được làm mờ bằng kernel Gaussian.

c Ý tưởng thực hiện

1. **Tính Guassian kernel**: Bước đầu tiên là tính toán kernel Gaussian 2D dựa trên tham số **'sigma'**. Kernel Gaussian được tạo ra từ hàm Gauss có công thức như sau:

$$kernel(x, y, \sigma) = \frac{e^{\frac{-(x^2+y^2)}{2\sigma^2}}}{2\pi\sigma^2} \tag{1}$$

   Trong đó, **'x'**, **'y'** là tọa độ trên lưới (meshgrid), và **'sigma'** là độ lệch chuẩn được cung cấp.

2. **Convolution (Tích chập)**: Sau khi tính toán kernel Gaussian, tiến hành tích chập (convolution) kernel này với ảnh đầu vào. Tích chập là một phép toán biến đổi giữa kernel và ảnh để tạo ra ảnh mới đã làm mờ.
   Việc tích chập được thực hiện bằng cách trượt kernel qua từng điểm ảnh của ảnh đầu vào và tính tổng các tích chập tại mỗi điểm ảnh. Quá trình này được thực hiện cho từng kênh màu riêng lẻ.

3. **Padding**: Trong quá trình tích chập, việc tính toán tại các điểm biên của ảnh gốc có thể dẫn đến việc mất thông tin, gây hiện tượng biên đốm (boundary artifacts). Để tránh vấn đề này, thường sử dụng padding để bao quanh ảnh ban đầu với các giá trị pixel ngoài biên ảnh. Trong trường hợp này, sử dụng reflection padding (lề phản chiếu) để giải quyết hiện tượng biên đốm.

d Mô tả hàm

1. Chuyển đổi ảnh đầu vào thành kiểu dữ liệu **'float16'** để thực hiện tính toán số thực trong quá trình làm mờ.
2. Tạo lưới lấy mẫu **'x, y'** để tạo ra lưới tọa độ để tính toán kernel Gaussian.
3. Gọi hàm **'guassian(x, y, sigma)'** để tính toán kernel Gaussian dựa trên lưới **'x, y và sigma'**.
4. Gọi hàm **'convolve2D(image, kernel)'** để thực hiện việc tích chập 2D giữa ảnh đầu vào và kernel Gaussian.
   - Hàm sử dụng hàm **'lib.stride_tricks.as_strided'** của NumPy để tạo các mảng con trùng lắp của kênh màu với cùng hình dạng như kernel.
   - Sau đó, hàm thực hiện phép nhân từng phần tử giữa kernel và mỗi mảng con này bằng cách sử dụng hàm **'einsum'** của NumPy, từ đó tính được tích chập 2D.

e Kết quả

- Làm mờ hình ảnh bằng thuật toán Guassian.

