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

BÁO CÁO MÔN HỌC
TOÁN ỨNG DỤNG VÀ THỐNG KÊ

Lab 02: Image Processing

Sinh viên thực hiện

Họ và tên	:	Võ Nguyễn Hoàng Kim
Mã số sinh viên	:	21127090
Lớp	:	21CLC07

Phụ lục

I. Thông tin sinh viên

Họ tên: Võ Nguyễn Hoàng Kim

Mã số sinh viên: 21127090

Lớp: 21CLC07

II. Các chức năng đã hoàn thành

1. Thay đổi độ sáng cho ảnh
2. Thay đổi độ tương phản
3. Lật ảnh (ngang – dọc)
4. Chuyển đổi ảnh RGB thành ảnh xám/sepia
5. Làm mờ/sắc nét ảnh
6. Cắt ảnh theo kích thước
7. Cắt ảnh theo khung hình tròn

III. Mô tả chi tiết

Các hàm dưới đây sẽ trả về 2 tham số lần lượt là ma trận ảnh (sau khi đã xử lý) và tên chức năng tương ứng.

1. Thay đổi độ sáng cho ảnh

a. Ý tưởng thực hiện

- Để thay đổi độ sáng của ảnh, ta cộng tất cả điểm màu có trong ảnh với một số bất kì (có thể là số âm hoặc số dương).
- Nếu là số âm: Độ sáng của ảnh sẽ giảm đi.
- Nếu là số dương: Độ sáng của ảnh sẽ tăng lên.

b. Mô tả chi tiết

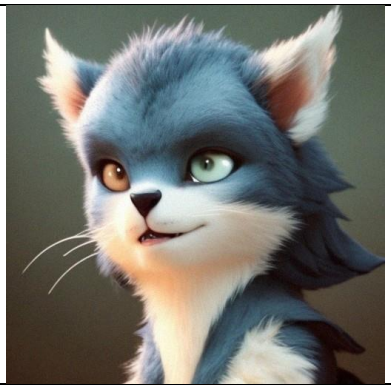
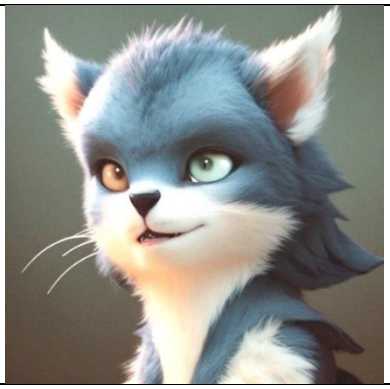
Cài đặt hàm **brightenImage(img, b)**

- Tham số truyền vào: *img* là ma trận có được từ việc chuyển đổi ảnh sang ma trận, *b* là số để chỉnh độ sáng của ảnh.
- Để dễ dàng xử lý, ta sẽ chuyển ma trận *img* về mảng 1 chiều và duyệt tuần tự các phần tử của mảng, thực hiện phép cộng giữa các phần tử (là điểm màu) với *b*. Do các điểm màu có giá trị trong khoảng [0, 255] nên khi thực hiện phép cộng, ta cần lưu ý với các giá trị nằm ngoài khoảng đó:
  - Nếu giá trị có được từ phép cộng lớn hơn 255, ta sẽ gán giá trị đó bằng với 255.

- Nếu giá trị có được từ phép cộng bé hơn 0, ta sẽ gán trá trị đó với 0.

