

UNIVERSITY OF SCIENCE
VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY



**Báo cáo đồ án
Image Processing**

TOÁN ỨNG DỤNG VÀ THỐNG KÊ CHO CNTT

Giảng viên lý thuyết Vũ Quốc Hoàng
Lê Thanh Tùng

Giảng viên thực hành Phan Thị Phương Uyên
Nguyễn Văn Quang Huy

Sinh viên thực hiện Nguyễn Quốc Huy 21127511

Mục lục

1 LỜI NÓI ĐẦU	3
2 CÁC CHỨC NĂNG	3
2.1 THAY ĐỔI ĐỘ SÁNG CHO ẢNH	3
2.1.1 Ý tưởng	3
2.1.2 Thực hiện	3
2.2 THAY ĐỔI ĐỘ TƯƠNG PHẢN	4
2.2.1 Ý tưởng	4
2.2.2 Thực hiện	4
2.3 LẬT ẢNH NGANG - DỌC	4
2.3.1 Ý tưởng	4
2.3.2 Thực hiện	5
2.4 CHUYỂN ĐỔI ẢNH RGB THÀNH ẢNH XÁM SEPIA	6
2.4.1 Ý tưởng	6
2.4.2 Thực hiện	9
2.5 LÀM MỎI SẮC NÉT ẢNH	10
2.5.1 Ý tưởng	10
2.5.2 Thực hiện	13
2.6 CẮT ẢNH THEO KÍCH THƯỚC	15
2.6.1 Ý tưởng	15
2.6.2 Thực hiện	15
2.7 CẮT ẢNH THEO KHUNG HÌNH TRÒN	16
2.7.1 Ý tưởng	16
2.7.2 Thực hiện	16
2.8 CẮT ẢNH 2 HÌNH ELIP CHÉO NHAU (THÊM)	17
2.8.1 Ý tưởng	17
2.8.2 Thực hiện	19
3 CÁC HÀM BỔ TRỢ KHÁC	20
3.1 get_output_filename	20
3.2 show_image	20
3.3 show_gallery	21
3.4 save_image	21
3.5 image_handler	21

4 KẾT QUẢ	21
4.1 MỨC ĐỘ HOÀN THÀNH	21
4.2 THAY ĐỔI ĐỘ SÁNG CHO ẢNH	24
4.3 THAY ĐỔI ĐỘ TƯƠNG PHẢN	25
4.4 LẬT ẢNH NGANG - DỌC	26
4.5 CHUYỂN ĐỔI ẢNH RGB THÀNH ẢNH XÁM SEPIA	27
4.6 LÀM MỜ SẮC NÉT ẢNH	28
4.7 CẮT ẢNH THEO KÍCH THƯỚC	29
4.8 CẮT ẢNH THEO KHUNG HÌNH TRÒN	29
4.9 CẮT ẢNH 2 HÌNH ELIP CHÉO NHAU	30
5 THAM KHẢO	31

1 LỜI NÓI ĐẦU

- Đây là **đồ án 2 - Image Processing** môn **Toán Ứng dụng và Thống kê cho Công nghệ Thông tin**, khoa Công nghệ Thông tin, trường Đại học Khoa học Tự nhiên - Đại học Quốc Gia Hồ Chí Minh. Được thực hiện bởi sinh viên **Nguyễn Quốc Huy**, **MSSV 21127511**.
- Toàn bộ giải thích, ứng dụng, ví dụ, kết quả và tham khảo về đồ án đều được nêu chi tiết và đầy đủ qua báo cáo.
- Giáo viên giảng dạy:**
 - Vũ Quốc Hoàng
 - Lê Thanh Tùng
 - Phan Thị Phương Uyên
 - Nguyễn Văn Quang Huy

2 CÁC CHỨC NĂNG

2.1 THAY ĐỔI ĐỘ SÁNG CHO ẢNH

2.1.1 Ý tưởng

- Cộng một số α vào mỗi phần tử trên ma trận màu.
- Độ sáng của các pixel trên ảnh phụ thuộc vào độ lớn của số α .
- Nếu số α âm, ảnh sẽ tối, ngược lại số α dương thì ảnh sẽ sáng hơn.

2.1.2 Thực hiện

- Sử dụng phép toán cộng ma trận với một số để cộng số α vào ma trận màu.
- Độ sáng α được đặt mặc định là 30.
- Giới hạn lại giá trị của ma trận để sau khi thay đổi giá trị của ma trận theo số α , tránh trường hợp phần tử có giá trị lớn hơn 255 hoặc bé hơn 0. Sử dụng **np.clip()** trong thư viện **Numpy** để giới hạn.

```
1     new_img_clip = np.clip(adjust_img, 0, 255).astype(np.uint8)
```

2.2 THAY ĐỔI ĐỘ TƯƠNG PHẢN

2.2.1 Ý tưởng

- Ở chức năng thay đổi độ sáng cho ảnh, ta cộng vào ma trận mỗi pixel một số α nhất định. Bằng cách này, ta vẫn giữ nguyên khoảng cách (sự chênh lệch giá trị) giữa các pixel. Do đó, ta không nhìn thấy sự khác biệt quá rõ giữa các phần tử sau khi thay đổi. Để nhìn thấy rõ rệt sự chênh lệch lớn hơn giữa các phần tử hay còn gọi là **độ tương phản**, ta sẽ thực hiện phép nhân ma trận với một số vô hướng α nhất định.
- Độ tương phản của các pixel trên ảnh phụ thuộc vào độ lớn của hệ số α .
- Cũng giống như thay đổi độ sáng, số α dương thì độ tương phản sẽ tăng lên, và ngược lại.