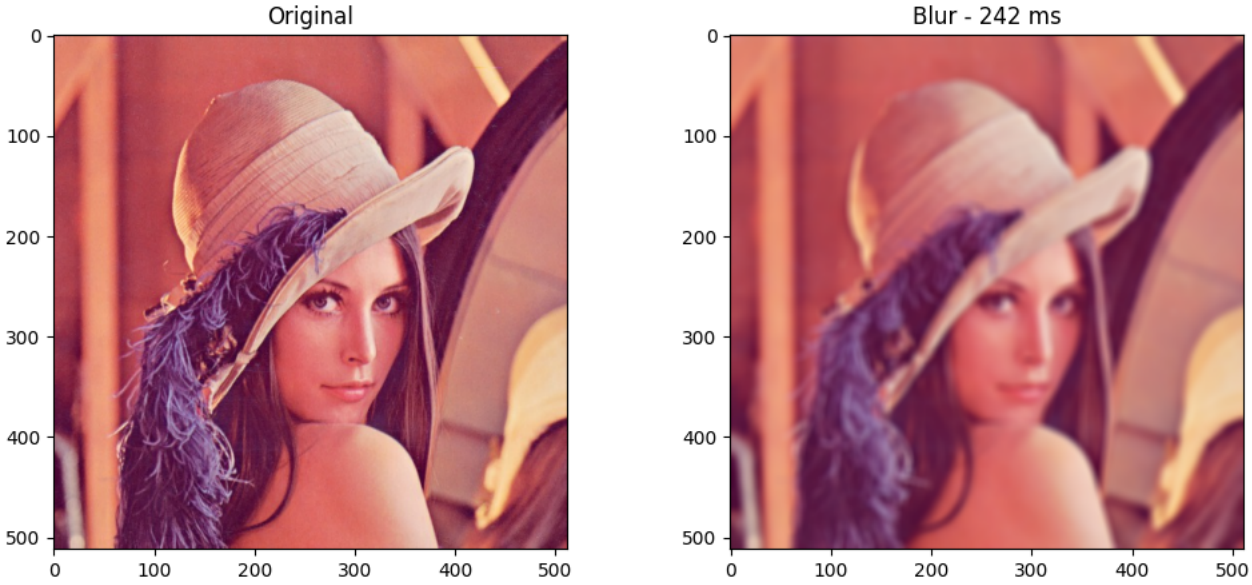

Figure 6: Hình ảnh sau khi làm mờ với kích thước kernel = 10

f Nhận xét

- Ảnh kết quả sẽ có hiệu ứng mờ mịt và làm giảm chi tiết. Điều này là do sử dụng Gaussian blur để làm mờ ảnh, tạo ra một bản sao của hình ảnh nhưng với các điểm ảnh mịn hơn.

- Độ mờ của ảnh kết quả phụ thuộc vào **'kernel_size'** và **'sigma'**.
  + **'kernel_size'** là kích thước của ma trận kernel Gaussian, ảnh mờ càng mịn và chi tiết bị mất đi càng nhiều khi **'kernel_size'** càng lớn.
  + **'sigma'** là tham số điều chỉnh mức độ phân tán của hàm Gauss, ảnh mờ càng mịn khi **'sigma'** càng lớn. Tuy nhiên, cần lưu ý rằng cũng có thể xảy ra hiện tượng hiện tượng "mờ đục" (oversmoothing) nếu **'sigma'** quá lớn.

3.8 Làm sắc nét hình ảnh

```
def sharpen_image(image):
```

a Tham số truyền vào

- **'image'** (ndarray): Hình ảnh truyền vào ở dạng mảng nhiều chiều.
- **'kernel_size'** (int): Kích thước của kernel Guassian. Mặc định là 10 nếu người dùng không cung cấp.
- **'sigma'** (int): Tham số sigma đại diện cho độ lệch chuẩn của phân phối Guassian. Mặc định là 5 nếu người dùng không cung cấp.

b Tham số trả về

- Một mảng NumPy biểu diễn ảnh sau khi được làm sắc nét bằng kernel Gaussian.

c Ý tưởng thực hiện

1. Làm mờ hình ảnh truyền vào bằng thuật toán Guassian.
2. Lấy ảnh gốc trừ cho ảnh đã làm mờ để giữ lại những đặc điểm nổi bật của hình ảnh.
3. Lấy ảnh gốc cộng cho kết quả trên để tạo ra một hình ảnh được làm nổi bật các đường nét.

d Mô tả hàm

1. Gọi hàm **'blur_image'** để làm mờ ảnh truyền vào.
2. Lấy ảnh gốc trừ cho ảnh đã được làm mờ, sau đó lại cộng cho ảnh gốc. Để tránh việc tràn số, trước khi thực hiện, ảnh đã được ép kiểu sang **'float16'**.
3. Gọi hàm **'np.clip'** để giới hạn giá trị trong khoảng hợp lệ của các điểm ảnh.
4. Ép kiểu ảnh về lại định dạng của ảnh màu **'uint8'**.

e Kết quả

- Làm sắc nét hình ảnh bằng thuật toán Guassian.