- Sau khi thực hiện phép cộng với tất cả phần tử có trong mảng, mảng sẽ được trả về lại kích thước ban đầu của hình ảnh (trở về ma trận với kích thước ban đầu nhưng với các giá trị đã thay đổi) bằng hàm "reshape".

c. Kết quả đạt được

	
Ảnh ban đầu	Ảnh sau khi thay đổi độ sáng (tăng độ sáng)
Thời gian xử lý: 2.86 (s)	

2. Thay đổi độ tương phản

a. Ý tưởng thực hiện

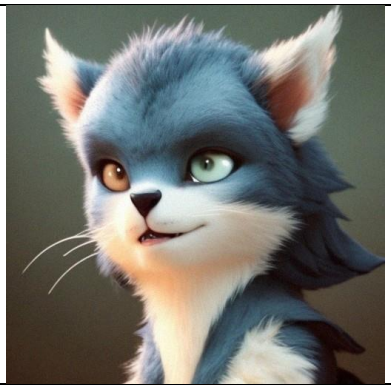
- Để thay đổi độ tương phản của ảnh, ta nhân tất cả điểm màu có trong ảnh với một số bất kì.

b. Mô tả chi tiết

Cài đặt hàm **contrastImage**(*img, a*).

- Tham số truyền vào: *img* là ma trận có được từ việc chuyển đổi ảnh sang ma trận, *a* là số để chỉnh độ tương phản của ảnh.
- Để dễ dàng xử lý, ta sẽ chuyển ma trận *img* về mảng 1 chiều và duyệt tuần tự các phần tử của mảng, thực hiện phép nhân giữa các phần tử (là điểm màu) với *a*. Do các điểm màu có giá trị trong khoảng [0, 255] nên khi thực hiện phép nhân, ta cần lưu ý với các giá trị nằm ngoài khoảng đó.
  - Nếu giá trị có được từ phép nhân lớn hơn 255, ta sẽ gán giá trị đó bằng với 255. Ngược lại sẽ giữu nguyên.
- Sau khi thực hiện phép nhân với tất cả phần tử có trong mảng, mảng sẽ được trả về lại kích thước ban đầu của hình ảnh (trở về ma trận với kích thước ban đầu nhưng với các giá trị đã thay đổi) bằng hàm "reshape".

c. Kết quả đạt được

	
Ảnh ban đầu	Ảnh sau khi thay đổi độ tương phản
Thời gian xử lý: 2.12 (s)	

3. Lật ảnh ngang – dọc

a. Ý tưởng thực hiện

- Do hình ảnh là một ma trận với các phần tử là những điểm màu, do đó, muốn lật hình ảnh, ta sẽ thực hiện lật ma trận tương ứng với chiều mà đề bài mong muốn.

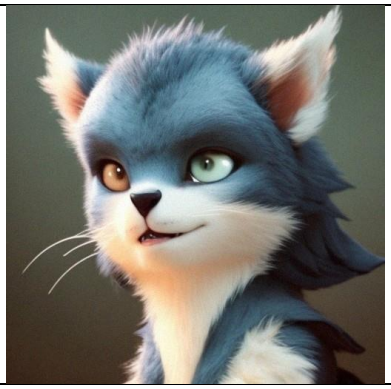
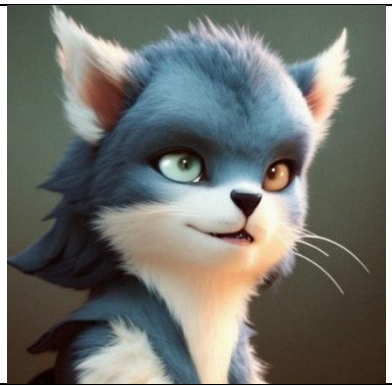
b. Mô tả chi tiết

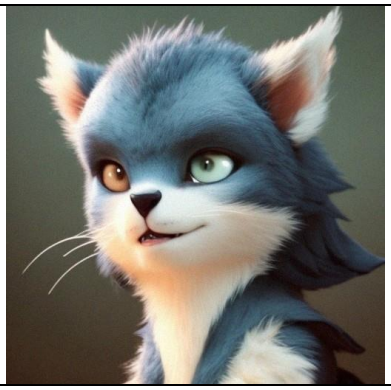
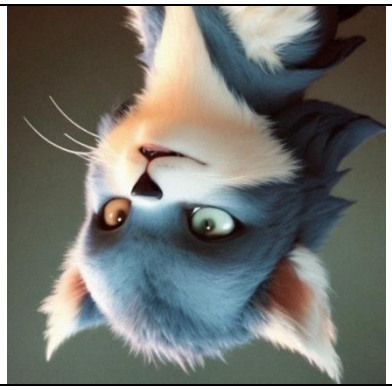
Cài đặt hàm **flipImage**(*img, type*).

