TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG-HCM
KHOA CÔNG NGHỆ THÔNG TIN

TOÁN ỨNG DỤNG VÀ THỐNG KÊ
ĐỒ ÁN 2
IMAGE PROCESSING

Giảng viên hướng dẫn:

Vũ Quốc Hoàng
Nguyễn Văn Quang Huy
Ngô Đình Hy
Phan Thị Phương Uyên

Sinh viên thực hiện:

21127232 – Nguyễn Thanh Bình

Thành phố Hồ Chí Minh, Tháng 07/2023

MỤC LỤC

I. Các chức năng đã hoàn thành

Chức năng	Mức độ hoàn thành
1. Thay đổi độ sáng	100%
2. Thay đổi độ tương phản	100%
3. Lật ảnh ngang - dọc	100%
4. Chuyển đổi thành ảnh xám / sepia	100%
5. Làm mờ / sắc nét	100%
6. Cắt ảnh theo kích thước (cắt ở trung tâm)	100%
7. Cắt ảnh theo khung hình tròn	100%
8. Cắt ảnh theo khung là 2 hình elip chéo nhau	100%

II. Ý tưởng thực hiện

1. Khái quát chung

Xử lí ảnh là một lĩnh vực của khoa học máy tính liên quan đến việc thu thập, xử lý và phân tích hình ảnh. Nó được sử dụng trong nhiều lĩnh vực khác nhau, bao gồm y học, an ninh, giao thông vận tải và giải trí. Đồ án này được xây dựng dựa trên việc cài đặt các yêu cầu xử lí ảnh cơ bản như:

- Thay đổi độ sáng
- Thay đổi độ tương phản
- Lật ảnh ngang - dọc
- Chuyển đổi thành ảnh xám / sepia
- Làm mờ / sắc nét ảnh
- Cắt ảnh ở trung tâm theo kích thước cho trước
- Cắt ảnh theo khung hình tròn
- Cắt ảnh theo khung là 2 hình elip chéo nhau

Trước khi đi vào chi tiết mô tả cài đặt các chức năng trên, ta sẽ đi qua các ý tưởng thực hiện chúng.

2. Thay đổi độ sáng

- Mỗi điểm ảnh trong bức ảnh màu gồm 3 kênh màu là R, G, B. Khi giá trị của 3 kênh màu này càng tiến dần 255 thì độ sáng bước ảnh càng tăng và ngược lại khi 3 kênh màu này càng tiến dần về 0 thì độ sáng sẽ giảm.
- Dựa trên tính chất đó, ta có thể thực hiện việc cộng 3 kênh màu của tất cả các điểm ảnh với một hằng số nào đó để thay đổi độ sáng của bức hình theo ý muốn.
- Tuy nhiên, sau khi tính toán ta cần giới hạn lại giá trị 3 kênh màu của tất cả điểm ảnh nằm trong khoảng hợp lệ.

3. Thay đổi độ tương phản

- Độ tương phản của một bức hình được tạo nên do sự chênh lệch giữa các điểm ảnh của nó. Sự chênh lệch đó càng lớn thì độ tương phản càng cao và ngược lại.
- Dựa trên tính chất đó, ta có thể thực hiện việc nhân 3 kênh màu của tất cả điểm ảnh với một hằng số nào đó để thay đổi độ tương phản của bức hình theo ý muốn.
- Tuy nhiên, sau khi tính toán ta cần giới hạn lại giá trị 3 kênh màu của tất cả điểm ảnh nằm trong khoảng hợp lệ.

4. Lật ảnh ngang – dọc

- Để thực hiện việc lật một bức ảnh theo chiều ngang hoặc dọc ta chỉ cần đổi chỗ các vector cột hoặc dòng theo thứ tự ngược lại so với ban đầu.
- Ví dụ như ta có ma trận điểm ảnh gồm 4 vector dòng v_1, v_2, v_3 và v_4 theo thứ tự từ trên xuống, để lật ảnh theo chiều dọc ta cần đổi chỗ chúng như sau:

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} \rightarrow \begin{bmatrix} v_4 \\ v_3 \\ v_2 \\ v_1 \end{bmatrix}$$

- Tương tự cho lật ảnh theo chiều ngang, nếu ma trận ban đầu có 4 vector cột v_1, v_2, v_3 và v_4 theo thứ tự từ trái sang, ta chỉ cần đổi chỗ các vector chúng như sau:

$$[v_1\ v_2\ v_3\ v_4] \rightarrow [v_4\ v_3\ v_2\ v_1]$$

5. Chuyển đổi thành ảnh xám / sepia

- Đối với ảnh xám, sự khác biệt giữa ảnh màu và ảnh xám đó là số lượng kênh màu, mỗi điểm ảnh của ảnh xám chỉ có 1 kênh màu giá trị trong đoạn [0, 255] nhưng ảnh màu thì lại có 3 kênh màu R, G, B. Do đó để có thể chuyển đổi một bức ảnh màu thành ảnh xám ta có thể lấy trung bình 3 giá trị R, B, B của ảnh màu để thu được 1 giá trị kênh màu trong ảnh xám.
- Đối với ảnh sepia, ý tưởng thực hiện khá tương đồng với ý tưởng chuyển đổi ảnh xám nhưng thay vì tính trung bình 3 kênh màu thì ảnh sepia sẽ tính toán lại 3 kênh màu theo công thức sau:

$$newR = 0.393R + 0.769G + 0.189B$$
$$newG = 0.349R + 0.686G + 0.168B$$
$$newB = 0.272R + 0.534G + 0.131B$$

- Tuy nhiên, sau khi tính toán ta cần giới hạn lại giá trị 3 kênh màu của tất cả điểm ảnh nằm trong khoảng hợp lệ.

6. Làm mờ / sắc nét ảnh

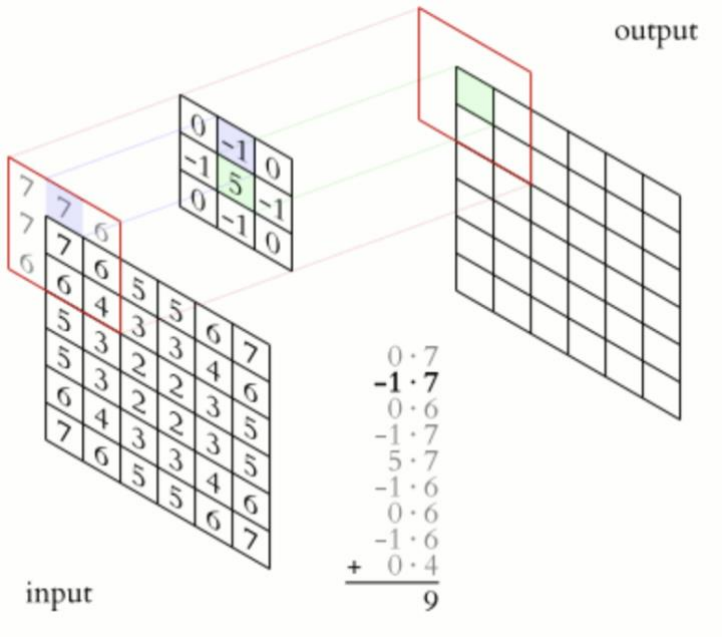
- Ý tưởng thực hiện của việc làm mờ và sắc nét ảnh là tương tự nhau chỉ khác ở chỗ sử dụng kernel trong việc tính convolution.
  - Với làm mờ, ma trận kernel được sử dụng là:

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

- Đối với làm sắc nét, ma trận kernel sẽ là:

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

- Convolution (tích chập) là quá trình tính tổng của từng điểm ảnh với các điểm ảnh lân cận nó theo trọng số của ma trận kernel. Để hiểu đơn giản, ta có thể đi qua ví dụ như sau:
  Ma trận ban đầu là ma trận 2 chiều có kích thước 6x6 (các giá trị cụ thể như trong hình) và ma trận kernel là ma trận 2 chiều có kích thước 3x3 (các giá trị cụ thể như trong hình).

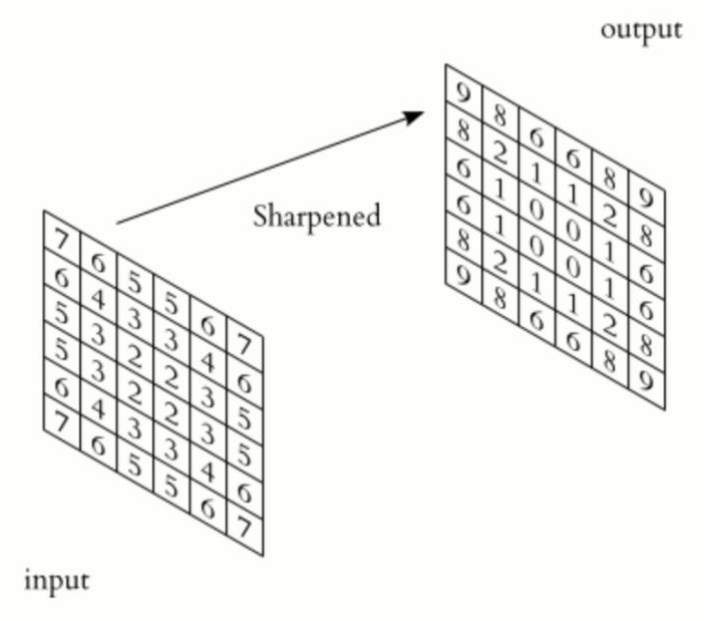


Nguồn: en.wikipedia.org

Ta lần lượt xét từng phần tử trong ma trận đầu vào, tại từng vị trí, ta thực hiện tính tích chập bằng tổng của chính phần tử đó với các phần tử xung quanh nó nhân với trọng số (những phần tử nào nằm ngoài ma trận đầu vào thì coi như bằng 0), ví dụ như tại vị trí đầu tiên bằng 7 thì tích chập sẽ bằng:

$$0.7 - 1.7 + 0.6 - 1.7 + 5.7 - 1.6 + 0.6 - 1.6 + 0.4 = 9$$

Ta ghi lại giá trị 9 vào ma trận kết quả và lặp lại quá trình trên với các phần tử còn lại cho đến hết.



Nguồn: en.wikipedia.org

7. Cắt ảnh ở trung tâm theo kích trước cho trước

- Để thực hiện việc cắt ảnh ở trung tâm theo một kích thước cho trước ta đơn giản chỉ cần giữ lại những điểm ảnh trong phạm vi kích thước đó tính từ trung tâm bức ảnh bằng cách dùng kĩ thuật Indexing của Python.
- Ví dụ như kích thước cần cắt là α, kích thước ảnh gốc ban đầu là β thì ta chỉ giữ lại những điểm ảnh có thứ tự trong đoạn $\left[\frac{\beta}{2} - \frac{\alpha}{2} ; \frac{\beta}{2} + \frac{\alpha}{2}\right]$ và nằm trong các dòng nằm trong đoạn $\left[\frac{\beta}{2} - \frac{\alpha}{2} ; \frac{\beta}{2} + \frac{\alpha}{2}\right]$

8. Cắt ảnh theo khung hình tròn

- Để thực hiện việc cắt ảnh theo khung hình tròn, ta chỉ cần xác định được tâm hình tròn nằm ở đâu trong ma trận ảnh và bán kính của nó, vì ảnh là hình vuông nên tâm của hình tròn cũng chính là tâm hình vuông và bán kính của nó bằng ½ cạnh hình vuông.
- Sau khi xác định được tâm và bán kính, ta thực hiện vòng lặp qua tất cả các điểm ảnh và tính khoảng cách từ điểm ảnh đó tới tâm, nếu khoảng cách lớn hơn bán kính thì tô màu đen, ngược lại thì giữ nguyên.

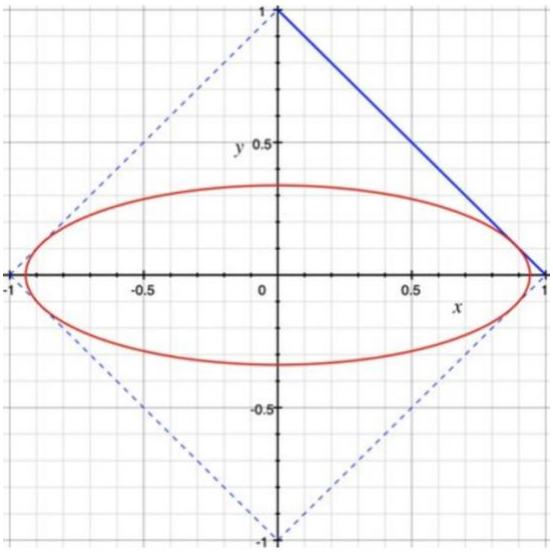
9. Cắt ảnh theo khung là 2 hình elip chéo nhau

- Ý tưởng cắt ảnh theo khung là 2 hình elip chéo nhau cũng tương tự như hình tròn, nhưng nó khó hơn ở chỗ là phải xác định được tiêu cực, tiêu điểm, độ dài trục lớn, độ dài trục bé của 2 hình elip chéo nhau.
- Ta có công thức về mối quan hệ giữa các thông số của elip nội tiếp và hình vuông như sau:

$$a^2 + b^2 = \frac{x^2}{2}$$

Trong đó:

- a là ½ độ dài trục lớn elip
- b là ½ độ dài trục bé elip
- x là độ dài cạnh hình vuông