2.2.2 Thực hiện

- Sau khi tìm hiểu, tìm được có nhiều công thức để có được số α nhằm thay đổi độ tương phản, và đơn giản nhất là nhân ma trận với số α đó.
- Sử dụng phép toán nhân ma trận với một số để nhân số α vào ma trận màu.
- Độ tương phản được đặt mặc định là 1.7
- Giới hạn lại giá trị của ma trận để sau khi thay đổi giá trị của ma trận theo số α , tránh trường hợp phần tử có giá trị lớn hơn 255 hoặc bé hơn 0. Sử dụng **np.clip()** trong thư viện **Numpy** để giới hạn.

```
1     new_img_clip = np.clip(adjust_img, 0, 255).astype(np.uint8)
```

2.3 LẬT ẢNH NGANG - DỌC

2.3.1 Ý tưởng

Lật ảnh dọc (Vertical flip):

- Một ảnh là một ma trận màu có nhiều hàng, nhiều cột để lật ảnh theo chiều dọc ta cần đảo ngược vị trí của các hàng trong ma trận, ví dụ:

Original

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

>>>>>

Vertical flip

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

- Thứ tự các phần tử trong một hàng không thay đổi, chỉ thay đổi vị trí các hàng.

Lật ảnh ngang (Horizontal flip):

- Một ảnh là một ma trận màu có nhiều hàng, nhiều cột để lật ảnh theo chiều dọc ta cần đảo ngược vị trí của các cột trong ma trận, ví dụ:

Original

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

>>>>>

Horizontal flip

4	3	2	1
8	7	6	5
12	11	10	9
16	15	14	13

- Thứ tự các phần tử trong một cột không thay đổi, chỉ thay đổi vị trí các cột. Nghĩa là không đổi vị trí các hàng.

2.3.2 Thực hiện

Lật ảnh dọc (Vertical flip):

- Ta sẽ sử dụng kỹ thuật slicing để đảo ngược vị trí các hàng với cú pháp `[::-1]`
- Kỹ thuật này tuân theo nguyên tắc:

```
list_name[start:stop:step]
```

Nghĩa là ta sẽ lấy hết tất cả các phần tử trong một mảng từ phần tử cuối cùng với bước nhảy là -1. Hay nói cách khác, ta sẽ đảo ngược vị trí các phần tử của một mảng đã cho.

- Với kỹ thuật này có một lợi thế, một mảng mới sẽ được tạo ra theo đúng ý muốn của người dùng mà không làm thay đổi giá trị của mảng ban đầu.

Lật ảnh ngang (Horizontal flip):

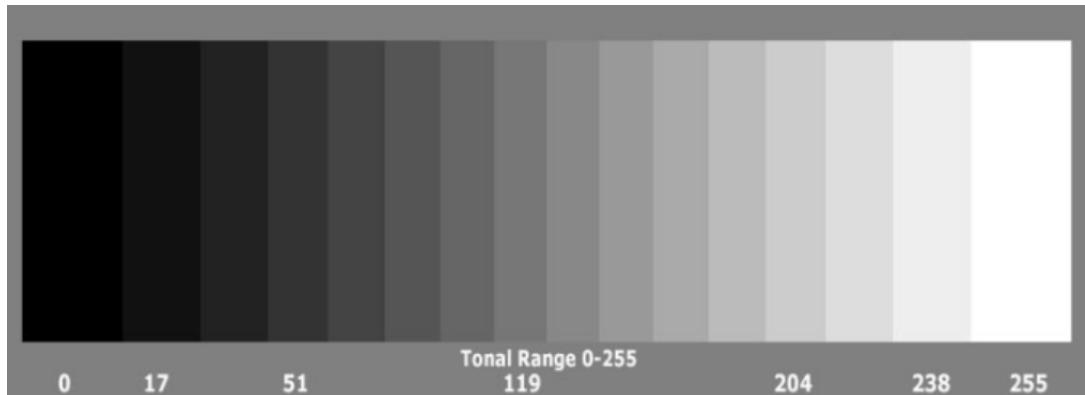
- Ta cũng sẽ sử dụng kỹ thuật slicing để thực hiện đảo ngược vị trí của các cột nhưng có khái niệm.
- Vì thuật toán này ảnh hưởng đến vị trí các cột, nghĩa là làm thay đổi vị trí các phần tử trong một hàng nên ta sẽ truy cập vào sâu thêm 1 chiều với cú pháp [, :: -1].
- Cú pháp này có nghĩa là ta sẽ truy cập vào từng hàng và lấy hết tất cả các phần tử trong hàng đó từ vị trí cuối với bước nhảy là -1. Điều này có nghĩa ta sẽ đảo vị trí từng phần tử trong từng hàng.

2.4 CHUYỂN ĐỔI ẢNH RGB THÀNH ẢNH XÁM | SEPIA

2.4.1 Ý tưởng

Ảnh xám (Grayscale):

- Tone xám là màu đơn giản nhất vì nó xác định màu sắc chỉ sử dụng một thành phần là độ sáng. Mức độ sáng được mô tả bằng giá trị nằm trong khoảng từ 0 (đen) đến 255 (trắng).
- Tính chất của mã màu tông màu xám là cả ba kênh màu R, G, B đều có giá trị như nhau. Nên biện pháp là đồng nhất giá trị của cả 3 kênh màu.



- Có các cách để đồng nhất giá trị các kênh màu như sau:

- Gán giá trị 2 kênh màu từ kênh màu nào đó:

$$\mathbf{R} = \mathbf{G} = \mathbf{B}$$

Cách này về lý thuyết sẽ không sai, màu sẽ vẫn cho ra tone xám nhưng sẽ gặp phải vài trường hợp như là màu xám quá đậm hoặc quá nhạt dẫn đến mất và mờ nét của bức ảnh.

- Lấy giá trị trung bình của cả 3 kênh màu (Average method):

$$\text{Grayscale} = (\mathbf{R} + \mathbf{G} + \mathbf{B}) / 3$$

Cách làm này sẽ cho chúng ta sự cân bằng hơn giữa các màu, nhưng cách này có thể sẽ không cho ra được kết quả như mong đợi, đa phần kết quả sẽ cho ra tone màu xám khá đậm, tưởng như đen do kênh màu R và B đang quá cao.