- Tham số truyền vào: *img* là ma trận có được từ việc chuyển đổi ảnh sang ma trận, *type* là một chuỗi để nhận dạng cách lật mà người dùng mong muốn.
- Ta sử dụng một ma trận mới (được gọi là *resImage)* có cùng kích thước với ma trận ban đầu (*img)*
- Nếu Type = 'Vertical' – Thực hiện lật ảnh dọc:
  - Để thực hiện lật ảnh theo chiều dọc, ta chỉ cần gán giá trị các phần tử của *img* theo trình tự từ dưới lên cho các phần tử của *resImg* theo trình tự từ trên xuống.
- Nếu Type = 'Horizontal' – Thực hiện lật ảnh ngang:
  - Tương tự như cách trên, để lật ảnh ngang, ta chỉ cần gán giá trị các phần tử của *img* theo trình tự từ dưới phải sang trái cho các phần tử của *resImg* theo trình tự từ trái sang phải.

c. Kết quả đạt được

Lật ngang

	
Ảnh ban đầu	Ảnh sau khi lật ngang
Thời gian xử lý: 0.19 (s)	

Lật dọc	
	
Ảnh ban đầu	Ảnh sau khi lật dọc
Thời gian xử lý: 0.0 (s)	

4. Chuyển đổi ảnh RGB thành ảnh xám/sepia

a. Ý tưởng thực hiện

Chuyển đổi sang ảnh xám:

- Từ ảnh RGB ban đầu (với 3 kênh màu lần lượt red – green – blue), ta thực hiện phép tính sau với kênh màu tương ứng để ra được điểm màu xám:
  - $0.299 \cdot Red + 0.587 \cdot Green + 0.114 \cdot Blue$

Chuyển đổi sang ảnh sepia:

- Ảnh sepia sẽ có chút khác biệt, từ ảnh RGB ban đầu (với 3 kênh màu lần lượt red – green – blue), ta thực hiện phép tính sau

- tr = 0.393R + 0.769G + 0.189B.
- tg = 0.349R + 0.686G + 0.168B.
- tb = 0.272R + 0.534G + 0.131B.

- Trong đó, tr, tg, tb là các giá trị mới của điểm màu red – green – blue. Sau khi có được các kết quả đó, ta thực hiện gán giá trị lại cho ảnh để được ảnh sepia

b. Mô tả chi tiết

Cài đặt hàm **grayImage**(*img*)

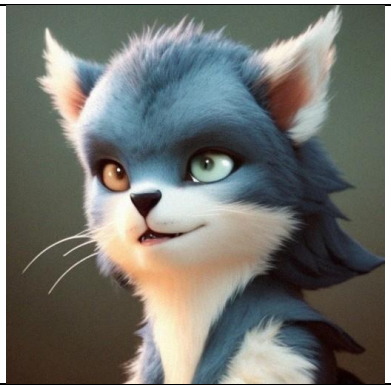
- Tham số truyền vào: *img* là ma trận có được từ việc chuyển đổi ảnh sang ma trận.
- Chuyển đổi ma trận ban đầu thành mảng 1 chiều để dễ dàng lấy các kênh màu thực hiện phép tính.
- Với ảnh RGB, các phần tử của một điểm màu lần lượt là giá trị của Red – Green – Blue.
- Dựa theo phép tính ở phần ý tưởng: 0.3*Red + 0.59*Green + 0.11*Blue, kết quả của phép tính được gán vào một biến và thực hiện kiểm tra.
- Việc kiểm tra để chắc chắn rằng điểm màu sẽ có giá trị trong khoảng [0,255]. Nếu giá trị điểm màu đó lớn hơn 255, ta thực hiện gán giá trị 255 cho điểm màu đó.
- Sau khi thực hiện với tất cả điểm màu có trong ảnh, mảng sẽ được trả về lại kích thước ban đầu của hình ảnh (trở về ma trận với kích thước ban đầu nhưng với các giá trị đã thay đổi) bằng hàm "reshape".