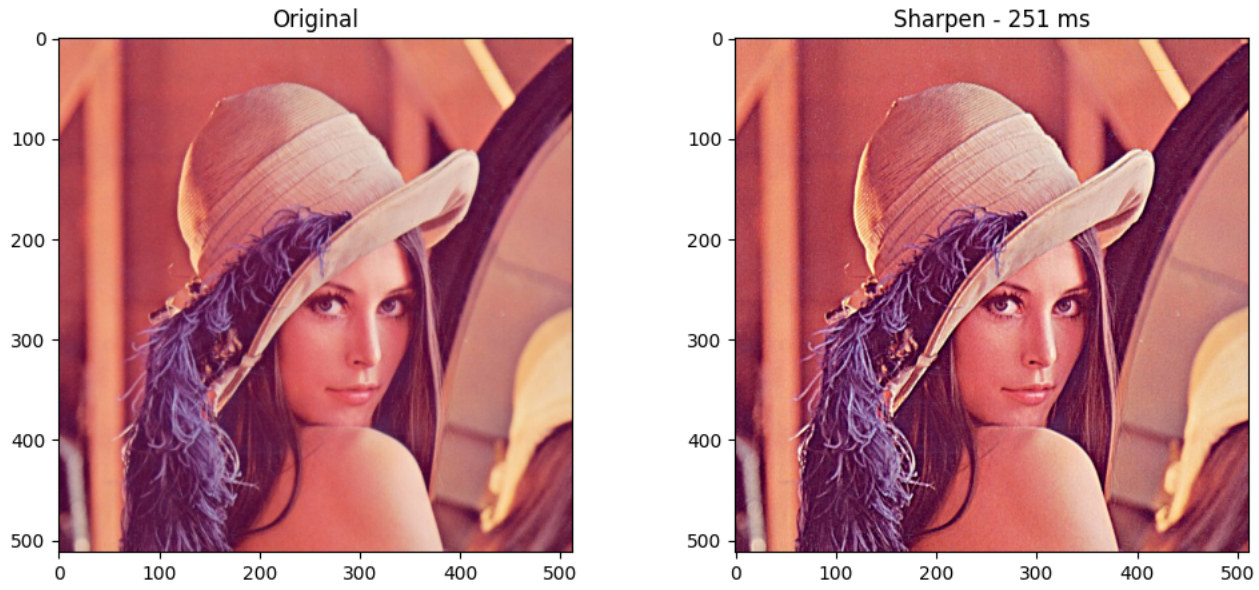

Figure 7: Hình ảnh sau khi làm sắc nét với kích thước kernel = 10

f Nhận xét

- Ảnh kết quả sẽ có hiệu ứng sắc nét hơn, với các cạnh và chi tiết được làm nổi bật hơn so với ảnh gốc. Phương pháp làm sắc nét này thường giúp tạo ra ảnh có vẻ rõ ràng, sắc nét hơn và cân bằng màu sắc tốt hơn.
- Hiệu quả của sắc nét hình ảnh phụ thuộc vào **'kernel_size'** và **'sigma'**. Như trong hàm **'blur_image'**, **'kernel_size'** và sigma đóng vai trò quan trọng trong quá trình làm sắc nét. Kích thước và độ phân tán của kernel Gaussian sẽ ảnh hưởng đến mức độ sắc nét và chi tiết trong ảnh kết quả.
- Phương pháp làm sắc nét có thể làm tăng nhiễu trong hình ảnh, đặc biệt khi hai giá trị trên được tăng quá cao. Điều này có thể làm tăng nhiễu và giảm chất lượng của ảnh kết quả. Để tránh tình trạng này, cần thử nghiệm và điều chỉnh kỹ càng các tham số trên để có được kết quả sắc nét tốt nhất mà không làm tăng nhiễu quá mức.

3.9 Cắt ảnh ở trung tâm theo kích thước

```
def cut_center(image, width, height, size)
```

a Tham số truyền vào

- **'image'** (ndarray): Hình ảnh truyền vào ở dạng mảng nhiều chiều.

- **'width, height'** (int): Tham số đại diện cho chiều rộng và cao của ảnh.
- **'size'** (int): Tham số đại diện cho kích thước ảnh sau khi cắt.

b Tham số trả về

- Hàm trả về một hình vuông (ndarray) đã sau khi thực hiện cắt.

c Ý tưởng thực hiện

- Cắt các phần ảnh dư thừa, chỉ giữ lại phần ảnh mong muốn. Chiều dài, rộng của ảnh mới được xác định từ vị trí dòng, cột trung tâm của ảnh cũ, lùi về các cạnh đúng bằng một nửa kích thước mong muốn.

d Mô tả hàm

1. Nếu kích thước truyền vào lớn hơn số dòng hoặc cột, thì xem như kích thước mong muốn bằng với số dòng hoặc cột đó.
2. **'lower_bound', 'upper_bound'**: Là chỉ số dùng để xác định vùng cần cắt từ hình ảnh gốc. **'lower_bound'** là chỉ số thấp nhất của hàng và cột cần cắt, và **'upper_bound'** là chỉ số cao nhất của hàng và cột cần cắt. Điều này sẽ giúp chúng ta cắt một vùng vuông từ hình ảnh gốc.
3. `slicing image[lower_bound : upper_bound, lower_bound : upper_bound]`: Kỹ thuật slicing này sẽ giữ lại toàn bộ các dòng, cột tính từ vị trí trung tâm ra các cạnh sao cho số dòng, cột bằng với 1/2 kích thước truyền vào.
4. **Kết quả** `new_image`: Kết quả là hình vuông đã được cắt tính từ trung tâm của hình ảnh gốc.

e Kết quả

- Cắt ảnh tại trung tâm.