Nguồn: quora.com

- Nhìn vào hình minh họa, ta có thể thấy vì nội tiếp hình vuông nên nên độ dài trục lớn elip luôn phải nhỏ hơn độ dài đường chéo hình vuông một khoảng nhỏ bằng ε (cụ thể trong đồ án này là $\frac{x}{10}$) nên ta có công thức sau:

$$2a = \sqrt{2}x - \varepsilon \implies a = \frac{\sqrt{2}}{2}x - \frac{\varepsilon}{2} \text{ với } \varepsilon \text{ là một hằng số}$$

- Từ đó, ta tính b theo công thức thứ nhất phía trên, ta được:

$$b = \sqrt{\frac{x^2}{2} - a^2}$$

- Tiếp đó, ta tính xác định tiêu cự bằng cách tính c là ½ tiêu cự:

$$c = \sqrt{a^2 - b^2}$$

- Sau khi đã có được a, b và c ta dễ dàng xác định được vị trí hai tiêu điểm của elip, việc còn lại là lặp qua các điểm ảnh, tính tổng khoảng cách từ chúng tới hai tiêu điểm, nếu tổng khoảng cách đó lớn hơn độ dài trục lớn thì điểm đó tô màu đen, ngược lại thì giữ nguyên.

III. Mô tả các hàm

1. Các thư viện cần thiết được sử dụng

- Các thư viện được sử dụng trong đồ án này bao gồm:
  - PIL (open(), save() từ Image) để đọc và ghi ảnh.
  - Matplotlib để hiển thị ảnh.
  - Numpy để tính toán và xử lí ma trận điểm ảnh.

2. **Hàm thay đổi độ sáng**
   - Nguyên mẫu hàm: `adjust_brightness(img_arr, scalar)`
   - Đầu vào: *img_arr* là ma trận điểm ảnh (numpy array), *scalar* là hằng số.
   - Đầu ra: ma trận kết quả thu được sau thực hiện cộng *img_arr* với *scalar*.
   - Các bước thực hiện:
     - Ta thực hiện phép cộng *img_arr* với *scalar* (cụ thể là 50) để tăng độ sáng cho bức ảnh. Vì *img_arr* là numpy array nên khi thực hiện phép cộng với *scalar* sẽ xảy ra broadcasting là làm cho tất cả các giá trị trong img_arr tăng lên một lượng bằng 50.
     - Sau khi thực hiện phép cộng xong, ta dùng hàm *np.clip(img_arr, 0, 255)* để giới hạn lại miền giá trị các kênh màu của tất cả các điểm ảnh trong khoảng hợp lệ là từ 0 đến 255.
3. **Hàm thay đổi độ tương phản**
   - Nguyên mẫu hàm: `adjust_contrast(img_arr, scalar)`
   - Đầu vào: *img_arr* là ma trận điểm ảnh (numpy array), *scalar* là hằng số.
   - Đầu ra: ma trận kết quả thu được sau thực hiện nhân *img_arr* với *scalar*.
   - Các bước thực hiện:
     - Ta thực hiện phép nhân *img_arr* với *scalar* (cụ thể là 1.5) để tăng độ tương phản cho bức ảnh. Vì *img_arr* là numpy array nên khi thực hiện phép nhân với *scalar* sẽ xảy ra broadcasting là làm cho tất cả các giá trị trong img_arr gấp lên 1.5 lần.
     - Sau khi thực hiện phép nhân xong, ta dùng hàm *np.clip(img_arr, 0, 255)* để giới hạn lại miền giá trị các kênh màu của tất cả các điểm ảnh trong khoảng hợp lệ là từ 0 đến 255.
4. **Hàm lật ảnh ngang / dọc**
   - Nguyên mẫu hàm: `flip(img_arr, direction)`
   - Đầu vào: *img_arr* là ma trận điểm ảnh (numpy array), *direction* là hướng mong muốn lật gồm hai giá trị *'horizontal'* – ngang hoặc *'vertical'* – dọc
   - Đầu ra: ma trận kết quả sau khi đã thực hiện phép lật.
   - Các bước thực hiện:
     - Kiểm tra giá trị của *direction* là ngang hay là dọc
     - Nếu *direction* là ngang thì ta thực hiện lật ảnh theo chiều ngang bằng cách sử dụng indexing và slicing như sau: *img_arr[:, ::-1]*. Tức là, index đầu tiên ta lấy hết tất cả các phần tử theo trục axis=0 (tất cả các dòng) bằng cách dùng slicing ":", sau đó index thứ 2 ta lấy hết các phần tử theo trục axis=1 (tất cả các cột) nhưng theo chiều ngược lại bằng cách dùng slicing "::-1".
     - Ngược lại nếu direction là dọc thì ta thực hiện lật ảnh theo chiều dọc bằng cách sử dụng indexing và slicing như sau: *img_arr[::-1]*. Tức là, index đầu tiên ta lấy hết các phần tử theo trục axis=0 (tất cả các dòng) nhưng theo chiều ngược lại.

5. **Hàm chuyển đổi thành ảnh xám**
   - Nguyên mẫu hàm: `convert_gray_image(img_arr)`
   - Đầu vào: *img_arr* là ma trận điểm ảnh (numpy array)
   - Đầu ra: ma trận kết quả sau khi đã chuyển đổi.
   - Các bước thực hiện:
     - Tạo một bản copy cho *img_arr* và lưu vào một biến khác là *img_res*
     - Kiểm tra xem số kênh màu của ảnh gốc có phải là 4 hay không, vì bức ảnh gốc ban đầu có thể là ảnh rgba (thêm 1 kênh cho độ trong so với ảnh màu thông thường). Nếu số kênh màu của ảnh gốc là 4, ta thực hiện xóa kênh màu đó đi bằng cách dùng hàm *np.delete()* như sau:

       img_res = np.delete(img_res, 3, axis=2)
     - Tính trung bình 3 giá trị kênh màu R, G, B kết quả thu được lưu vào *img_res* bằng cách dùng hàm *np.apply_along_axis()* giúp áp dụng hàm *np.mean()* cho từng điểm ảnh một cách đồng thời mà không cần dùng vòng lặp:

       img_res = np.apply_along_axis(np.mean, 2, img_res)
6. **Hàm chuyển đổi thành ảnh sepia**
   - Nguyên mẫu hàm: `convert_sepia_image(img_arr)`
   - Đầu vào: *img_arr* là ma trận điểm ảnh (numpy array)
   - Đầu ra: ma trận kết quả sau khi đã chuyển đổi.
   - Các bước thực hiện:
     - Tạo một bản copy cho *img_arr* và lưu vào một biến khác là *img_res*
     - Tính toán lại các giá trị của 3 kênh màu R, G, B theo công thức bằng cách dùng indexing và slicing.
     - Sử dụng hàm *np.clip()* để giới hạn lại miền giá trị các kênh màu của tất cả các điểm ảnh trong khoảng hợp lệ.
7. **Hàm lấy điểm ảnh**
   - Nguyên mẫu hàm: `get_pixel(img_arr, i, j)`
   - Đầu vào: *img_arr* là ma trận điểm ảnh (numpy array), tọa độ cột *i* và dòng *j* của vector điểm ảnh muốn lấy.
   - Đầu ra: vector điểm ảnh nằm tại dòng i và cột j
   - Các bước thực hiện:
     - Kiểm tra xem dòng i và cột j có nằm trong ma trận hay không
     - Nếu có thì trả về vector điểm ảnh tại đó, ngược lại trả về vector 0 với cỡ bằng với số kênh màu.
8. **Hàm tính convolution trên một điểm ảnh**
   - Nguyên mẫu hàm: `calculate_convo_pixel(img_arr, kernel, i, j)`
   - Đầu vào: *img_arr* là ma trận điểm ảnh (numpy array), *kernel* là ma trận 3x3 dùng để tính convolution, tọa độ cột *i* và dòng *j* của vector điểm ảnh muốn tính convolution.
   - Đầu ra: giá trị kết quả của việc tính convolution tại điểm ảnh đó.

- Các bước thực hiện:
  - Tạo một ma trận *matrix* rỗng (kiểu list)
  - Sử dụng hai vòng lặp lồng nhau, vòng 1 lặp trong đoạn [i – 1; i + 1] và vòng 2 lặp trong đoạn [j – 1; j + 1], ở mỗi bước lặp ta thêm các điểm ảnh lấy được vào ma trận *matrix* tạo ra một ma trận có kích thước bằng kernel (3x3).
  - Sau đó reshape hai ma trận *matrix* và *kernel* để có thể nhân lại được với nhau bằng broadcasting, sau đó dùng *np.sum()* tính tổng và thu được kết quả convolution tại điểm ảnh vị trí dòng i, cột j.

9. Hàm tính convolution trên toàn bộ ma trận

- Nguyên mẫu hàm: `do_convolution(img_arr, kernel)`
- Đầu vào: *img_arr* là ma trận điểm ảnh (numpy array), *kernel* là ma trận 3x3 dùng để tính convolution.
- Đầu ra: ma trận kết quả sau khi thực hiện quá trình convolution.
- Các bước thực hiện:
  - Tạo một ma trận *img_res* có kích thước giống ma trận *img_arr*.
  - Dùng vòng lặp qua tất cả các điểm ảnh trong *img_arr*, tính convolution trên các điểm ảnh đó bằng cách dùng hàm *calculate_convo_pixel()* và cập nhật lại cho ma trận *img_res*.

10. Hàm làm mờ ảnh

- Nguyên mẫu hàm: `blur(img_arr)`
- Đầu vào: *img_arr* là ma trận điểm ảnh (numpy array)
- Đầu ra: ma trận kết quả sau khi đã xử lí
- Các bước thực hiện:
  - Tạo ma trận *kernel* cho việc làm mờ ảnh
  - Tính toán convolution cho ma trận *img_arr* với *kernel* mới tạo bằng cách dùng hàm *do_convolution(img_arr, kernel)*.

11. Hàm làm sắc nét ảnh

- Nguyên mẫu hàm: `sharpen(img_arr)`
- Đầu vào: *img_arr* là ma trận điểm ảnh (numpy array)
- Đầu ra: ma trận kết quả sau khi đã xử lí
- Các bước thực hiện:
  - Tạo ma trận *kernel* cho việc làm sắc nét ảnh
  - Tính toán convolution cho ma trận *img_arr* với *kernel* mới tạo bằng cách dùng hàm *do_convolution(img_arr, kernel)*.

12. Hàm cắt ảnh theo kích thước (cắt ở trung tâm)

- Nguyên mẫu hàm: `crop(img_arr,size)`
- Đầu vào: *img_arr* là ma trận điểm ảnh (numpy array), *size* là kích thước cần cắt.
- Đầu ra: ma trận kết quả sau khi đã xử lí
- Các bước thực hiện:

- Tính lại *size* bằng ½ giá trị nhỏ nhất giữa size và kích thước ảnh để tránh cắt với *size* lớn hơn kích thước thật của ảnh.
- Tính vị trí trung tâm *center* của ảnh bằng ½ kích thước ảnh.
- Dùng indexing để lấy những điểm ảnh nào nằm trong đoạn [center - size ; center + size]

13. Hàm tính khoảng cách từ một điểm tới trung tâm

- Nguyên mẫu hàm: `calc_dist_to_center(r, i, j)`
- Đầu vào: *r* là vị trí trung tâm, *i* và *j* là vị trí điểm ảnh
- Đầu ra: giá trị khoảng cách từ điểm ảnh tới trung tâm
- Các bước thực hiện:
  - Tính giá trị khoảng cách từ điểm ảnh tới trung tâm theo công thức:

$$kc = \sqrt{(i-r)^2 + (j-r)^2}$$

14. Hàm cắt ảnh theo khung hình tròn

- Nguyên mẫu hàm: `crop_circle_frame(img_arr)`
- Đầu vào: *img_arr* là ma trận điểm ảnh (numpy array)
- Đầu ra: ma trận kết quả sau khi xử lí cắt ảnh.
- Các bước thực hiện:
  - Xác định trung tâm điểm ảnh và bán kính bằng cách lấy kích thước chia 2
  - Lặp qua tất cả các điểm ảnh, kiểm tra xem khoảng cách từ điểm ảnh đó tới trung tâm có lớn hơn bán kính hay không.
  - Nếu lớn hơn thì tô lại bằng màu đen, ngược lại thì giữ nguyên.

15. Hàm tìm tiêu điểm của elip

- Nguyên mẫu hàm: `find_focus(n)`
- Đầu vào: *n* là kích thước của ảnh (ảnh vuông cạnh có độ dài là n)
- Đầu ra: mảng tọa độ các tiêu điểm và độ dài trục lớn của elip.
- Các bước thực hiện dựa trên ý tưởng đã trình bày rõ ở mục I:
  - Tìm a, b và c
  - Sau khi có a, b và c ta xác định mảng tọa độ các tiêu điểm của 2 hình elip.