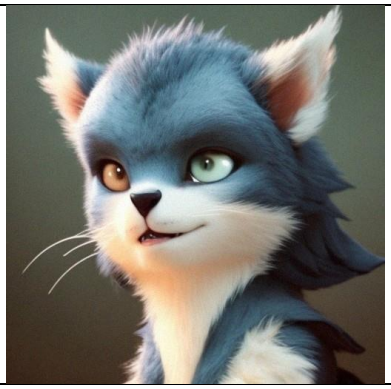
Cài đặt hàm **sepiaImage**(*img*)

- Chuyển đổi ma trận ban đầu thành mảng 1 chiều để dễ dàng lấy các kênh màu thực hiện phép tính.
- Dựa theo phép tính ở phần ý tưởn, ta thu được 3 giá trị mới là tr, tg, tb và thực hiện kiểm tra.
- Việc kiểm tra để chắc chắn rằng các giá trị đó trong khoảng [0,255]. Nếu giá trị nào lớn hơn 255, ta thực hiện gán 255 cho giá trị đó, ngược lại thì giữ nguyên.
- Sau khi 3 kênh màu tr, tg, tb được kiểm tra và gán các giá trị hợp lệ, ta thực hiện gán điểm màu (với 3 kênh màu mới) vào vị trí cũ tương ứng.
- Sau khi thực hiện với tất cả điểm màu có trong ảnh, mảng sẽ được trả về lại kích thước ban đầu của hình ảnh (trở về ma trận với kích thước ban đầu nhưng với các giá trị đã thay đổi).

c. Kết quả đạt được

Ảnh xám

	
Ảnh ban đầu	Ảnh sau khi thành xám
Thời gian xử lý: 1.02 (s)	

Ảnh Sepia	
	
Ảnh ban đầu	Ảnh sau khi thành sepia
Thời gian xử lý: 3.39 (s)	

5. Làm mờ/sắc nét ảnh

a. Ý tưởng thực hiện

- Sử dụng phương pháp (operation) Gassian blur 3x3 để làm mờ ảnh.
- Sử dụng phương pháp (operation) Sharpen để làm nét ảnh.

b. Mô tả chi tiết

Cài đặt hàm **blurImage**(*img*)

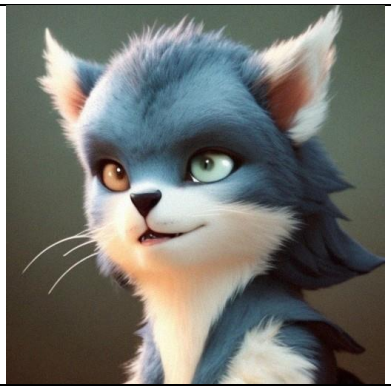
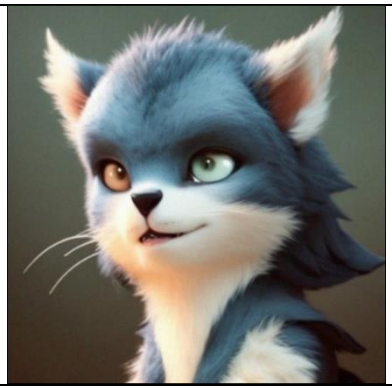
- Tham số truyền vào: *img* là ma trận có được từ việc chuyển đổi ảnh sang ma trận.
- Ta thực hiện duyệt từng điểm màu trong ma trận ảnh ban đầu. Với mỗi điểm màu, ta sẽ có một ma trận A tương ứng (có kích thước là 3x3, các giá trị và vị trí của chúng được lấy từ ma trận ảnh ban đầu) sao cho điểm màu đang xét nằm ở vị trí trung tâm.