- Lấy giá trị từ tổng các trọng số từ 3 kênh màu (Luminosity method):

$$\text{Grayscale} = (0.3 * \mathbf{R}) + (0.59 * \mathbf{G}) + (0.11 * \mathbf{B})$$

Vì màu đỏ có nhiều bước sóng hơn trong cả ba màu và xanh lục là mang lại hiệu ứng dịu mắt hơn. Điều đó có nghĩa là chúng ta phải giảm sự đóng góp của màu đỏ và tăng giá trị của màu xanh lá cây, cũng như đặt sự đóng góp của màu xanh lam vào giữa hai màu này. Với cách phân bố này, bức ảnh sẽ cho ra tone màu xám vừa dịu vừa rõ nét cũng như có thể nhìn thấy dễ dàng những chi tiết trong ảnh.

⇒ **Do đó, ta sẽ sử dụng cách thứ 3 vào chức năng này.**

Ảnh sepia:

- Sepia là một tone màu đặc trưng, thường dùng nhiều trong các kỹ thuật chỉnh ảnh, tạo ra một phong cách cổ kính xen lẫn trẻ trung mà các kỹ thuật photo thường dùng.
- Để tạo ra màu sepia, ta có công thức cố định như sau:

$$tr = 0.393R + 0.769G + 0.189B$$

$$tg = 0.349R + 0.686G + 0.168B$$

$$tb = 0.272R + 0.534G + 0.131B$$

- Cụ thể, để chuyển từ rgb sang sepia, ta sẽ tính toán các giá trị mới cho cả 3 kênh màu tr, tg, tb. Ví dụ, ta có mã RGB ban đầu như sau:

$$R = 100$$

$$G = 150$$

$$B = 200$$

Đầu tiên, ta sẽ tính toán giá trị đối với tr:

$$tr = 0.393(100) + 0.769(150) + 0.189(200)$$

$$tr = 192.45$$

$$tr = 192 \text{ (taking integer value)}$$

Tương tự, với kênh màu tg và tb ta cũng sẽ áp dụng với công thức cố định bên trên để tính toán, kết quả đạt được sẽ là:

$$R = 192$$

$$G = 171$$

$$B = 133$$

2.4.2 Thực hiện

Ảnh xám (Grayscale):

- Với cách thứ 3 mà ta sẽ áp dụng để chuyển từ rgb sang grayscale, trước tiên ta cần định nghĩa một list hằng **luminosity_constant** chứa các trọng số (weight hoặc luminosity). Đây là những trọng số sẽ sử dụng để tính toán giá trị của mỗi pixel trong ảnh xám từ giá trị của từng channel (R, G, B) trong ảnh màu.

```
luminosity_constant = [0.3, 0.59, 0.11]
```

- Để thực hiện việc chuyển đổi từ list trọng số hằng. Ta thực hiện phép nhân ma trận giữa ảnh màu và list **luminosity_constant** bằng cú pháp:

```
1     grayscale_image = np.dot(image [..., :3], luminosity_constant)
```

- Cụ thể:

- **np.dot**: Đây là hàm hỗ trợ của thư viện **NumPy** dùng để tính tích vô hướng (**dot product**) giữa hai ma trận. Trong trường hợp này, **np.dot** được sử dụng để nhân ma trận **image[...,:3]** và **luminosity_constant**
- **image[...,:3]**: đây là một cách để trích xuất ba kênh màu (Red, Green, Blue) của ảnh màu.
- ‘...’ : được sử dụng để chỉ định rằng chúng ta muốn giũa nguyên tất cả các chiều còn lại của ma trận ảnh.
- **:3** : là chỉ số slicing để giữ lại 3 phần tử đầu tiên, nói cách khác là 3 kênh đầu tiên.

Ảnh sepia:

- Như đã nói, để chuyển đổi từ RGB sang Sepia ta cần thay đổi giá trị của cả 3 kênh màu, để được như vậy , ta cần trích xuất được giá trị của chúng.

- Với các cú pháp như sau:

```

1     red = image[:, :, 0:1]
2     green = image[:, :, 1:2]
3     blue = image[:, :, 2:3]

```

- Nhắc lại nguyên tắc đã đề cập ở phần trên:

list_name[start:stop:step]

- **red = image[:, :, 0:1]**: Với cú pháp này, ta đang trích xuất kênh màu đỏ (Red). Hay nói cách khác, ta đang lấy toàn bộ các hàng và cột trong kênh đầu tiên của ma trận ảnh màu.
- Tương tự với **green = image[:, :, 1:2]** và **blue = image[:, :, 2:3]** ta lần lượt trích xuất các hàng và cột trong kênh thứ hai và thứ 3 tương ứng với kênh màu xanh lục và xanh lam.
- Tiếp theo, các giá trị tương ứng trong từng kênh màu sẽ được sử dụng để tính toán giá trị của từng pixel trong ảnh Sepia.
- **np.append()**: Hàm np.append() của thư viện NumPy được sử dụng để nối các ma trận theo trực (axis) đã chỉ định. Ở đây, ta nối tr, tg, và tb theo trực axis=2, tức là nối theo chiều kích thước của các kênh. Ta sử dụng hàm này để ghép các kênh màu Sepia đã tính thành ảnh Sepia hoàn thiện.
- Cuối cùng, để đảm bảo sau khi tính toán các giá trị màu trong ma trận nằm trong khoảng **từ 0 đến 255**, ta sử dụng **np.clip()**, hàm này đã đề cập ở phần trên.

2.5 LÀM MỜ | SẮC NÉT ẢNH

2.5.1 Ý tưởng

- Để thực hiện kỹ thuật làm mờ cũng như sắc nét ảnh, ta cần biết về phép tích chập ảnh (**Convolution**).