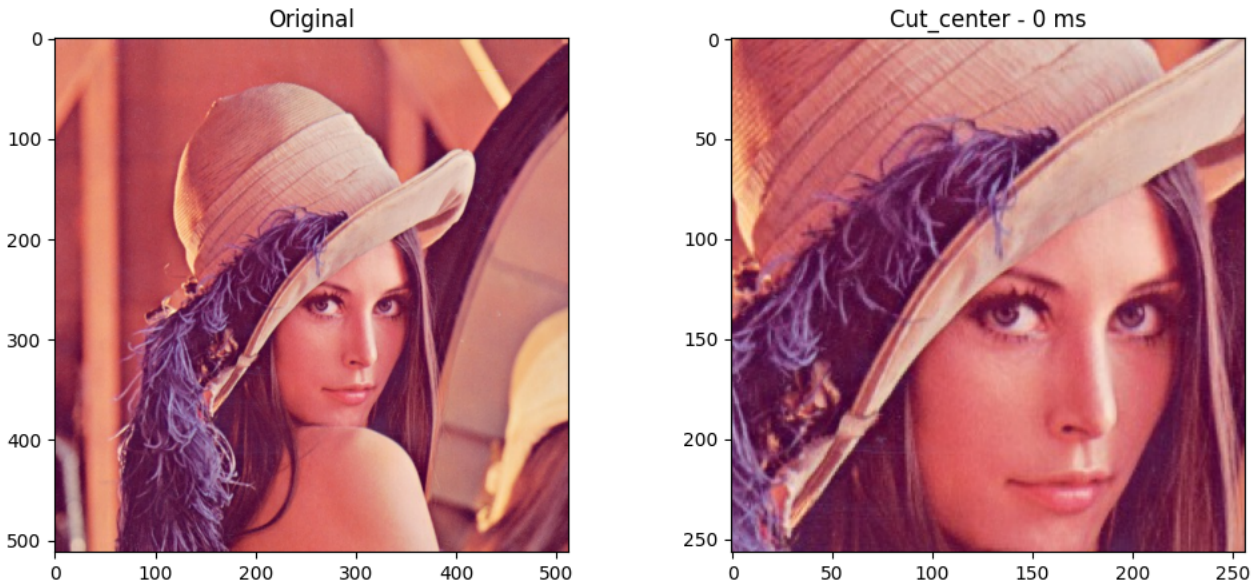

Figure 8: Hình ảnh sau cắt tại trung tâm với kích thước bằng 256x256

f Nhận xét

- Hàm cắt ảnh tại trung tâm cho kết quả tốt, tuy nhiên, nếu ảnh gốc có kích thước nhỏ hơn kích thước mong muốn thì kết quả sẽ không thay đổi.
- Hàm cho hiệu suất tốt về thời gian thực thi, do chỉ sử dụng thuật toán indexing của numpy để giữ lại các điểm ảnh cần thiết.
- Có thể phát triển thuật toán từ cắt trung tâm ảnh lên thành cắt ảnh theo vị trí bất kì.

3.10 Cắt ảnh tròn theo kích thước

```
def cut_circle (image, width, height)
```

a Tham số truyền vào

- **'image'** (ndarray): Hình ảnh truyền vào ở dạng mảng nhiều chiều.
- **'width, height'** (int): Tham số đại diện cho chiều rộng và cao của ảnh.

b Tham số trả về

- Hàm trả về mảng chứa các điểm ảnh có thể biểu diễn thành một hình tròn (ndarray).

c Ý tưởng thực hiện

- Xác định vị trí của tâm, bán kính của hình tròn. Giữ lại tất cả các điểm mà khoảng cách từ điểm đó tới tâm nhỏ hơn hoặc bằng bán kính của hình tròn.

d Mô tả hàm

1. Hàm `circular_mask(width, height)`: Hàm này tạo ra một ma trận boolean (giá trị True hoặc False) có kích thước **'width x height'**, đại diện cho mặt nạ (mask) của một vùng tròn.

   - **'width, height'**: Chiều rộng và cao của mặt nạ hình tròn.

   Cụ thể, hàm thực hiện các bước sau:

   - Tính toán center: Đây là bán kính của vùng tròn và được tính bằng cách lấy giá trị nhỏ nhất giữa **'width, height'** sau khi đã chia lấy phần nguyên cho 2.
   - Tạo một ma trận **'x, y'** sử dụng hàm **'np.ogrid'**. Ma trận này lưu trữ các giá trị từ 0 đến **'width - 1'** cho x và từ 0 đến**'height - 1'** cho y.
   - Tạo mặt nạ tròn bằng cách tính khoảng cách từ tâm của hình ảnh đến từng điểm **'(x, y)'** trong ma trận. Nếu khoảng cách nhỏ hơn hoặc bằng **'center'**, thì giá trị tương ứng trong mặt nạ sẽ là **'True'**, ngược lại sẽ là **'False'**.

2. Hàm `cut_circle(image, width, height)`: Hàm này nhận một hình ảnh image và thực hiện việc cắt một vùng tròn từ hình ảnh dựa trên mặt nạ được tạo ra bởi hàm **'circular_mask(width, height)'**.

   - **'image'**: Hình ảnh đầu vào có kích thước width x height.
   - **'width'**: Chiều rộng của hình ảnh.
   - **'height'**: Chiều cao của hình ảnh.

