TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

----------o0o----------

BÁO CÁO ĐỒ ÁN TOÁN ỨNG DỤNG VÀ THỐNG KÊ CHO CÔNG NGHỆ THÔNG TIN

Project 2: Image Processing

Tên sinh viên : Đặng Hà Huy
Lớp : 21CLC05
Mã số sinh viên : 21127296
Giảng viên : Phan Thị Phương Uyên

Thành phố Hồ Chí Minh, tháng 7 năm 2023

MỤC LỤC

NỘI DUNG CHÍNH

I/ Giới thiệu

- **Họ tên sinh viên:** Đặng Hà Huy
- **MSSV:** 21127296
- **Lớp:** 21CLC05

II/ Thống kê mức độ hoàn thành

STT	Các hàm chức năng	Mức độ hoàn thành	Đánh giá cá nhân
1	Điều chỉnh độ sáng của ảnh	100%	Hàm chạy tốt và tốc độ thực thi nhanh cho ra đúng yêu cầu
2	Điều chỉnh độ tương phản của ảnh	100%	Hàm chạy tốt và tốc độ thực thi nhanh
3	Lật ảnh ngang/dọc	100%	Hàm chạy tốt với tốc độ nhanh, code ngắn gọn và dễ hiểu
4	Chỉnh về ảnh grayscale/sepia	100%	Hàm chạy tốt và tốc độ thực thi nhanh
5	Làm mờ và nét ảnh	100%	Hàm chạy tốt và nhanh, vẫn có thể chỉnh sửa và cải thiện thêm để xử lý ảnh màu
6	Cắt ảnh (từ tâm của ảnh)	100%	Hàm chạy tốt và tốc độ thực thi nhanh code đơn giản và dễ hiểu
7	Cắt ảnh hình tròn	100%	Hàm chạy tốt và tốc độ thực thi nhanh có thể có cách cải thiện thêm
8	Cắt ảnh dạng 2 hình elip chồng lên nhau	80%	Ảnh đầu ra vẫn chưa hoàn toàn giống với kết quả đầu ra mong muốn
9	Hàm main	100%	Thực thi tốt và đầy đủ yêu cầu đề bài

III/ Phân tích và mô tả thuật toán

1/ Hàm điều chỉnh độ sáng

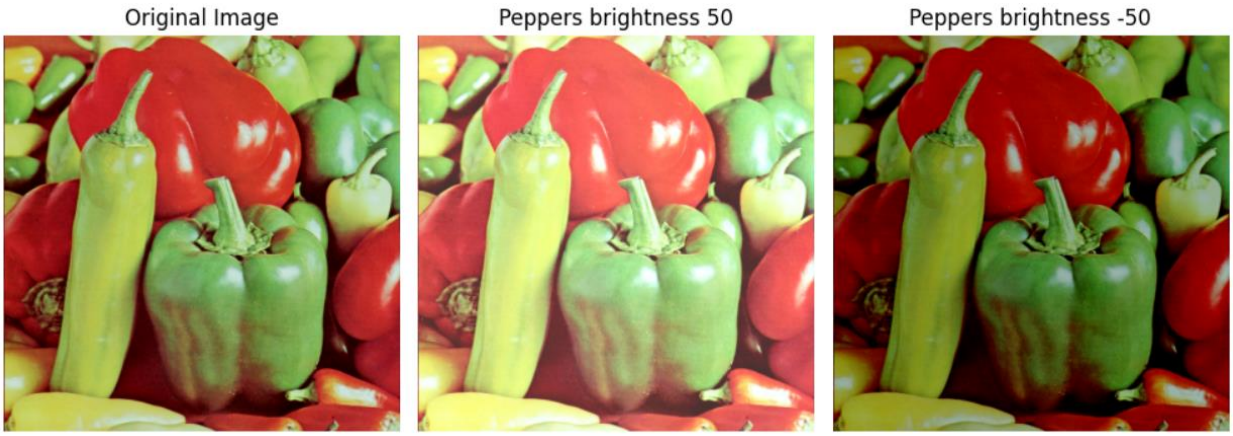
- Ý tưởng: Điều chỉnh tăng/giảm độ sáng ảnh dựa vào độ sáng do người dùng nhập vào

- Tên hàm: adjust_brightness(img : np.array, brightness : int)

- Đầu vào: Một bức ảnh đã chuyển sang dạng array (**img**) và độ sáng cần chỉnh kiểu số nguyên (**brightness**)

- Đầu ra: Một bức ảnh đã được tăng/giảm độ sáng dựa trên độ sáng do người dùng nhập vào (**new_img**)

- Ảnh demo:



- Mô tả thuật toán:

Bước 1. Nhập vào ảnh (**img**) (biểu diễn bằng mảng các giá trị số) và độ sáng (**brightness**). Nếu muốn tăng độ sáng thì nhập vào số dương, giảm độ sáng thì nhập vào số âm