- **Convolution** (tích chập) là toán tử mà ta thực hiện xoay cửa sổ 180 độ (flip over, tức flip 2 lần lần lượt theo trục x và y) rồi sau đó áp dụng phép **correlation** (tương quan).
- Còn **correlation** là toán tử mà ta áp dụng **sliding window** với phép biến đổi **tích vô hướng (dot product)** trên mỗi vùng ảnh.
- Để rõ hơn, sau đây là một ví dụ:

Giả sử đây là ma trận gốc ban đầu

ORIGINAL

2	6	8	7	2
7	3	6	1	11
8	7	4	6	2
1	2	3	4	7

Đây là Kernel để tích chập

KERNEL

1/9	1	1	1
	1	1	1
	1	1	1

Phép tính **correlation** định nghĩa rằng với mỗi phần tử (không tính padding - lè) trong một ma trận sẽ là tâm của ma trận tạm thời, ma trận tạm thời đó sau khi tính **tích vô hướng (dot product)** với **ma trận chập (kernel)** sẽ cho ra kết quả là một phần tử của ma trận mới. Cứ như vậy sau khi đi qua hết tất cả phần tử của ma trận gốc, ta sẽ có được toàn bộ ma trận mới. Ở ví dụ trên, ta có thể tính được phần tử đầu tiên của ma trận mới (Output) là:

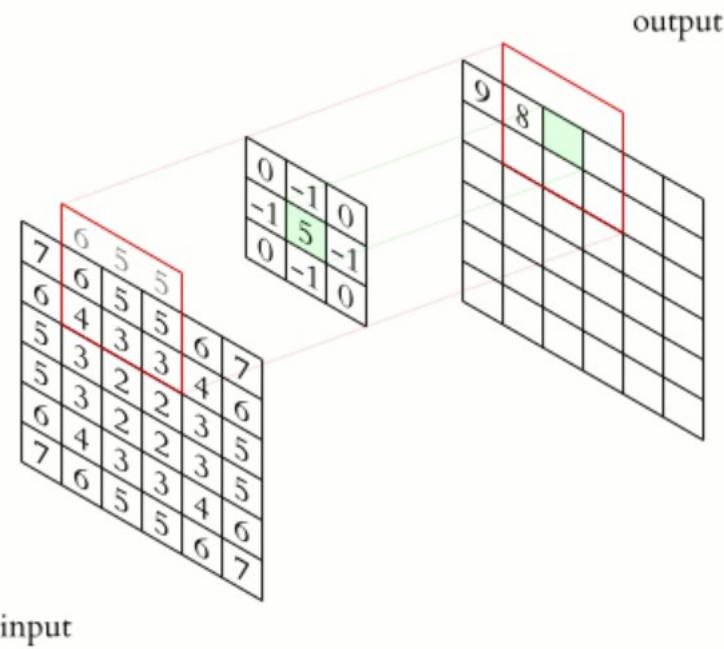
$$\text{Output}[0][0] = 2x(1/9) + 6x(1/9) + 8x(1/9) + 7x(1/9) + 3x(1/9) + 6x(1/9) + 8x(1/9) + 7x(1/9) + 4x(1/9) = 51/9 = \sim 5.67$$

Và ta sẽ có được một ma trận hoàn chỉnh nếu theo công thức đó:

OUTPUT

5.67	5	4.89
4.56	4	4.89

- Đó là các thao tác về **correlation**, nhưng vẫn còn các phần tử nằm ngoài lề (padding) vẫn chưa được xử lý, bước tiếp theo là áp dụng **convolution** để xử lý các padding đó.
- Phương pháp là thêm các padding ảo vào ma trận gốc để các phần tử ngoài lề được xem như là tâm của một ma trận, từ đó áp dụng **correlation** để hoàn thành các phần tử padding.



- Trong chức năng này ta sẽ lấy kernel làm nét như sau:

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

- Trong chức năng này ta sẽ lấy kernel để làm mờ như sau:

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

2.5.2 Thực hiện

- Trước tiên, ta cần lấy kích thước của ảnh đầu vào, gồm chiều cao và chiều rộng. Ngoài ra, ta cũng nên lấy số lượng kênh màu của ảnh (mặc dù đã có 3 kênh RGB).
- Ta sẽ sử dụng hàm **np.pad()** để tạo một bản sao của ảnh màu và thêm vào đó các viền màu vào ảnh. Viền được thêm vào với độ rộng 1 pixel xung quanh ảnh. Các giá trị viền được sao chép từ các pixel nằm ở mép của ảnh (**edge pixels**) theo nguyên tắc gần nhất, cụ thể:
- Cú pháp **np.pad()** được sử dụng trong chức năng này:

```
padded_image = np.pad(image, ((1, 1), (1, 1), (0, 0)), mode = edge)
```

- **((1, 1), (1, 1), (0, 0))**: đây là tham số **pad_width** để xác định độ dài viền được thêm vào ở mỗi chiều của ảnh.
- **(1, 1)** đầu tiên: cho chiều cao - nghĩa là độ dài viền ở phía trên và phía dưới của ảnh được đặt là 1 pixel.
- **(1, 1)** tiếp theo: cho chiều rộng - nghĩa là độ dài viền ở phía trái và phía phải của ảnh được đặt là 1 pixel.
- **(0, 0)**: là tham số cuối cùng - tham số này chỉ có ý nghĩa là không thêm viền vào tham số kênh màu.
- **mode = 'edge'**: đây là tham số xác định cách thức xử lý viền. Trong chức năng này, chúng ta sử dụng **'edge'** có nghĩa là các giá trị viền sẽ được sao chép từ các

pixel mép của ảnh (**edge pixel**).