16. Hàm tính tổng khoảng cách từ 1 điểm ảnh tới hai tiêu điểm của elip

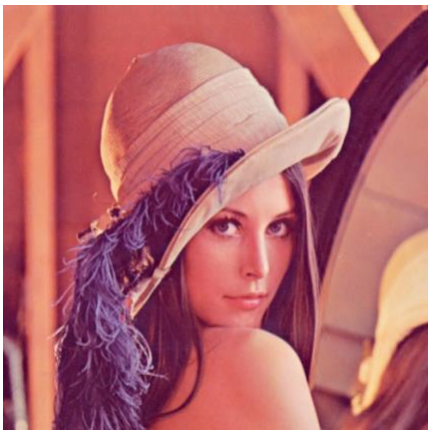
- Nguyên mẫu hàm: `sum_dist_to_focus (focus, i, j, main_diag)`
- Đầu vào: *focus* là mảng tọa độ của các tiêu điểm, *i* và *j* là tọa độ của điểm ảnh, *main_diag* là giá trị boolean cho biết xét hình elip trên đường chéo chính hay phụ.
- Đầu ra: tổng khoảng cách từ điểm ảnh tới hai tiêu điểm của elip nằm trên đường chéo chính hoặc phụ
- Các bước thực hiện:
  - Kiểm tra biến *main_diag* để biết elip nằm trên đường chéo chính hay phụ.
  - Tính tổng khoảng cách từ điểm ảnh tới hai tiêu điểm của elip đó.

17. Hàm cắt ảnh theo khung là 2 hình elip chéo nhau

- Nguyên mẫu hàm: `crop_ellipses_crossover(img_arr)`
- Đầu vào: *img_arr* là ma trận điểm ảnh (numpy array)
- Đầu ra: ma trận kết quả sau khi đã xử lí
- Các bước thực hiện:
  - Tìm mảng tọa độ các tiêu điểm và độ dài trục lớn của elip bằng cách dùng hàm *find_focus()*
  - Lặp qua các điểm ảnh, tính tổng khoảng từ nó tới 2 tiêu điểm của elip trên đường chéo chính và tổng khoảng cách từ nó tới 2 tiêu điểm của elip trên đường chéo phụ.
  - Kiểm tra hai tổng khoảng cách đó nhỏ hơn độ dài trục chính hay không, nếu cả hai đều nhỏ hơn thì giữ nguyên màu, ngược lại nếu một trong hai không thỏa thì tô màu đen.

IV. Kết quả chạy chương trình

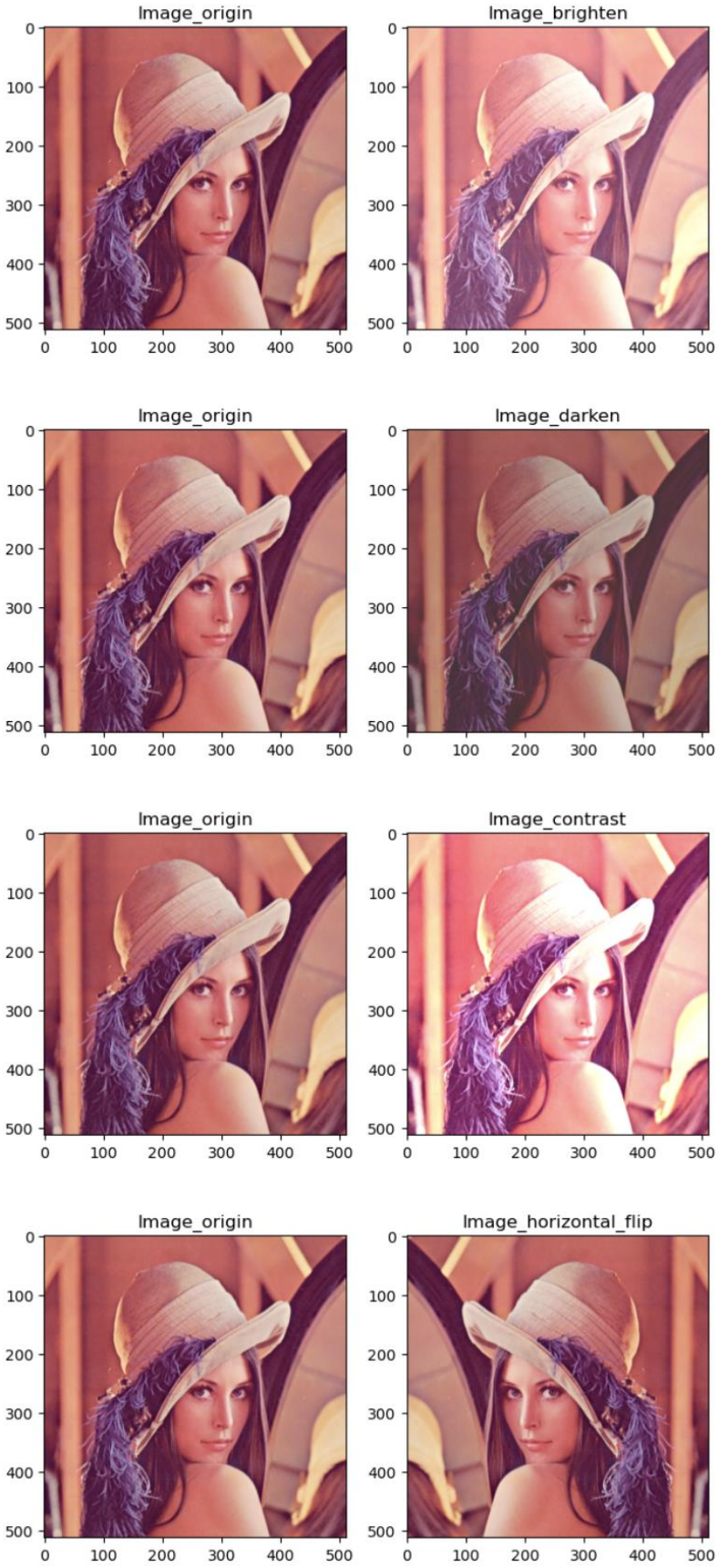
Ta chạy thử nghiệm với hai bức hình với kích thước 512x512 như sau:

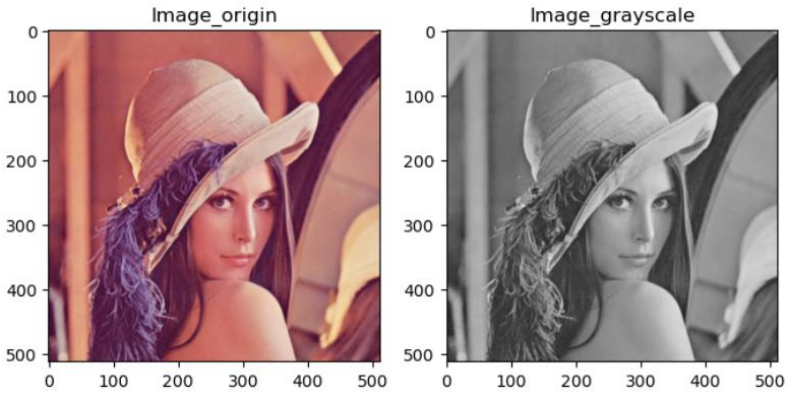


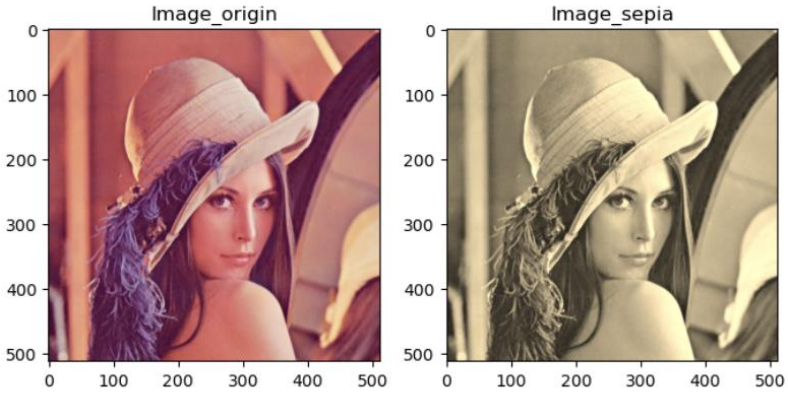
img.png

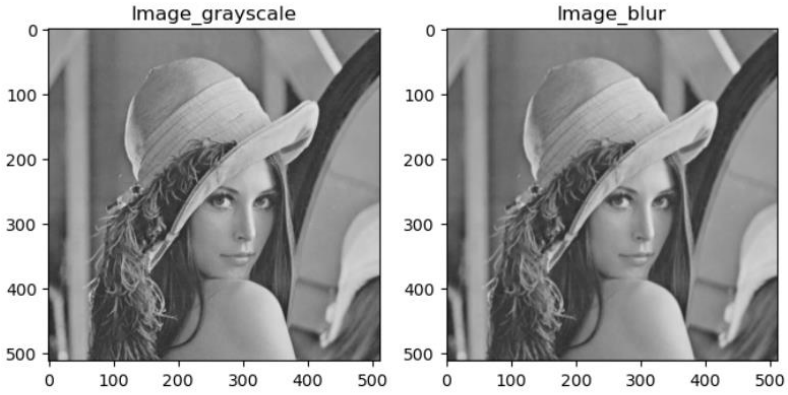
img-1.png

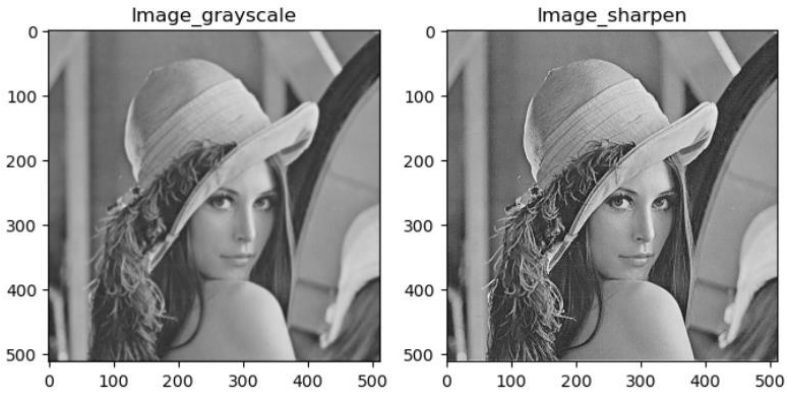
- Kết quả chạy với bức hình thứ nhất

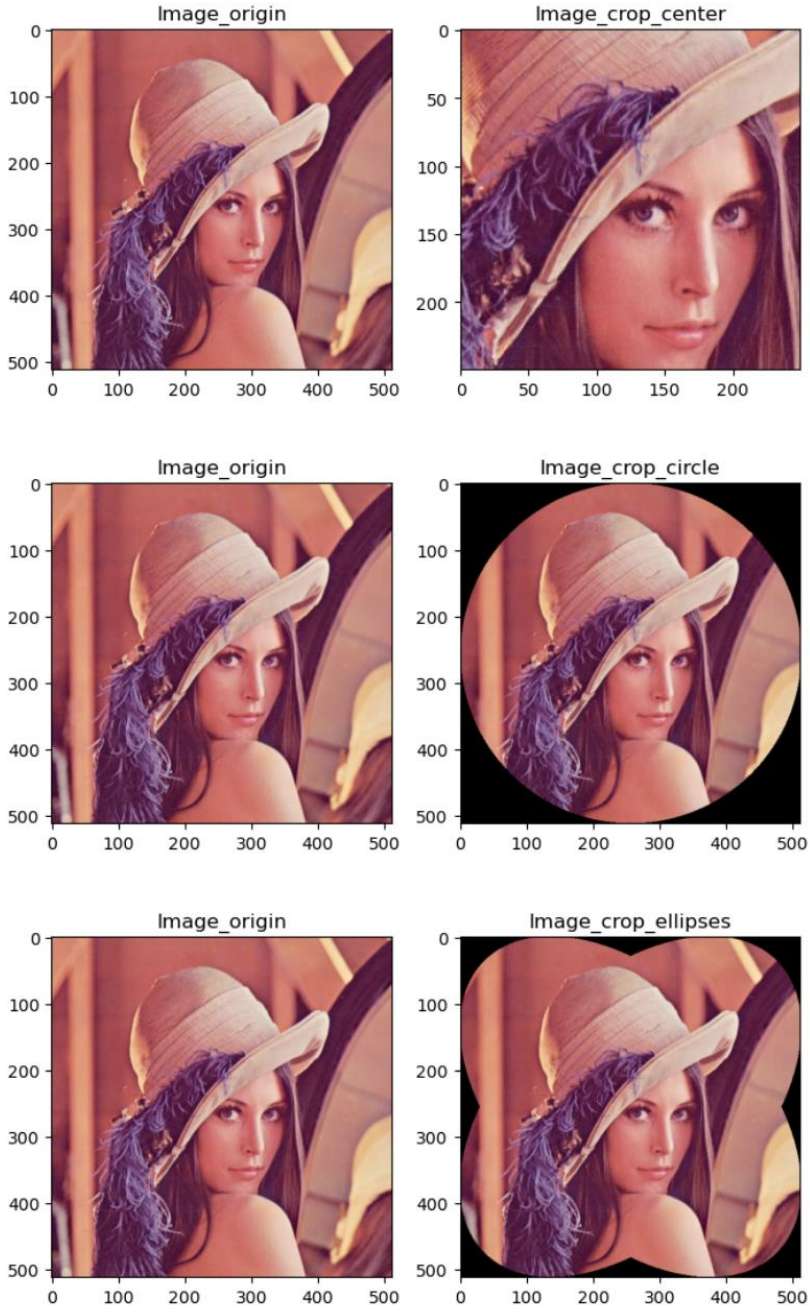

Image_origin
Image_brighten
Image_origin
Image_darken
Image_origin
Image_contrast
Image_origin
Image_horizontal_flip