Bước 2. Tạo một mảng 2 chiều chứa giá trị giá trị độ sáng mới. Ở đây với trường hợp mặc định là 50 thì nó sẽ có dạng '**[50]**'

Bước 3. Thực hiện phép cộng mảng độ sáng với mảng ban đầu (bức ảnh) để ra được một mảng mới (bức ảnh sau khi đã tăng độ sáng)

Bước 4. Sau khi cộng, giá trị của mỗi điểm ảnh sẽ được tăng lên 50. Giới hạn giá trị của mỗi điểm ảnh vào trong khoảng [0, 255] để đảm bảo rằng bức ảnh mới có thể được biểu diễn

Bước 5. Chuyển đổi bức ảnh về dạng uint8 (số nguyên không dấu 8-bit) để có thể biểu diễn được như mọi bức ảnh khác

2/ Hàm điều chỉnh độ tương phản

- Ý tưởng: Điều chỉnh tăng/giảm độ tương phản của ảnh dựa vào độ tương phản do người dùng nhập vào

- Tên hàm: adjust_contrast(img : np.array, contrast : int)

- Đầu vào: Một bức ảnh đã chuyển sang dạng array (**img**) và độ tương phản cần chỉnh kiểu số nguyên (**contrast**)

- Đầu ra: Một bức ảnh đã được tăng/giảm độ tương phản dựa trên độ tương phản mà người dùng nhập vào (**new_img**)

- Ảnh demo:

- Mô tả thuật toán:

Bước 1. Nhập vào ảnh (**img**) (biểu diễn bằng mảng các giá trị số) và độ tương phản (**contrast**). Nếu muốn tăng độ tương phản thì nhập vào số dương, giảm độ tương phản thì nhập vào số âm

Bước 2. Giới hạn giá trị độ tương phản trong khoảng [-255, 255] (giới hạn của giá trị tương phản) và chuyển đổi giá trị độ tương phản về kiểu dữ liệu số thực (**float**) để tính toán chính xác hơn

Bước 3. Tính toán hệ số điều chỉnh độ tương phản (**factor**) dựa trên độ tương phản đã nhập

$$factor = \frac{259 * (contrast + 255)}{255 * (259 - contrast)}$$

Bước 4. Áp dụng điều chỉnh tương phản lên ảnh theo thứ tự sau:

+ Chuyển đổi giá trị các điểm ảnh về dạng số thực (**float**)

+ Trừ đi giá trị 128 từ từng điểm ảnh để đưa tâm của phạm vi màu sắc về 0

+ Nhân với hệ số điều chỉnh tương phản (**factor**) và cộng 128 để đưa tâm phạm vi màu sắc về như cũ

+ Giới hạn giá trị các điểm ảnh trong khoảng từ [0, 255]

$$Component = factor * (Component - 128) + 128$$

Bước 5. Chuyển đổi bức ảnh về dạng uint8 (số nguyên không dấu 8-bit) để có thể biểu diễn được như mọi bức ảnh khác

3/ Hàm lật ảnh ngang hoặc dọc

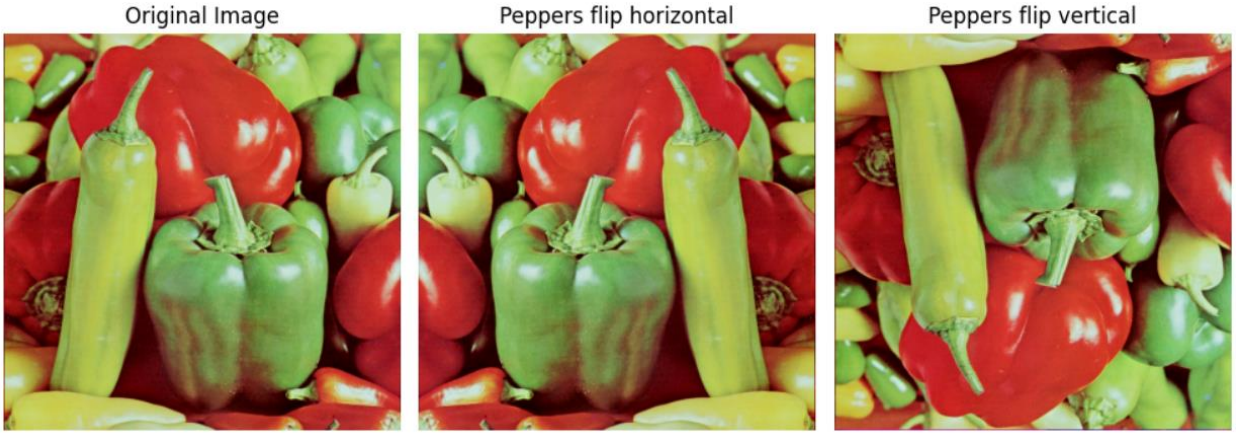
- Ý tưởng: Lật ảnh theo chiều ngang và dọc