- Sau đó, ta cần tạo ra một ma trận 0 mới có cùng kích thước với ma trận đầu bằng hàm `np.zeros_like()`. Đây cũng sẽ là ma trận kết quả sau khi làm mờ và ta bắt đầu với giá trị 0.
- Tiếp theo, ta sẽ đi qua phần quan trọng nhất đó là thực hiện phép tích chập và sử dụng các phép tính trên ma trận.
- Sử dụng cú pháp:

```
1     blurred_image[:, :, c] += padded_image[i:i + height, j:j + width, c] * kernel[i, j]
```

Với **c** là chỉ số kênh màu (R, G, B) và **i**, **j** lần lượt là các chỉ số hàng và cột trong kernel. Bằng cách này, chúng ta tính tổng trọng số của các pixel lân cận và kernel.

- Và cuối cùng, ta cũng không quên giới hạn giá trị các phần tử trong ma trận mới bằng hàm `np.clip()`.
- Đây cũng là nguyên tắc thực hiện chung của việc làm mờ cũng như làm sắc nét ảnh từ một ma trận màu.

2.6 CẮT ẢNH THEO KÍCH THƯỚC

2.6.1 Ý tưởng

- Đầu tiên ta cần xác định được cạnh của hình vuông cần cắt của ảnh thành phẩm.
- Sau đó, dựa vào chiều cao, độ rộng của ảnh có sẵn, và cạnh của hình thành phẩm, ta tính được tọa độ trái trên, trái dưới, phải trên, phải dưới của ảnh mới.
- Ví dụ, đối với cạnh trái ta sử dụng: **left = (width - size) // 2**. Đây là vị trí bắt đầu cắt ảnh từ phía trái, sao cho ảnh cắt có chiều rộng bằng cạnh (**size**) chia 2 để đảm bảo rằng việc cắt xảy ra đối xứng ở hai bên của ảnh.
- Sử dụng kỹ thuật slicing để tối ưu thuật toán và 256 là cạnh mặc định cắt từ trung tâm.

2.6.2 Thực hiện

- Như đã nói ở phần trên, ta sẽ sử dụng cú pháp bên dưới để tính toán tọa độ của ảnh crop trung tâm:

```

1      left = (width - size) // 2
2      top = (height - size) // 2
3      right = left + size
4      bottom = top + size

```

- Việc tiếp theo, ta cần cắt chúng đúng theo tọa độ đã tính.
- Khi chúng ta muốn cắt một phần của mảng ảnh **Numpy**, ta sử dụng kỹ thuật slicing để trích xuất vùng cần cắt. Trong trường hợp này, **top**, **bottom**, **left**, **right** là các tọa độ được tính toán trước đó để xác định vùng cần cắt. Qua cú pháp:

```
cropped_img_array = img_array[top:bottom, left : right , :]
```

- **top:bottom**: Đây là cú pháp slicing của NumPy để xác định các hàng của mảng mà chúng ta muốn trích xuất. **top** là chỉ số hàng bắt đầu cắt, và **bottom** là chỉ số hàng kết thúc cắt. Kết quả của slicing này sẽ trích xuất các hàng nằm trong khoảng từ **top** đến **bottom - 1**.
- **left:right**: Tương tự, đây là cú pháp slicing của NumPy để xác định các cột của mảng cần trích xuất. **left** là chỉ số cột bắt đầu cắt, và **right** là chỉ số cột kết thúc

cắt. Kết quả của slicing này sẽ trích xuất các cột nằm trong khoảng từ **left** đến **right - 1**

- Như vậy, dòng mã này sẽ trích xuất một vùng ảnh vuông từ mảng img_array. Khu vực này bắt đầu từ hàng top đến hàng bottom - 1, và từ cột left đến cột right - 1. Tất cả các kênh màu của mỗi pixel sẽ được giữ nguyên, do đó, kết quả sẽ là một ảnh màu có kích thước 256x256 pixel từ phần trung tâm của ảnh ban đầu.

2.7 CẮT ẢNH THEO KHUNG HÌNH TRÒN

2.7.1 Ý tưởng

- Sử dụng phương trình đường tròn để cắt hình theo khung tròn:

$$(x - a)^2 + (y - b)^2 = R^2$$

- Áp dụng công thức trên với **(a, b)** là tâm của đường tròn.
- R** là bán kính của đường tròn, vì đường tròn chúng ta sẽ cho tiếp xúc tới cạnh của hình nên bán kính sẽ bé hơn hoặc bằng cạnh nhỏ nhất của hình.
- a, b** chúng ta sẽ tính toán dựa trên giá trị của trung điểm mỗi cạnh.

- Những điểm không thuộc phương trình đường tròn thì đưa về màu đen (giá trị 0).

2.7.2 Thực hiện

- Đầu tiên, ta cần lấy kích thước của ảnh để tính tâm cũng như bán kính của đường tròn.
- Như đã nói, ta sẽ tiến hành tính tâm đường tròn bằng cách sử dụng trung điểm của kích thước ảnh, **x_center, y_center = width // 2, height // 2**.
- Tiếp theo, tạo ra một lưới (grid) 2 chiều (x, y) với kích thước là chiều rộng và chiều cao của ảnh. Các giá trị x và y lần lượt tạo ra một ma trận chứa các giá trị từ 0 đến (width - 1) và từ 0 đến (height - 1), cú pháp cụ thể:

1 y, x = np.ogrid [:height, :width]

- Ta sẽ sử dụng **np.ogrid** để tạo ra các ma trận 1 chiều mà mỗi phần tử chứa giá trị của trục **y** và trục **x** tương ứng.
 - Phép gán này sẽ tạo ra hai ma trận có kích thước lần lượt là (height, 1) và (1, width). Và Python sẽ tự động broadcasting các ma trận này để tạo thành hai ma trận có kích thước (height, width). Kết quả là chúng ta sẽ có một lưới 2 chiều **y** và **x** có kích thước bằng kích thước của ảnh đầu vào (height, width).
 - Như vậy, sau khi thực hiện hai dòng code trên, **y** sẽ là ma trận 2 chiều chứa các giá trị từ 0 đến height - 1, và **x** sẽ là ma trận 2 chiều chứa các giá trị từ 0 đến width - 1. Điều này cho phép chúng ta thực hiện các phép toán trên các điểm trên lưới của ảnh.
- Ta sẽ tạo một mảng numpy mới (**masked_img_array**) có kích thước giống với ảnh gốc và toàn bộ các giá trị đều là 0. Mảng này sẽ được sử dụng để lưu ảnh sau khi được cắt theo mask tròn:

```
1 masked_img_array = np.zeros_like(image)
```