- Từ ma trận A, ta thực hiện phép tính convolution (tích chập) và cộng tất cả lại với nhau, cho ra kết quả. Kết quả này sẽ được gán vào điểm màu mà ta đang xét ban đầu.
- Đối với các điểm màu nằm ở rìa, ta sẽ không tìm được ma trận A. Do đó, ta sẽ không thực hiện phép tính đối với các điểm đó mà sẽ gán nó cho giá trị 0.

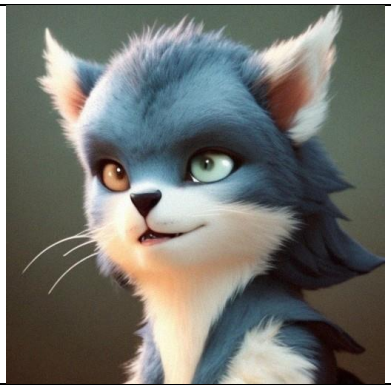
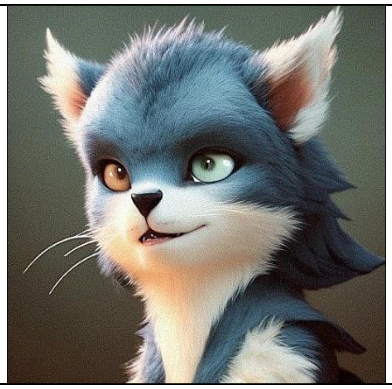
sharpenImage(*img*)

- Tham số truyền vào: *img* là ma trận có được từ việc chuyển đổi ảnh sang ma trận.
- Ta thực hiện duyệt từng điểm màu trong ma trận ảnh ban đầu. Với mỗi điểm màu, ta sẽ có một ma trận A tương ứng (có kích thước là 3x3, các giá trị và vị trí của chúng được lấy từ ma trận ảnh ban đầu) sao cho điểm màu đang xét nằm ở vị trí trung tâm.
- Từ ma trận A, ta thực hiện phép tính convolution (tích chập) và cộng tất cả lại với nhau, cho ra kết quả. Kiểm tra kết quả đó:
  - Nếu kết quả vừa tìm được bé hơn 0, ta sẽ gán giá trị cho nó bằng 0.
  - Nếu kết quả vừa tìm được lớn hơn 255, ta sẽ gán giá trị cho nó bằng 255.
- Sau khi thực hiện kiểm tra, kết quả sẽ được gán vào điểm màu mà ta đang xét ban đầu.
- Đối với các điểm màu nằm ở rìa, ta sẽ không tìm được ma trận A. Do đó, ta sẽ không thực hiện phép tính đối với các điểm đó mà sẽ gán nó cho giá trị 0.

c. Kết quả đạt được

<table>
<tr><td colspan="2">Làm mờ</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>Ảnh ban đầu</td><td>Ảnh sau khi làm mờ</td></tr>
<tr><td colspan="2">Thời gian xử lý: 6.91 (s)</td></tr>
</table>

Làm sắc nét

	
Ảnh ban đầu	Ảnh sau khi làm sắc nét
Thời gian xử lý: 9.04 (s)	

6. Cắt ảnh theo kích thước

a. Ý tưởng thực hiện

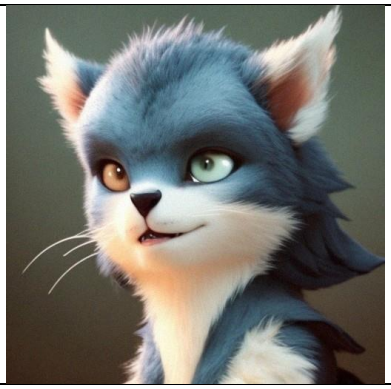
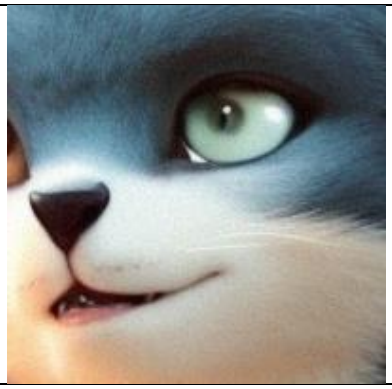
- Với việc cắt ảnh với kích thước mong muốn, ta thực hiện sao chép ma trận ban đầu (ảnh ban đầu) với kích thước mà người dùng cần cắt.
- Tuy nhiên, để có thể cắt trung tâm, ma trận (ảnh trả ra) cần sao chép các vùng lân cận của vị trí trung tâm và cho ra một ma trận với kích thước đúng như yêu cầu.