- Tên hàm: flip_img(img : np.array)

- Đầu vào: Một bức ảnh đã chuyển sang dạng array (**img**)

- Đầu ra: Hai bức ảnh với một bức được lật ngang (**horizontal_img**) và một bức được lật dọc (**vertical_img**)

- Ảnh demo:



- Mô tả thuật toán:

Bước 1. Nhập vào ảnh (**img**) (biểu diễn bằng mảng các giá trị số)

Bước 2. Sử dụng chức năng cắt lát mảng của Python để xử lý ảnh:

+ **Đối với lật ảnh theo chiều ngang:** thực hiện lát cắt (slicing) theo trục thứ hai (trục ngang) của ma trận ảnh. Giữ nguyên tất cả các hàng (toàn bộ các dòng) của ma trận ảnh, nhưng đảo ngược thứ tự của các cột do đó ảnh sẽ bị lật ngược theo chiều ngang.

+ **Đối với lật ảnh theo chiều dọc:** thực hiện lát cắt (slicing), theo trục đầu tiên (trục dọc) của ma trận ảnh. Giữ nguyên tất cả các cột (toàn bộ các cột) của ma trận ảnh, nhưng đảo ngược thứ tự của các hàng do đó ảnh sẽ bị lật ngược theo chiều dọc.

Bước 3. Chuyển đổi bức ảnh về dạng uint8 (số nguyên không dấu 8-bit) để có thể biểu diễn được như mọi bức ảnh khác

4/ Hàm chuyển đổi về ảnh grayscale/sepia

- **Hàm chuyển ảnh về dạng grayscale**

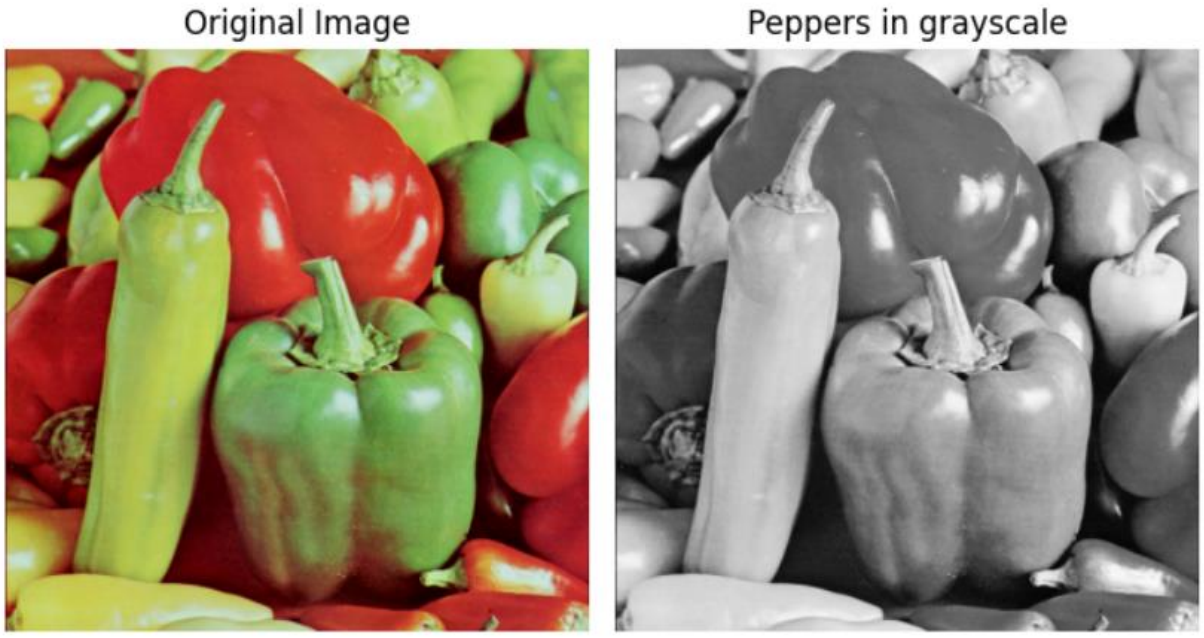
- Ý tưởng: Điều chỉnh màu sắc của ảnh và dạng trắng đen (grayscale)

- Tên hàm: grayscale(img : np.array)

- Đầu vào: Một bức ảnh đã chuyển sang dạng array (**img**)

- Đầu ra: Một bức ảnh đã được chuyển về dạng trắng đen (**grayscale_img**)

- Ảnh demo:



- Mô tả thuật toán:

Bước 1. Nhập vào ảnh (**img**) (biểu diễn bằng mảng các giá trị số)