- Và cuối cùng, thực hiện việc cắt hình tròn bằng cách sao chép các giá trị từ ảnh gốc vào mảng mới đã tạo (**masked_img_array**) dựa trên mask tròn (**mask**). Điều này sẽ chỉ giữ lại các giá trị ảnh nằm trong hình tròn, các giá trị ảnh nằm bên ngoài hình tròn sẽ không thay đổi (vẫn là 0).

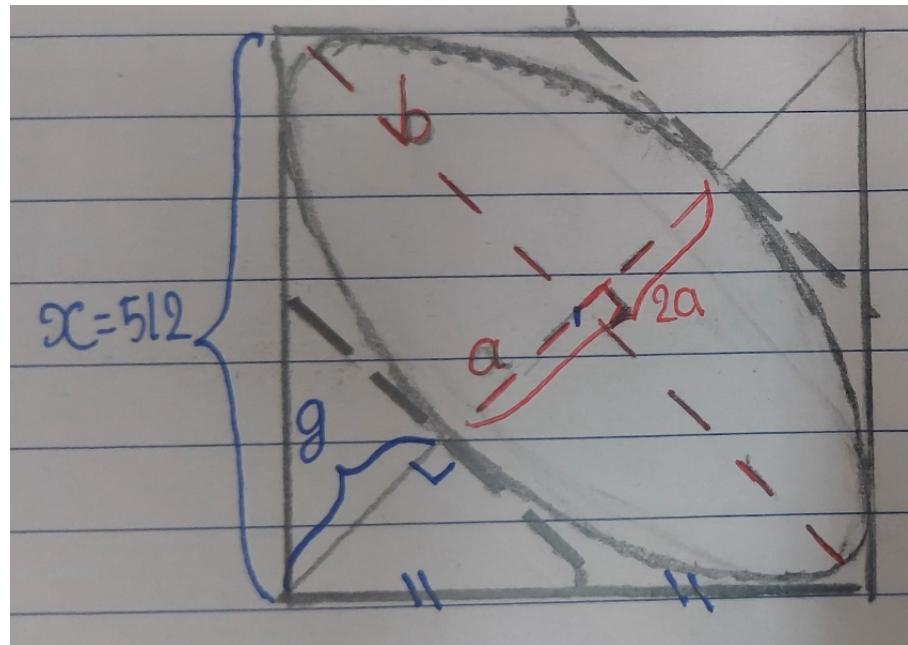
2.8 CẮT ẢNH 2 HÌNH ELIP CHÉO NHAU (THÊM)

2.8.1 Ý tưởng

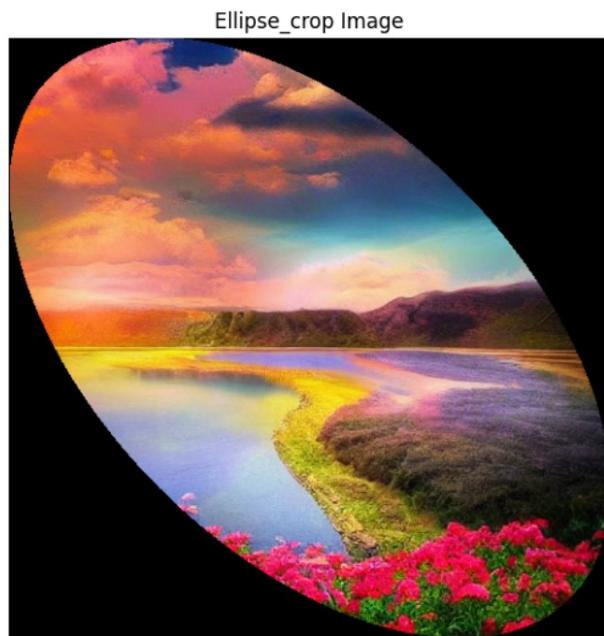
- Sử dụng phương trình đường ellipse để cắt hình theo khung ellipse:

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

- Áp dụng công thức trên với (**a**, **b**) là các đường chéo của hình ellipse, ta sẽ phải tính toán **a** và **b** sao cho hình ellipse tiếp xúc với cạnh hình vuông.
- **a**, **b** chúng ta sẽ tính toán dựa trên giá trị của mỗi cạnh.



- Với hình trên, ta giả sử đường nối từ trung điểm cạnh dưới và cạnh trái hình vuông tiếp xúc với điểm xa nhất của ellipse. Từ đó ta tìm được cạnh a với công thức là: $a = X/(2\sqrt{2})$, với X là cạnh hình vuông (hay cạnh nhỏ nhất nếu người dùng nhập vào hình chữ nhật).
- Cũng từ đó, ta tính được cạnh b với công thức là: $b = a\sqrt{3}$
- Những điểm không thuộc phương trình đường ellipse thì đưa về màu đen (giá trị 0).
- Nếu làm theo bước này, ta sẽ có được hình như sau:



- Để có 2 hình ellipse chéo nhau thì ta sẽ tiến hành xoay ellipse và làm lại các bước trên theo một góc θ .