   Cụ thể, hàm thực hiện các bước sau:

   - Gọi hàm **'circular_mask(width, height)'** để tạo một mặt nạ tròn (mask).
   - Khởi tạo một ma trận kết quả **'result'** có kích thước giống như hình ảnh đầu vào **'image'**.
   - Sao chép các giá trị từ hình ảnh đầu vào vào biến **'result'** dựa trên mặt nạ **'mask'**. Các điểm nằm trong vùng tròn sẽ được sao chép giữ nguyên giá trị, các điểm nằm ngoài vùng tròn sẽ có giá trị là 0 (do ma trận **'result'** được khởi tạo bằng 0).

3. **Kết quả** `new_image`: Kết quả là hình tròn đã được cắt từ hình ảnh gốc.

e Kết quả

- Cắt ảnh theo hình tròn có đường kính là chiều rộng của ảnh vuông, và tâm hình tròn là tâm ảnh.

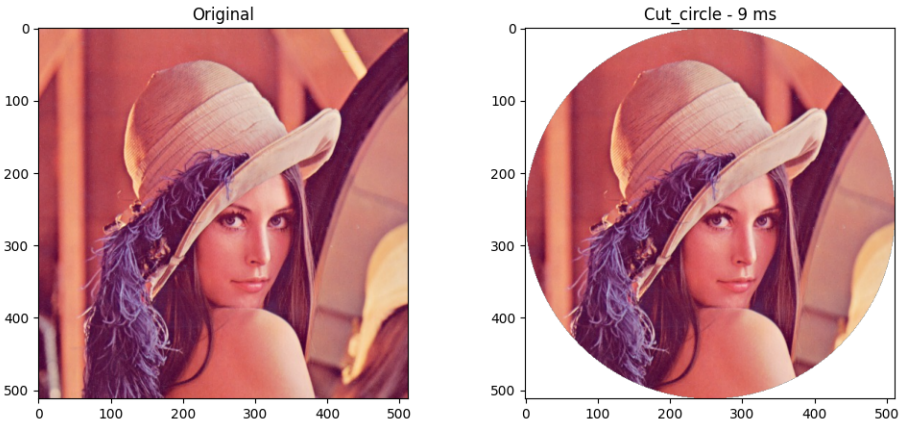

Figure 9: Hình ảnh sau khi cắt hình tròn.

f Nhận xét

- Hàm này có thời gian thực thi tương đối nhanh chóng và hiệu quả, vì nó chỉ thực hiện các phép toán đơn giản như tính toán vùng hình tròn và gán giá trị 0 cho các điểm ảnh nằm ngoài vùng cắt.
- Kết quả cắt hình tròn sẽ tạo ra một vùng cắt có hình dạng tương đối đơn giản, thích hợp cho một số ứng dụng đơn giản như tạo hiệu ứng hình tròn.
- **'Hạn chế'**: Hàm không có khả năng thay đổi bán kính của hình tròn cắt. Bán kính của hình tròn luôn là giá trị nhỏ nhất giữa **'width // 2, height // 2'** . Điều này có thể hạn chế trong việc tùy chỉnh vùng cắt cho phù hợp với nhu cầu cụ thể.

3.11 Cắt ảnh elip

```
def cut_ellipse(image, width, height)
```

a Tham số truyền vào

- **'image'** (ndarray): Hình ảnh truyền vào ở dạng mảng nhiều chiều.
- **'width, height'** (int): Tham số đại diện cho chiều rộng và cao của ảnh.

b Tham số trả về

- Hàm trả về một hình được cắt theo hình dạng 2 elip chéo nhau (ndarray).

c Ý tưởng thực hiện

- Xác định phương trình hình elip:

$$\frac{(x * \cos\alpha + y * \sin\alpha)^2}{a^2} + \frac{(x * \cos\alpha - y * \sin\alpha)^2}{b^2} = 1 \tag{2}$$

- Từ hình ảnh mẫu của đề bài, có thể đưa ra kết luận rằng, đối với ảnh vuông, hình elip sẽ đi qua 2 điểm:
  + Giao điểm giữa các cạnh của hình vuông với hình elip. Đối với hình minh hoạ bên dưới là các điểm có toạ độ: (4, -2), (2, -4), (-4, 2), (-2, 4).
  + Trung điểm của đường chéo hình vuông nằm trong góc phần tư thứ 1 hoặc 3. Đối với hình minh hoạ bên dưới là các điểm có toạ độ: (2, 2), (-2, -2).