Bước 2. Mảng trọng số "**weight**"gồm các giá trị [0.299, 0.587, 0.114] theo công thức chuyển đổi về ảnh grayscale

$$grayscale = 0.299 * Red + 0.587 * Green + 0.144 * Blue$$

Bước 3. Trích dẫn 3 kênh màu của ảnh bằng cắt lát để chỉ lấy 3 kênh màu đầu tên (RGB) và bỏ qua các kênh màu còn lại

Bước 4. Thực hiện tích vô hướng (dot product) giữa các giá trị trong kênh màu với trọng số tương ứng của nó. Tương đương với việc nhân mỗi các giá trị màu Red, Green, Blue với trọng số tương ứng và cộng chúng lại với nhau

Bước 5. Kết quả phép nhân là một mảng hai chiều mới (**grayscale_img**) chứa các giá trị ảnh xám mới ứng với bức ảnh ban đầu

Bước 6. Giới hạn các giá trị điểm ảnh vào khoảng [0, 255] và chuyển đổi bức ảnh về dạng uint8 (số nguyên không dấu 8-bit) để có thể biểu diễn được như mọi bức ảnh khác

- **Hàm chuyển ảnh về dạng sepia**

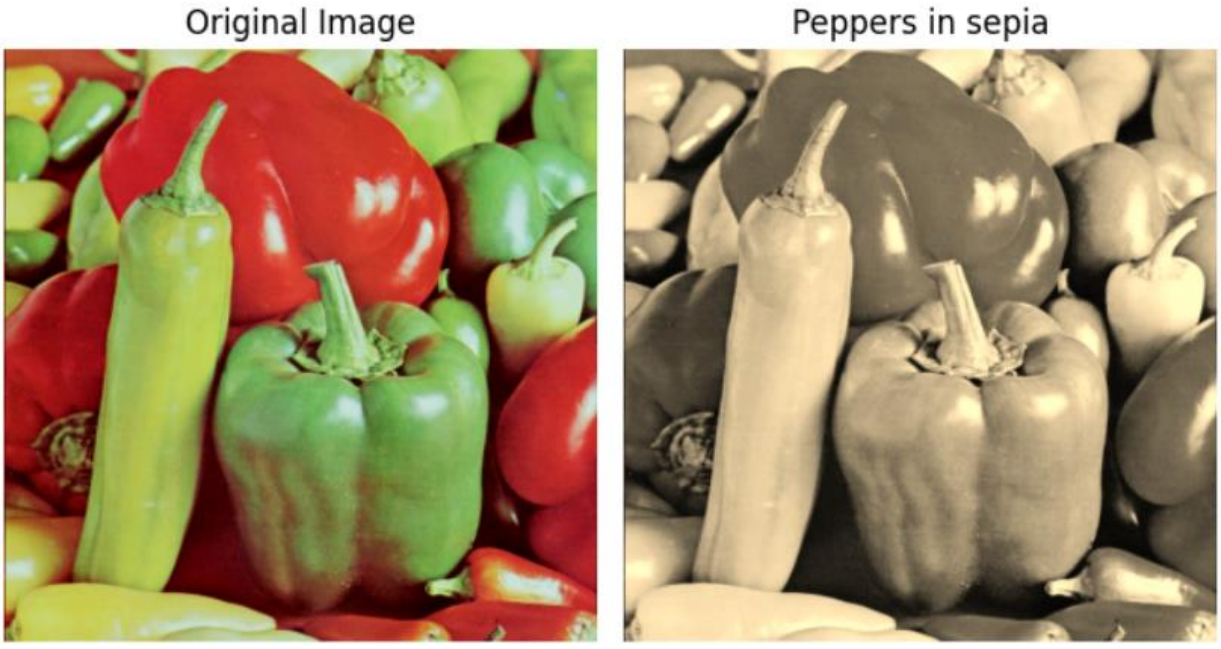
- Ý tưởng: Điều chỉnh màu sắc của ảnh và dạng sepia

- Tên hàm: sepia(img : np.array)

- Đầu vào: Một bức ảnh đã chuyển sang dạng array (**img**)

- Đầu ra: Một bức ảnh đã được chuyển về dạng sepia (**sepia_img**)

- Ảnh demo:



- Mô tả thuật toán:

Bước 1. Nhập vào ảnh (**img**) (biểu diễn bằng mảng các giá trị số)

Bước 2. Mảng 2 chiều (**sepia_matrix**) chứa các giá trị được sử dụng để chuyển đổi ảnh dựa trên công thức chuyển đổi ảnh sang ảnh sepia

$$tR = 0.393 * Red + 0.769 * Green + 0.189 * Blue$$
$$tG = 0.349 * Red + 0.686 * Green + 0.168 * Blue$$
$$tB = 0.272 * Red + 0.534 * Green + 0.131 * Blue$$