2.8.2 Thực hiện

- Đầu tiên, tính toán tâm của hình ảnh như một mảng NumPy chứa các tọa độ x và y. Nó được tính toán là nửa chiều cao của hình ảnh và nửa chiều rộng của nó.
- Tiếp theo, tính toán trực chính và trực phục (a, b) theo công thức ở phía trên.
- Tương tự **cắt khung tròn**, ta cũng tạo hai mảng NumPy 2D x và y, biểu thị các tọa độ x và y của mỗi điểm ảnh trong hình ảnh qua cú pháp:

```
1   y, x = np.ogrid [:image.shape [0], :image.shape [1]]
```

- Sau đó, ta tính toán tọa độ $(x - h)$ và $(y - h)$:

```
1   x_centered = x - center [1]
2   y_centered = y - center [0]
```

Với $center[1]$ và $center[0]$ là tâm của hình đã tính bên trên.

- Ta bắt đầu mask 2 phương trình ellipse như công thức, ellipse 1 xoay theo 45 độ tức π chia 4 và hình thứ 2 xoay 135 độ tức $3 * \pi$ chia 4. Như vậy ta sẽ có 2 hình ellipse chéo nhau 1 góc 90 độ.
- Cuối cùng, ta sẽ sử dụng chỉ số boolean để đặt các điểm ảnh nằm ngoài cả hai hình ellip thành màu đen (giá trị điểm ảnh [0, 0, 0]) trong mảng result theo cú pháp:

```
1     result [( ellipse_1 > 1) & ( ellipse_2 > 1) ] = [0, 0, 0]
```

3 CÁC HÀM BỔ TRỢ KHÁC

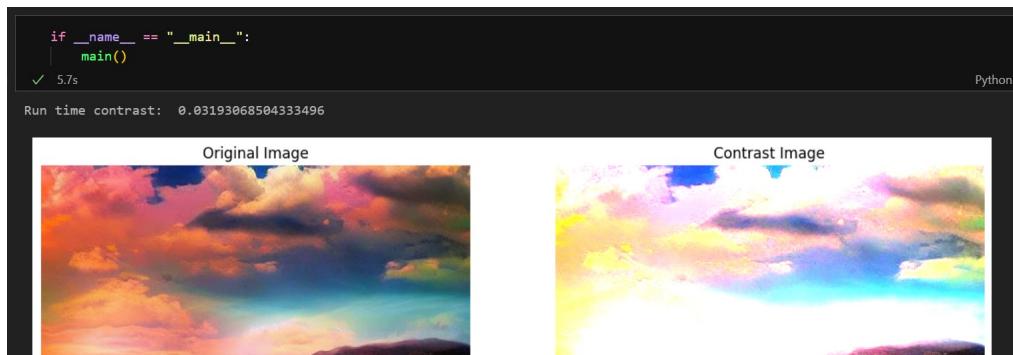
3.1 get_output_filename

Hàm này có chức năng đơn giản là tách phần tên ảnh ban đầu với phần đuôi (extension), sau đó thêm **_chức năng** tương ứng vào tên. Ví dụ:

- Input:** image.png
- Chức năng:** blur
- Output:** image_blur.png

3.2 show_image

- Hàm này có chức năng xuất 2 ảnh lên **Jupyter Notebook** gồm ảnh gốc và ảnh thành phẩm theo chức năng. Ngoài ra, có thể xuất 3 ảnh nếu đó là những chức năng tương tự nhau. Như hình:



3.3 show_gallery

- Tương tự **show_image**, hàm này cũng có chức năng show ảnh thành phẩm lên **Jupyter Notebook** nhưng sẽ áp dụng khi người dùng nhập số 0 (tất cả chức năng), xuất hàng loạt ảnh ra như một thư viện ảnh

3.4 save_image

- Hàm này có chức năng đơn giản là lưu file ảnh thành phẩm với tên đầu vào và chức năng tương ứng (kết hợp với hàm **get_output_filename**)

3.5 image_handler

- Hàm này sẽ có nhiệm vụ chính là xử lý luồng chạy của thuật toán khi người dùng nhập chức năng nào đó, đồng thời cũng sẽ xuất ảnh và lưu ảnh, tính toán thời gian dựa trên những hàm trên.