b. Mô tả chi tiết

Cài đặt hàm **cropImage**(*img, height, width*)

- Tham số truyền vào: *img* là ma trận có được từ việc chuyển đổi ảnh sang ma trận. *height* và *width* lần lượt là chiều cao và chiều rộng mà sau khi thực hiện cắt ảnh, ảnh trả về với kích thước đó.
- Để có thể cắt ảnh (cắt trung tâm ảnh), ta sẽ xác định vùng để sao chép (xác định vùng trung tâm tương ứng với kích thước truyền vào).
- Sau khi xác định được vùng, ta thực hiện việc sao chép các phần tử từ ma trận ảnh ban đầu.
- Sau khi việc sao chép kết thúc, ta sẽ được một ma trận ảnh với kích thước là *height* và *width*.

c. Kết quả đạt được

	
Ảnh ban đầu	Ảnh sau khi cắt
Thời gian xử lý: 0.04 (s)	

7. Cắt ảnh theo khung tròn

a. Ý tưởng thực hiện

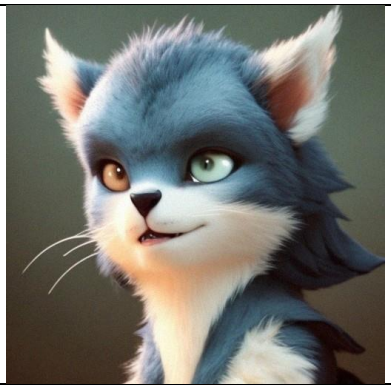
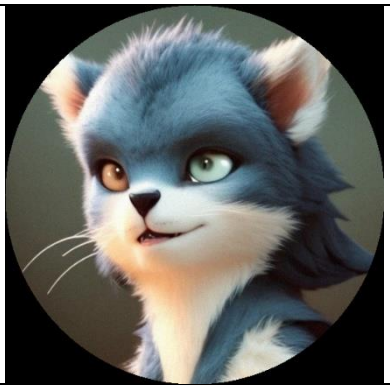
- Cắt hình ảnh theo khung tròn, ta cần xác định tâm của hình tròn (là tâm của ảnh).
- Khi tâm đã được xác định, ta xác định tiếp bán kính của hình tròn và thực hiện cắt ảnh theo hình tròn.

b. Mô tả chi tiết

Cài đặt hàm **circleCropImage**(*img*)

- Tham số truyền vào: *img* là ma trận có được từ việc chuyển đổi ảnh sang ma trận.
- Thực hiện tìm vị trí (tọa độ) của tâm ảnh với vị trí là (chiều dài chia 2, chiều rộng chia 2).
- Bán kính của hình tròn được xác định từ tâm đến cạnh của hình. Tránh trường hợp các hình có chiều dài và chiều rộng khác nhau, bán kính hình tròn sẽ là khoảng cách ngắn nhất của tâm đến cạnh của ảnh.
- Sử dụng hàm ogrid để tạo 2 array với các phần tử (số phần tử tương ứng với một nửa kích thước của hình ảnh ban đầu) nằm dọc và ngang (lần lượt cho chiều cao và chiều rộng)
- Thực hiện tạo một mặt nạ (mask) hình tròn: sử dụng định lý Pythago để xác định các phần tử nào nằm trong hình tròn. Với các phần tử nằm trong hình tròn, giá trị của phần tử đó trong mask sẽ là True, ngược lại là False.
- Từ mask ở trên, các phần tử nào mà tại mask cho ra giá trị False, ta sẽ gán giá trị 0 cho phần tử đó, ngược lại là True thì sẽ giữ nguyên giá trị.
- Kết thúc hàm, hình ảnh ban đầu sẽ được cắt theo khung tròn.