Được chuyển thành ma trận 3x3 như sau

$$sepia\ matrix = \begin{bmatrix} 0.393 & 0.769 & 0.189 \\ 0.349 & 0.686 & 0.168 \\ 0.272 & 0.534 & 0.131 \end{bmatrix}$$

Bước 3. Thực hiện tích vô hướng giữa hình ảnh gốc và ma trận chuyển đổi sepia (**sepia_matrix**) hoán vị. Điều này sẽ tạo ra một ma trận mới (**sepia_img**) chứa các giá trị mới tương ứng với mỗi điểm ảnh trong hình ảnh sepia

$$(sepia\ matrix)^T = \begin{bmatrix} 0.393 & 0.349 & 0.272 \\ 0.769 & 0.686 & 0.534 \\ 0.189 & 0.168 & 0.131 \end{bmatrix}$$

Bước 4. Cuối cùng là giới hạn các giá trị điểm ảnh vào khoảng [0, 255] và chuyển đổi bức ảnh về dạng uint8 (số nguyên không dấu 8-bit) để có thể biểu diễn được như mọi bức ảnh khác

5/ Hàm làm mờ và làm nét ảnh

- **Hàm làm mờ ảnh**

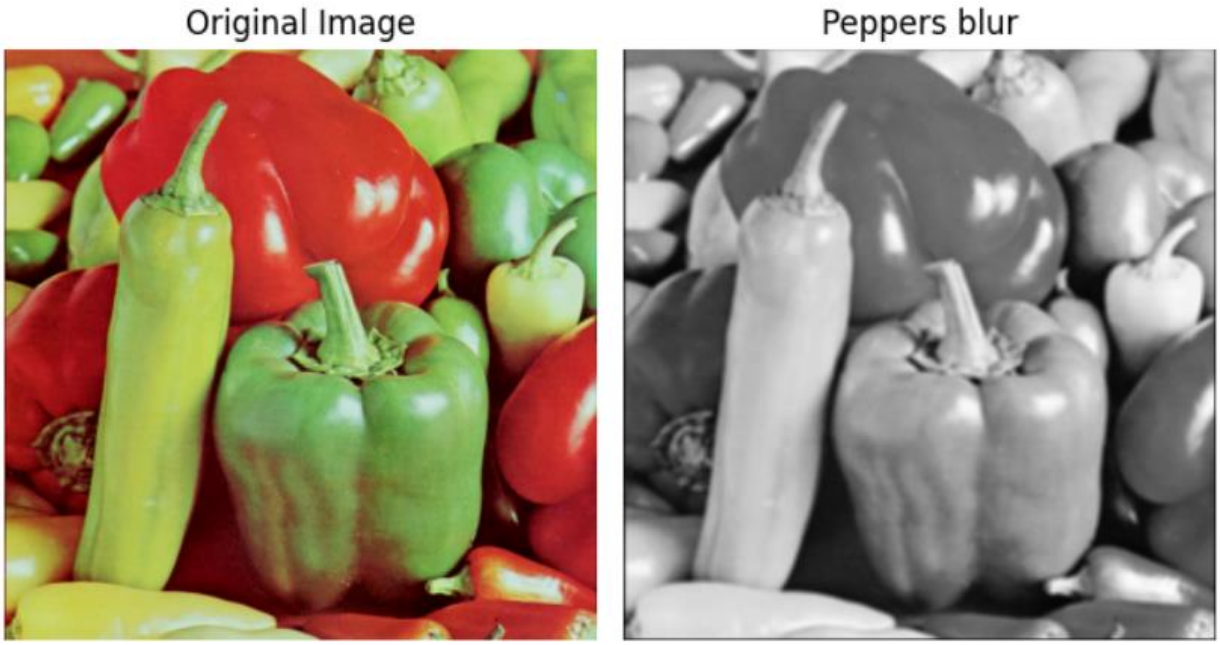
- Ý tưởng: Sử dụng box blur (hoặc trung bình động) để làm mờ ảnh

- Tên hàm: box_blur(img : np.array, r : int)

- Đầu vào: Một bức ảnh đã chuyển sang dạng array (**img**) và bán kính (**r**) của kernel

- Đầu ra: Một bức ảnh đã được làm mờ bằng box blur (**blur_img**) với bán kính nhân được nhập vào (mặc định là r = 2)

- Ảnh demo:



- Mô tả thuật toán:

Bước 1. Nhập vào ảnh (**img**) (biểu diễn bằng mảng các giá trị số) và giá trị bán kính của kernel kiểu số nguyên (**r**)