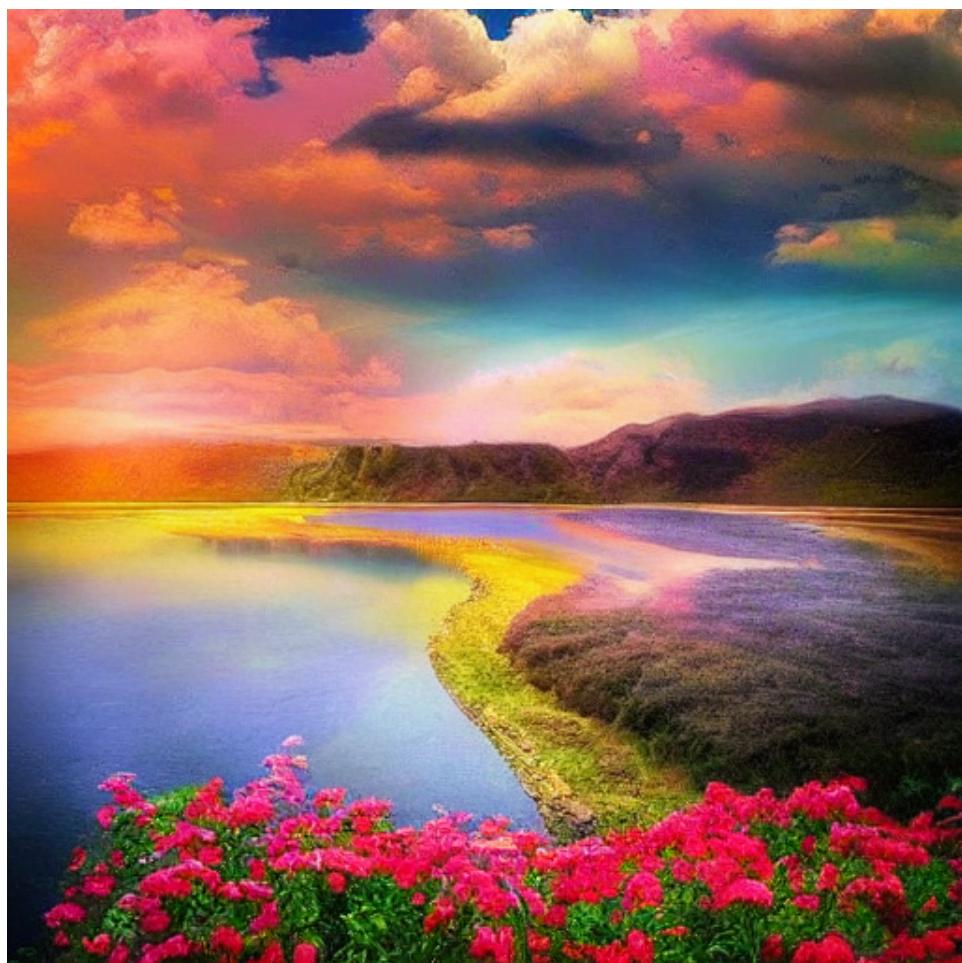
4 KẾT QUẢ

4.1 MỨC ĐỘ HOÀN THÀNH

Sau đây là nhận xét mức độ hoàn thành đồ án dựa trên những chức năng và kết quả mà đồ án đã yêu cầu, cột thời gian chỉ mang tính chất tương đối vì nó là trung bình của đo nhiều lần

STT	Chức năng	Phần trăm	Kết quả	Thời gian chạy (512x512) (giây)
1	Độ sáng	100%	Hoàn thành đủ yêu cầu	0.01
2	Độ tương phản	100%	Hoàn thành đủ yêu cầu	0.02
3	Lật ảnh ngang	100%	Hoàn thành đủ yêu cầu	0.007
4	Lật ảnh dọc	100%	Hoàn thành đủ yêu cầu	0.001
5	Ảnh xám	100%	Hoàn thành đủ yêu cầu	0.01
6	Ảnh sepia	100%	Hoàn thành đủ yêu cầu	0.06
7	Làm mờ	100%	Hoàn thành đủ yêu cầu	0.09
8	Làm sắc nét	100%	Hoàn thành đủ yêu cầu	0.08
9	Cắt ảnh	100%	Hoàn thành đủ yêu cầu	0.002
10	Cắt khung tròn	100%	Hoàn thành đủ yêu cầu	0.023
11	Cắt 2 khung ellip chéo	90%	Cắt được 2 elip chéo nhau, tiếp xúc cạnh	0.035

Sau đây là kết quả chạy thử của những chức năng đã thực hiện được dựa trên 1 bức ảnh vuông với kích thước 512x512, đây là ảnh gốc:



4.2 THAY ĐỔI ĐỘ SÁNG CHO ẢNH



(a) Ảnh gốc



(b) Ảnh đã chỉnh độ sáng 30



(a) Ảnh gốc



(b) Ảnh đã chỉnh độ sáng 50

4.3 THAY ĐỔI ĐỘ TƯƠNG PHẢN



(a) Ảnh gốc



(b) Ảnh đã chỉnh độ tương phản 1.7



(a) Ảnh gốc



(b) Ảnh đã chỉnh độ tương phản 2.5

4.4 LẬT ẢNH NGANG - DỌC



(a) Ảnh gốc



(b) Ảnh lật ngang



(a) Ảnh gốc



(b) Ảnh lật dọc

4.5 CHUYỂN ĐỔI ẢNH RGB THÀNH ẢNH XÁM | SEPIA



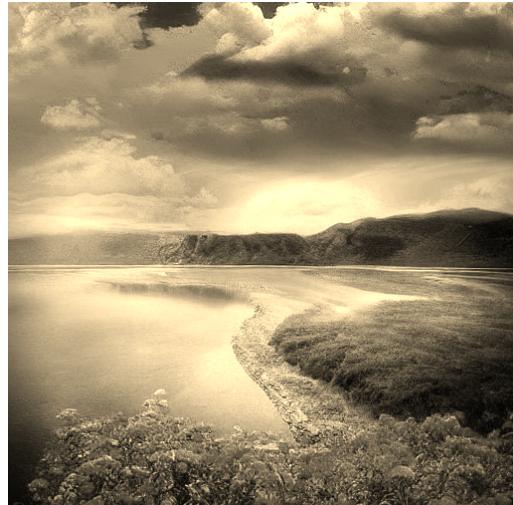
(a) Ảnh gốc



(b) Ảnh xám



(a) Ảnh gốc



(b) Ảnh sepia

4.6 LÀM MỜ | SẮC NÉT ẢNH



(a) Ảnh gốc



(b) Ảnh đã làm mờ



(a) Ảnh gốc



(b) Ảnh đã làm sắc nét

4.7 CẮT ẢNH THEO KÍCH THƯỚC



(a) Ảnh gốc



(b) Ảnh đã cắt theo kích thước 256x256

4.8 CẮT ẢNH THEO KHUNG HÌNH TRÒN



(a) Ảnh gốc



(b) Ảnh đã cắt theo khung tròn

4.9 CẮT ẢNH 2 HÌNH ELIP CHÉO NHAU



(a) Ảnh gốc



(b) Ảnh đã cắt theo 2 khung ellipse chéo nhau

5 THAM KHẢO

- Đảo ngược trong Python/ Jupyter Notebook
- Đổi RGB sang Xám (Grayscale) 1
- Đổi RGB sang Xám (Grayscale) 2
- Tìm hiểu về ảnh Sepia
- Kỹ thuật làm mờ và sắc nét ảnh
- Tìm hiểu về convolution và correlation
- Tham khảo về đường tròn
- Phương trình ellipse
- Thư viện Numpy
- Sự trợ giúp của cô Uyên và thầy Huy trong tiết thực hành

Hết!