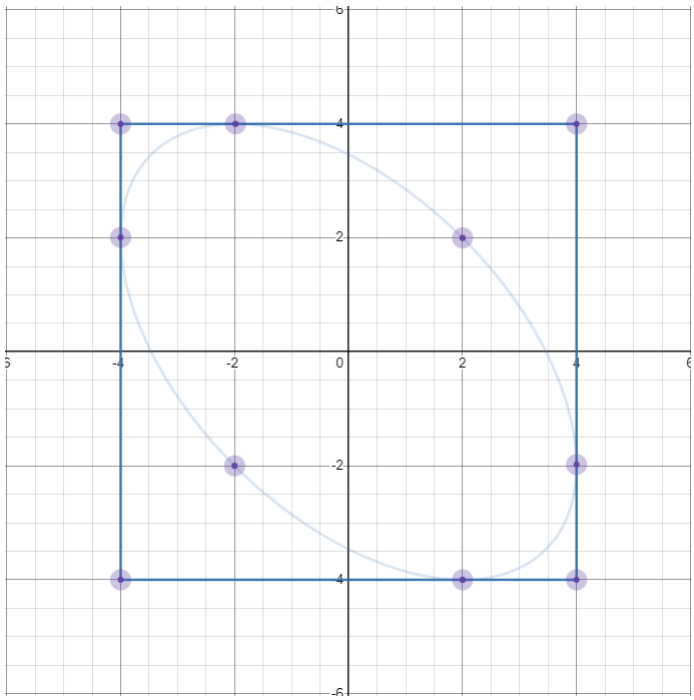

Figure 10: Giao điểm của elip với hình ảnh vuông

- Nói cách khác, phương trình elip sẽ đi qua 2 điểm sau:
  + **'edge'** (int): là một nửa chiều dài của cảnh nhỏ hơn (**'width, height'**).
  + $(x, y) = (edge \div 2, -edge \div 4)$
  + $(x, y) = (edge \div 4, edge \div 4)$
- Từ 2 điểm trên, thế vào phương trình elip để tìm giá trị của tham số **'a'**, **'b'** cho phương trình.
  + Với điều kiện đi qua điểm thứ nhất có toạ độ x = y và góc $\alpha = 45°$, t có thể suy ra giá trị của a.

$$a^2 = (x * \cos\alpha + y * \sin\alpha)^2 \tag{3}$$

  + Thế toạ độ của điểm còn lại và giá trị a^2 vừa tìm được vào phương trình elip để tìm giá trị của b^2.

d Mô tả hàm

1. Hàm `solve_equation(point1, point2, alpha=45)`: Hàm này giải một hệ phương trình bậc hai để tính toán hai tham số chính A và B của phương trình hình elip. Điểm **'point1, point2'** là hai điểm mà phương trình elip đi qua, và **'alpha'** là góc xoay của hình elip.

2. Hàm `ellipse_mask(width, height, A, B, alpha=45)`: Hàm này tạo ra một ma trận boolean (giá trị True hoặc False) có kích thước **'width x height'**, đại diện cho mặt nạ (mask) của hình elip.

   - **'width'**(int): Chiều rộng của hình ảnh.
   - **'height'**(int): Chiều cao của hình ảnh.
   - **'A, B'**(float): Hai tham số chính của phương trình elip.
   - **'alpha'**(int): Góc xoay của hình elip.

   Cụ thể, hàm thực hiện các bước sau:

   - Tạo ma trận x, y sử dụng hàm np.ogrid. Ma trận này lưu trữ các giá trị từ 0 đến **'width - 1'** cho x và từ 0 đến **'height - 1'** cho y.
   - Thực hiện việc xoay tọa độ của mỗi điểm **'(x, y)'** trong ma trận theo góc alpha để tạo một ma trận **'rotate_x'** và **'rotate_y'**.
   - Tính toán giá trị của điều kiện **'rotate_x ** 2 / B + rotate_y ** 2 / A <= 1.0'**, và gán giá trị True cho các điểm thỏa mãn điều kiện này trong mặt nạ, các điểm còn lại có giá trị False.

3. Hàm `cut_ellipse(image, width, height)`: Hàm thực hiện cắt một vùng có dạng 2 ellipse chéo nhau từ hình ảnh đầu vào.

   - **'image'**(ndarray): Ảnh đầu vào.
   - **'width'**(int): Chiều rộng của hình ảnh.
   - **'height'**(int): Chiều cao của hình ảnh.

   Cụ thể, hàm thực hiện các bước sau:

   - Xác định góc xoay **'alpha'** và các điểm **'point1, point2'** để tính toán các tham số **'A, B'** cho hình elip bằng cách gọi hàm **'solve_equation'**.
   - Khởi tạo một ma trận kết quả **'result'** có kích thước giống như hình ảnh đầu vào **'image'**.
   - Gọi hàm **'ellipse_mask'** để tạo các mặt nạ elip với góc xoay **'alpha'** và **'-alpha'**.