Bước 2. Tính toán kích thước của kernel dựa vào bán kính đã nhập

$$kernel\ size = 2 * r + 1$$

Bước 3. Khởi tạo kernel bằng một ma trận **kernel_size** x **kernel_size** (với mặc định r = 2, kernel_size = 5 nên kernel sẽ là một ma trận 5x5). Kernel này có tất cả các phần tử bằng nhau và được chuẩn hóa sao cho tổng các phần tử bằng 1 để giữ cho độ sáng tổng thể của ảnh sau khi làm mờ không thay đổi.

$$kernel = \frac{1}{(2 * r + 1)^2} \overbrace{\begin{bmatrix} 1 & \cdots & 1 & \cdots & 1 \\ \vdots & \ddots & \vdots & \iddots & \vdots \\ 1 & \cdots & 1 & \cdots & 1 \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ 1 & \cdots & 1 & \cdots & 1 \end{bmatrix}}^{2 * r + 1} \begin{matrix} \} \text{ r} \\ \\ \\ \} \text{ r} \end{matrix}$$

Bước 4. Tiếp đến, chuyển đổi ảnh về dạng grayscale để làm giảm số lượng tính toán cần thiết trong quá trình làm mờ. Tạo một ảnh mới có kích thước giống với ảnh gốc và giá trị ban đầu của tất cả các điểm ảnh là 0

Bước 5. Sau đó, viền (padding) được thêm vào ảnh ở rìa xung quanh bức ảnh để có thể áp dụng kernel vào biên của bức ảnh gốc mà không bị lỗi tràn số.

$$padding = \left\lfloor \frac{kernel\_size}{2} \right\rfloor$$

Bước 6. Thuật toán bắt đầu duyệt qua từng điểm ảnh trừ biên để không làm thay đổi kích thước ảnh. Với mỗi điểm ảnh, lấy các giá trị trong kernel cùng với giá trị xung quanh điểm ảnh đó (theo kích thước của kernel) nhân chúng lại với nhau và tính tổng. Kết quả là giá trị mới cho điểm ảnh tương ứng trong ảnh đã làm mờ

Bước 7. Cuối cùng là giới hạn các giá trị điểm ảnh vào khoảng [0, 255] và chuyển đổi bức ảnh về dạng uint8 (số nguyên không dấu 8-bit) để có thể biểu diễn được như mọi bức ảnh khác

- **Hàm làm nét ảnh**

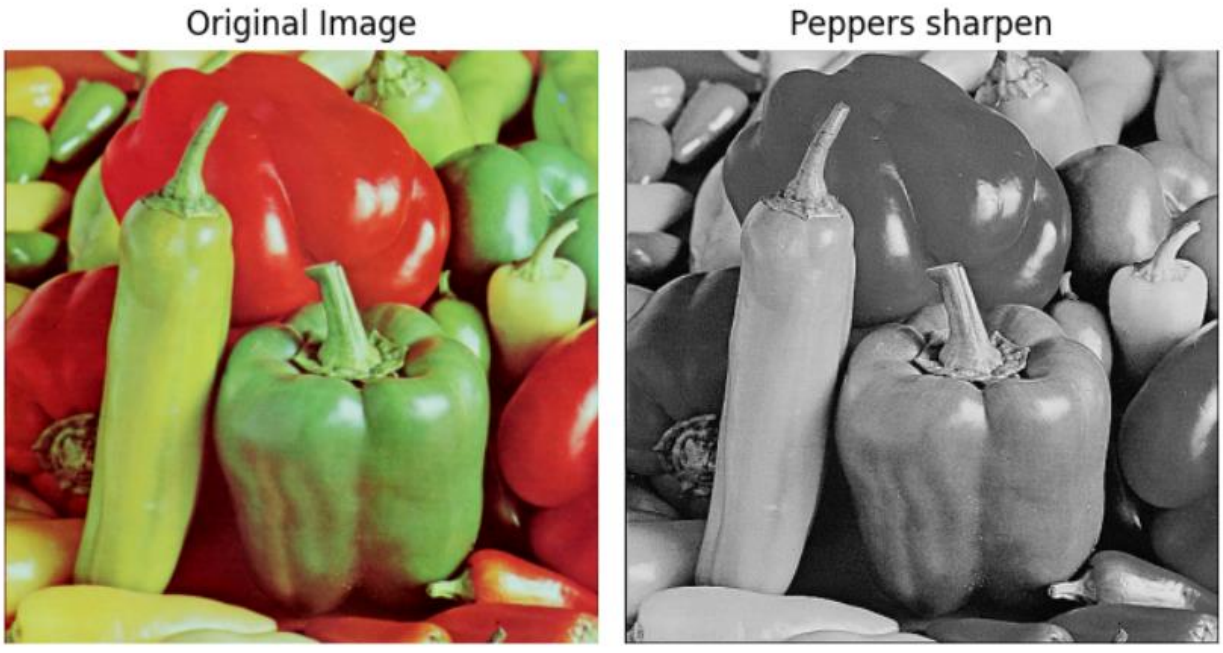
- Ý tưởng: Sử dụng kernel để làm nét ảnh