Image_origin
Image_grayscale

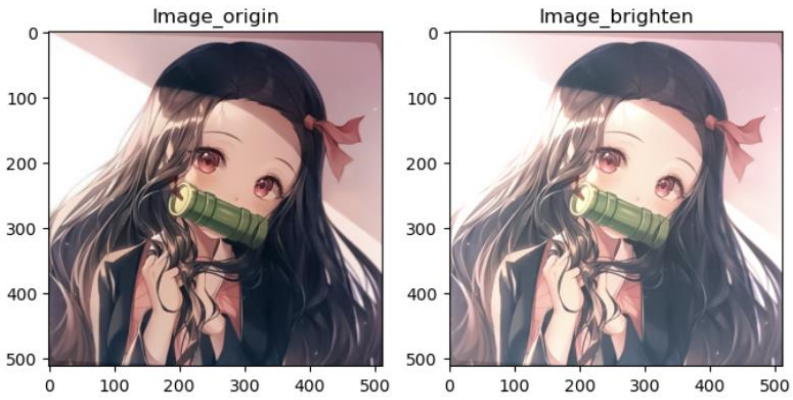

Image_origin
Image_sepia

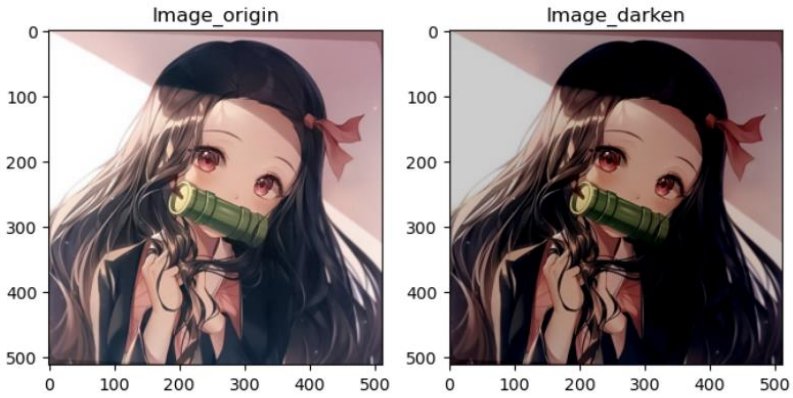

Image_grayscale
Image_blur

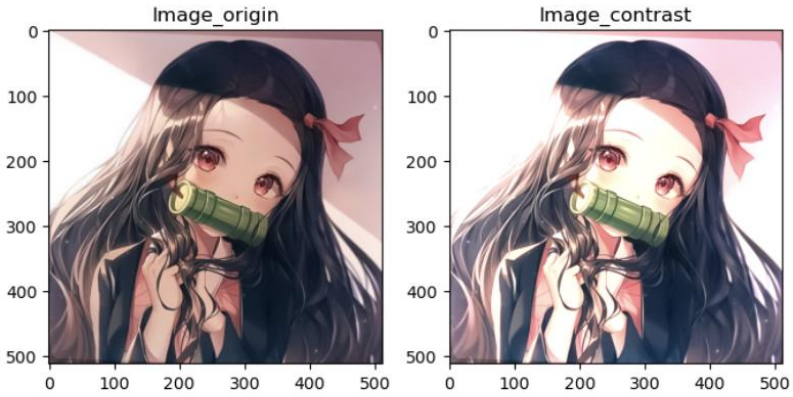

Image_grayscale
Image_sharpen

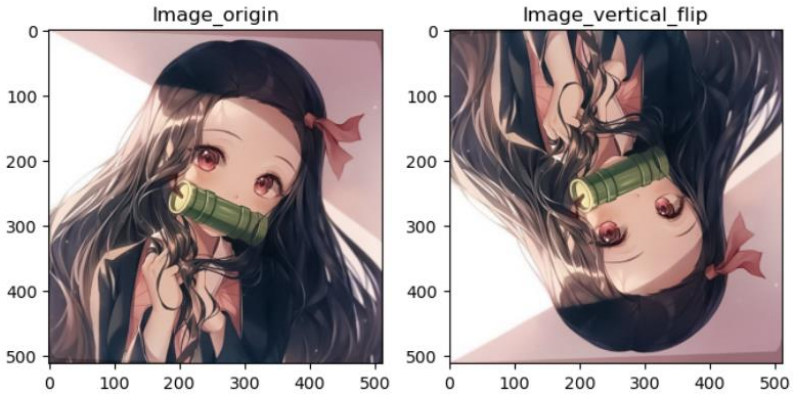


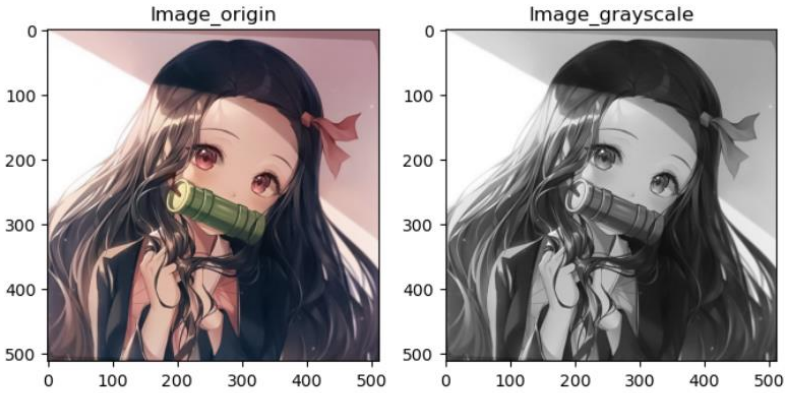
- Kết quả chạy với bức hình thứ hai

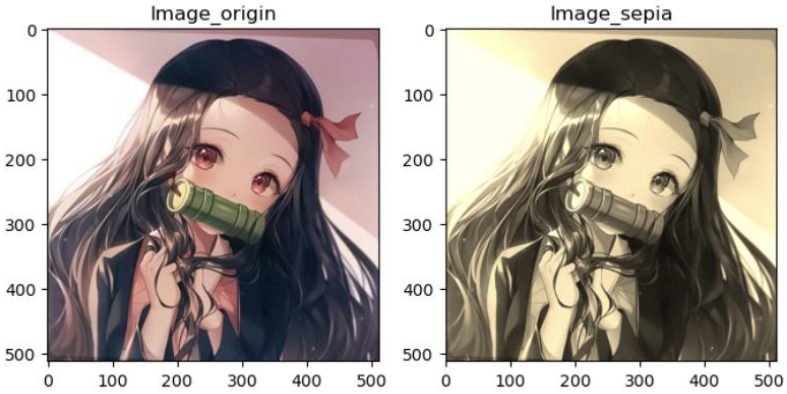

Image_origin
Image_brighten

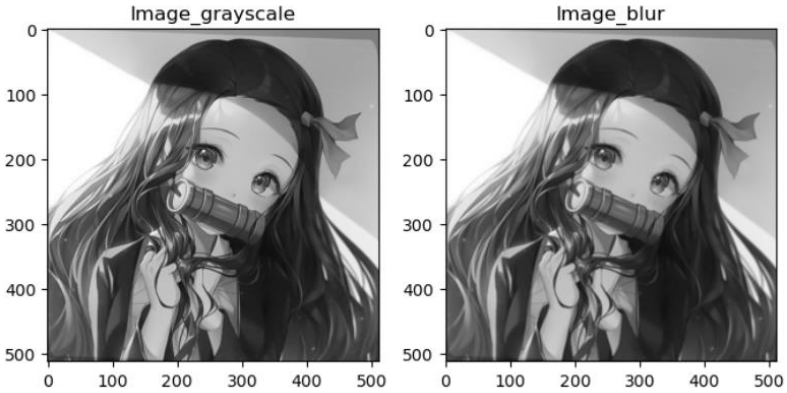

Image_origin
Image_darken

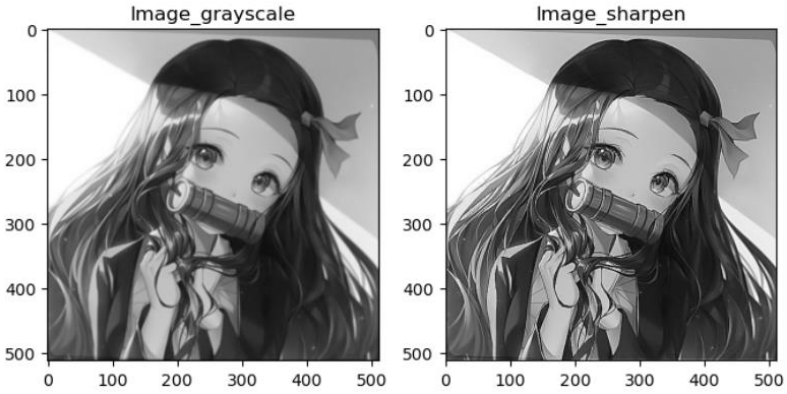

Image_origin
Image_contrast

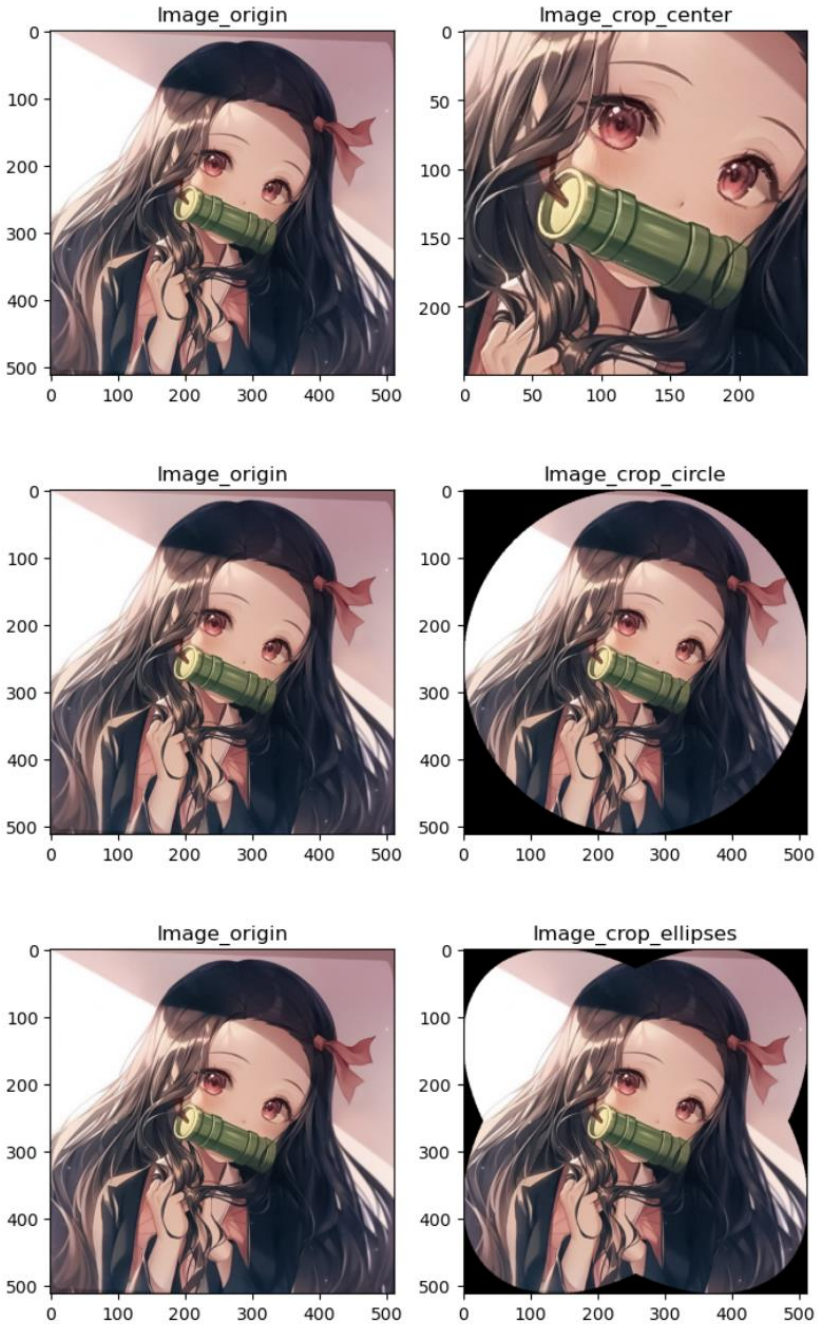

Image_origin
Image_vertical_flip


Image_origin
Image_grayscale


Image_origin
Image_sepia


Image_grayscale
Image_blur


Image_grayscale
Image_sharpen


Image_origin
Image_crop_center
Image_origin
Image_crop_circle
Image_origin
Image_crop_ellipses

V. Tài liệu tham khảo

[1] Lab 03, hướng dẫn Project 02

[2] en.wikipedia/kernel image processing

[3] dyclassroom.com/how to convert color image into sepia image

[4] vietjack.me/phuong trinh duong elip

[5] users.math.uoc.gr/cong thuc moi quan he giua elip noi tiep va hinh vuong

[6] stackoverflow.com/questions/31633635/what is the meaning of "a[::-1]" in Python?

[7] freecodecamp.org/news/slicing and indexing in python