- Sao chép các giá trị từ hình ảnh đầu vào **'image'** vào **'result'** dựa trên các mặt nạ elip. Các điểm nằm trong vùng elip sẽ được sao chép giữ nguyên giá trị, các điểm nằm ngoài vùng elip sẽ có giá trị là 0.

4. **Kết quả** `new_image`: Kết quả là hình elip đã được cắt từ hình ảnh gốc.

e Kết quả

- Cắt ảnh theo theo 2 hình ellipse chéo nhau.

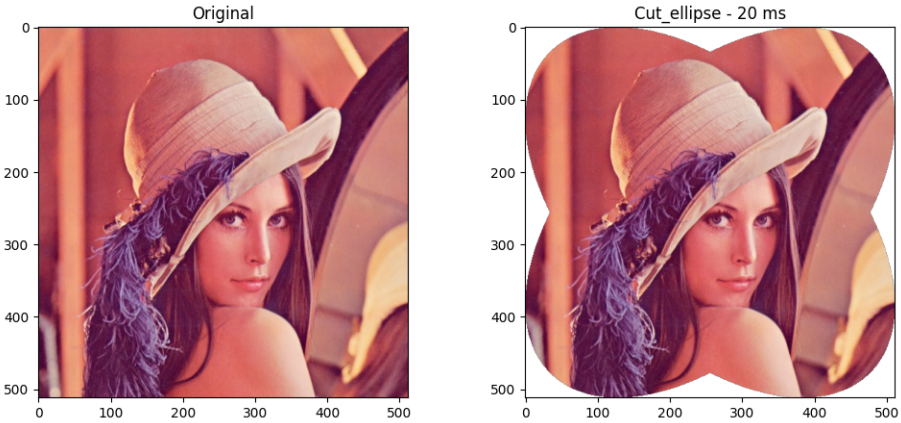

Figure 11: Hình ảnh sau khi cắt hình tròn.

f Nhận xét

- **'Điểm mạnh'**:
  + Tạo ra một vùng cắt hình elip chéo từ hình ảnh gốc, tạo hiệu ứng độc đáo và thú vị.
  + Thời gian thực thi hàm này khá nhanh chóng và hiệu suất tốt, do hàm chỉ thực hiện các phép toán đơn giản như tính toán mask và cắt ảnh.
- **'Điểm yếu'**:
  + Hàm chỉ cắt hình elip chéo có hướng 45 độ và có kích thước nhất định dựa vào hai điểm point1 và point2. Nếu cần cắt các hình elip khác hoặc có hướng khác, hàm này không thực hiện được yêu cầu này.

4 Các hàm chức năng trong chương trình:

4.1 Lấy thông tin của ảnh

```
def get_image_info(image)
```

a Tham số truyền vào

- **'image'** (ndarray): Hình ảnh truyền vào ở dạng mảng nhiều chiều.

b Tham số trả về

- Hàm trả về chiều cao, rộng và số chiều của ảnh.

c Mô tả hàm

1. Giá trị chiều cao, rộng tương ứng với thuộc tính **'.shape[i]'** với i lần lượt là 0, 1.
2. Giá trị số chiều tương ứng với thuộc tính **'.shape[2]'**. Số chiều sẽ bằng giá trị của thuộc tính nếu > 2, nếu giá trị thuộc tính trả về là 2 nghĩa là ảnh có 1 giá trị màu (ảnh xám).

4.2 Danh sách các kênh màu của hình ảnh

```
def slicing_color_channel(image)
```

a Tham số truyền vào

- **'image'** (ndarray): Hình ảnh truyền vào ở dạng mảng nhiều chiều.

b Tham số trả về

- Hàm trả về danh sách (list) giá trị của từng kênh màu của ảnh.

c Mô tả hàm

1. Gán giá trị **'color_channels'**: Đây là một danh sách các kênh màu được cắt từ hình ảnh gốc. Mỗi kênh màu là một mảng hai chiều lưu trữ giá trị màu tương ứng cho từng pixel của kênh đó.
2. Mỗi lần duyệt, chúng ta cắt ra một kênh màu bằng cách sử dụng cú pháp **'image[:, :, i]'**, nghĩa là lấy toàn bộ hàng và cột của kênh thứ i từ hình ảnh.

4.3 Đọc ảnh và ép kiểu sang Numpy array

```
def read_image(img_name)
```

a Tham số truyền vào

- **'img_name'** (string): Tên của hình ảnh cần đọc.

b Tham số trả về

- Hàm trả về mảng (numpy.array) chứa đựng giá trị của các pixel của ảnh.

c Mô tả hàm

1. Đầu tiên, hàm sử dụng thư viện Pillow (PIL) để đọc hình ảnh từ tệp có tên **'img_name'**.
2. Chuyển đổi đối tượng hình ảnh PIL thành một mảng numpy (numpy.array). Điều này giúp biểu diễn hình ảnh dưới dạng mảng số trong numpy, cho phép xử lý và tính toán hình ảnh một cách dễ dàng.