- Tên hàm: sharpen(img : np.array)

- Đầu vào: Một bức ảnh đã chuyển sang dạng array (**img**)

- Đầu ra: Một bức ảnh đã được làm nét bằng kernel (**sharpen_img**)

- Ảnh demo:



- Mô tả thuật toán:

Bước 1. Nhập vào ảnh (**img**) (biểu diễn bằng mảng các giá trị số)

Bước 2. Khởi tạo kernel làm nét ảnh theo công thức

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

Bước 3. Chuyển đổi ảnh về dạng grayscale để làm giảm số lượng tính toán cần thiết trong quá trình làm nét ảnh. Tạo một ảnh mới có kích thước giống với ảnh gốc và giá trị ban đầu của tất cả các điểm ảnh là 0 (**sharpen_img**)

Bước 4. Sau đó, viền (padding) được thêm vào ảnh ở rìa xung quanh bức ảnh để tránh việc truy cập vượt quá kích thước ảnh

Bước 5. Bắt đầu duyệt từng điểm ảnh bên trong ảnh không tính viền, đối với mỗi điểm ảnh, lấy giá trị xung quanh nó 3x3 nhân với kernel đã khởi tạo. Sau đó tính tổng của các phép toán nhân đó và lưu vào ma trận (**sharpen_img**)

Bước 6. Cuối cùng là giới hạn các giá trị điểm ảnh vào khoảng [0, 255] và chuyển đổi bức ảnh về dạng uint8 (số nguyên không dấu 8-bit) để có thể biểu diễn được như mọi bức ảnh khác

7/ Hàm cắt ảnh từ trung tâm

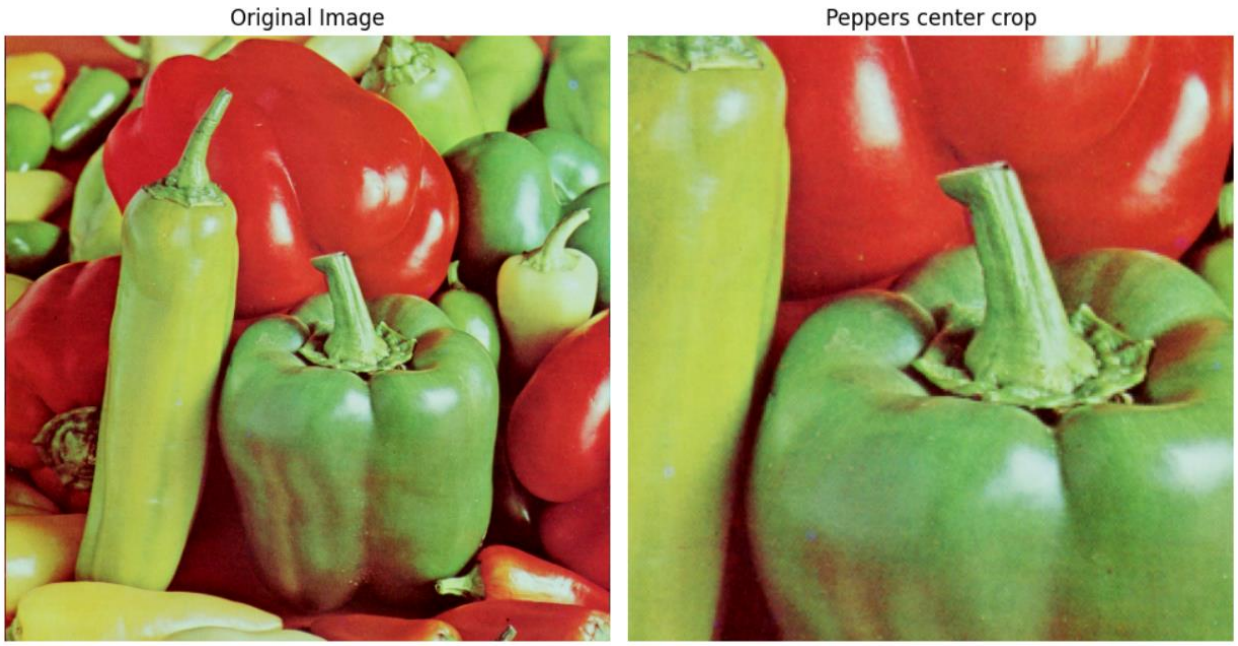
- Ý tưởng: Cắt ảnh theo kích thước tính từ tâm bức ảnh

- Tên hàm: center_crop(img : np.array, size : int)

- Đầu vào: Một bức ảnh đã chuyển sang dạng array (**img**) và kích thước của bức ảnh sau khi được cắt (**size**)