c. Kết quả đạt được

	
Ảnh ban đầu	Ảnh sau khi cắt theo khung tròn
Thời gian xử lý: 0.006 (s)	

8. Hàm main

- Nhận tên ảnh từ bàn phím do người dùng nhập. Lưu ý, do hàm được cài đặt chỉ nhận mỗi tên ảnh, do đó, ảnh mà người dùng nhập vào phải nằm cùng mục với file source code, nếu không, chương trình sẽ báo lỗi "Error! Not find image" và yêu cầu nhập lại.
- Chương trình chỉ xử lý các ảnh có đuôi là png hoặc jpg. Với những ảnh có định dạng khác, chương trình sẽ có thể xảy ra lỗi. Do đó, chỉ nhập các ảnh có dạng jpg và png.
- Sau khi nhận được tên ảnh, chương trình sẽ thực hiện việc đọc ảnh và chuyển nó thành một ma trận để dễ dàng xử lý.
- Chương trình sẽ tiếp tục yêu cầu nhập số thứ tự của chức năng mà người dùng muốn:
  - **1 – Thay đổi độ sáng**: Độ sáng được thay đổi là một số được cài đặt mặc định là 30. Nếu người dùng muốn thay đổi một số khác, cần thay đổi trong source code.
  - **2** – **Thay đổi độ tương phản**: Độ tương phản được thay đổi là một số được cài đặt mặc định là 1.5. Nếu người dùng muốn thay đổi một số khác, cần thay đổi trong source code.
  - **3** – **Lật ảnh**: Khi chọn chức năng này, người dùng sẽ phải nhập thêm chiều mà ảnh sẽ lật (Vertical – lật dọc; Horizontal – lật ngang). Nếu kiểu mà người dùng chọn để ảnh lật theo khác 2 chiều trên, chương trình sẽ không thực hiện lật ảnh (Ảnh trả ra sẽ là ảnh ban đầu và không được xử lý).
  - **4 – Chuyển đổi ảnh RGB thành ảnh xám và sepia**: Thực hiện cả 2 chức năng và kết quả trả ra là 2 ảnh với chức năng tương ứng.
  - **5** – **Làm mờ / sắc nét ảnh**: Thực hiện cả 2 chức năng và kết quả trả ra là 2 ảnh với chức năng tương ứng.
  - **6 – Cắt ảnh theo kích thước**: Kích thước để cắt ảnh được mặc định trong chương trình là 200x200. Nếu người dùng muốn thay đổi kích thước, cần thay đổi trong source code (Kích thước truyền vào phải hợp lệ - nằm trong kích thước của ảnh ban đầu).
  - **7** – **Cắt ảnh theo khung tròn**.

- **0 – Thực hiện tất cả các chức năng:** Mỗi chức năng là một ảnh tương ứng.

- Sau khi các chức năng được thực hiện, ở màn hình console sẽ hiện ra thời gian mà chức năng đó đã thực hiện (tính theo giây).
- Kết quả mà các chức năng trả ra sẽ là một file ảnh (được lưu cùng thư mục với source code) được đặt tên theo cú pháp <tên ảnh>_<tên chức năng>.png

IV. Nguồn tham khảo

https://dyclassroom.com/image-processing-project/how-to-convert-a-color-image-into-sepia-image

https://tranvantri.wordpress.com/2014/09/14/tu-rgb-sang-da-muc-xam-grayscale/

https://www.iostream.vn/article/xu-ly-anh-voi-opencv-do-sang-do-tuong-phan-va-bieu-do-tan-so-histogram-ijcgu1

Kernel (image processing) - Wikipedia