4.4 Lưu ảnh

```
def save_image(image, name, captions, mode)
```

a Tham số truyền vào

- **'image'** (list): Hình ảnh cần lưu (ndarray).
- **'name'** (string): Tên của hình ảnh cần đọc.
- **'captions'** (list): Danh sách các chú thích cho từng hình ảnh (string).
- **'mode'** (int): Chức năng mà người dùng chọn. 0 là thực hiện toàn bộ chức năng xử lý ảnh hoặc chỉ số của từng chức năng.

b Mô tả hàm

1. Tách tên cơ sở và định dạng tệp từ tham số **'name'** bằng cách tìm vị trí dấu chấm (.).
2. Xác định tên ảnh mới **'img_name'** dựa trên chế độ lưu trữ **'mode'** và danh sách **'captions'**.
3. Lưu từng hình ảnh với tên **'img_name'**.

4.5 Hiển thị ảnh

```
def show_image(images, new_images, captions, runtime, mode)
```

a Tham số truyền vào

- **'image'** (ndarray): Hình ảnh gốc.
- **'new_images'** (list): Danh sách các hình ảnh mới cần hiển thị dưới dạng mảng numpy.
- **'captions'** (list): Danh sách các chú thích cho từng hình ảnh (string).

- **'mode'** (int): Chức năng mà người dùng chọn. 0 là thực hiện toàn bộ chức năng xử lý ảnh hoặc chỉ số của từng chức năng.

b Mô tả hàm

1. Xác định số hàng và số cột của các subplot trong hình ảnh dựa trên **'mode'** và số lượng hình ảnh mới.
2. Tạo hình ảnh subplot sử dụng matplotlib với số hàng và số cột đã xác định, và thiết lập kích thước của hình ảnh subplot.
3. Hiển thị ảnh gốc và các ảnh đã qua xử lý kèm theo thông tin chức năng và thời gian thực hiện chức năng đó.

4.6 Khởi tạo chú thích cho ảnh

```
def set_captions()
```

a Mô tả hàm

Hàm **'set_captions()'** định nghĩa một từ điển captions chứa các chú thích cho các phép biến đổi hình ảnh. Key (khóa) của từ điển là số nguyên, tương ứng với mỗi phép biến đổi, và value (giá trị) là mô tả cho từng phép biến đổi.

4.7 Gọi hàm thực hiện chức năng

```
def exec_mode(image, width, height, mode, num_channels)
```

a Tham số truyền vào

- **'image'** (ndarray): Hình ảnh gốc.
- **'width'**(int): Chiều rộng của hình ảnh.
- **'height'**(int): Chiều cao của hình ảnh.
- **'mode'** (int): Chức năng mà người dùng chọn. 0 là thực hiện toàn bộ chức năng xử lý ảnh hoặc chỉ số của từng chức năng.

b Tham số trả về

- **'result'**: Danh sách các hình ảnh sau khi được biến đổi dưới dạng mảng numpy.
- **'runtime'**: Danh sách thời gian thực hiện (miliseconds) của từng phép biến đổi tương ứng trong **'result'**.

c Mô tả hàm

1. Hàm sử dụng danh sách **'operations'** chứa các cặp giá trị (function, arguments) để biểu diễn các phép biến đổi có sẵn, mỗi phép biến đổi được đại diện bởi một hàm và các đối số tương ứng của nó.
2. Hàm trả về **'result'** và **'runtime'** sau khi hoàn thành quá trình thực hiện phép biến đổi.

4.8 Hàm main

```
def main()
```

a Mô tả hàm

1. Nhập tên tệp hình ảnh và chức năng xử lý ảnh từ người dùng.
2. Đọc hình ảnh và khởi tạo thông tin chú thích cho các chức năng.
3. Lấy thông tin chiều cao, chiều rộng và số lượng kênh màu của hình ảnh.
4. Thực hiện chức năng xử lý hình ảnh.
5. Hiển thị và lưu những hình ảnh đã xử lý ra file.

5 Tài liệu tham khảo:

- Hướng dẫn đồ án 02 - GV: Phan Thị Phương Uyên
- Kỹ thuật slicing và indexing
- Kỹ thuật broadcasting
- Cân bằng màu sắc cho ảnh RGB
- Chuyển đổi ảnh màu RGB thành ảnh đơn sắc
- Chuyển đổi ảnh màu RGB thành ảnh Sepia
- Tích chập 2D sử dụng thư viện Numpy
- Phương trình elip xiên
- Thư viện Numpy
- Thư viện Matplotlib