- Đầu ra: Một bức ảnh đã được cắt theo kích thước tính từ tâm bức ảnh (**crop_img**) (kích thước được mặc định là 250x250, yêu cầu cần nhập ảnh có kích thước lớn hơn 250x250)

- Ảnh demo:



- Mô tả thuật toán:

Bước 1. Nhập vào ảnh (**img**) (biểu diễn bằng mảng các giá trị số) và kích thước ảnh sau khi cắt (**size**) (Mặc định size là 250 pixels)

Bước 2. Lấy kích thước của bức ảnh (**width** và **height**) và kiểm tra với kích thước cần cắt. Nếu kích thước cần cắt lớn hơn kích thước ảnh hiện tại thì sẽ báo lỗi và dừng chương trình

Bước 3. Tính toán các vị trí cắt trên (**top**), dưới (**bottom**), trái (**left**) và phải (**right**):

$$top = \left\lfloor \frac{(height - size)}{2} \right\rfloor \ \& \ bottom = top + size$$

$$left = \left\lfloor \frac{(width - size)}{2} \right\rfloor \ \& \ right = left + size$$

Bước 4. Bắt đầu cắt ảnh sử dụng cắt lát mảng theo các thông số top, bottom, left, right đã tính. Kết quả là một mảng mới (**crop_img**)

Bước 5. Chuyển đổi bức ảnh về dạng uint8 (số nguyên không dấu 8-bit) để có thể biểu diễn được như mọi bức ảnh khác

8/ Hàm cắt ảnh theo khung tròn

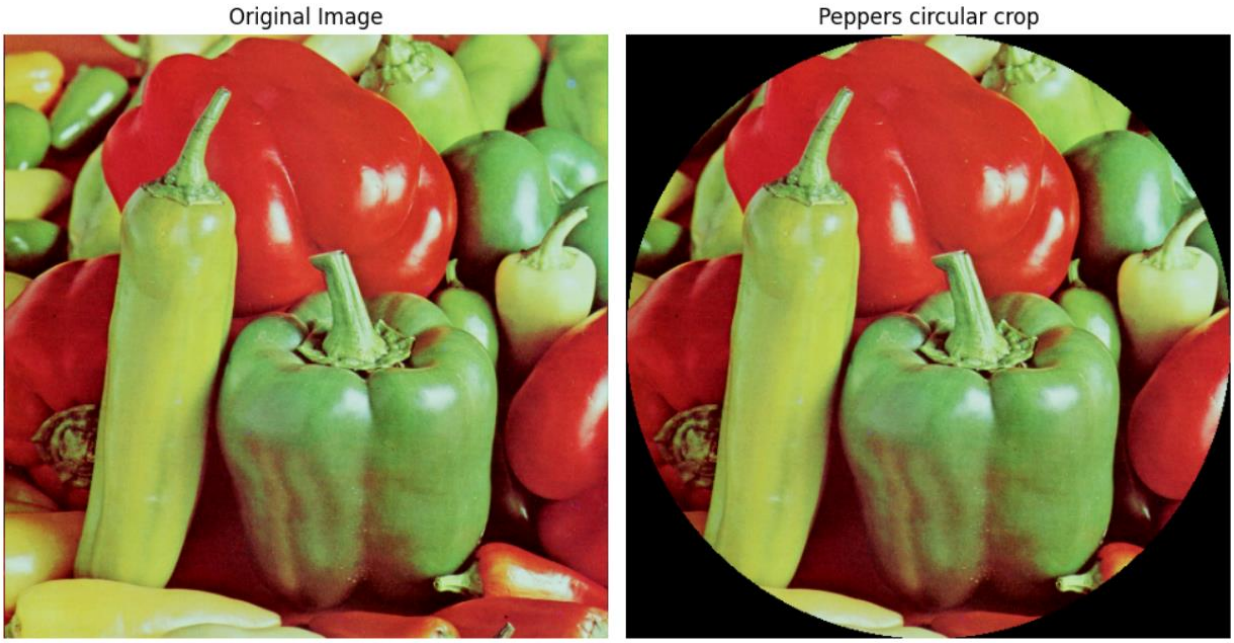
- Ý tưởng: Cắt bức ảnh bằng một khung hình tròn, giữ lại ảnh trong khung tròn và bôi đen toàn bộ ở ngoài khung tròn. Sử dụng boolean mask để thực hiện

- Tên hàm: circular_crop(img : np.array)

- Đầu vào: Một bức ảnh đã chuyển sang dạng array (**img**)

- Đầu ra: Một bức ảnh đã được cắt theo khung hình tròn với phần ảnh trong hình tròn được giữ nguyên và bên ngoài hình tròn được bôi đen (**circular_img**)

- Ảnh demo:



- Mô tả thuật toán:

Bước 1. Nhập vào ảnh (**img**) (biểu diễn bằng mảng các giá trị số)

Bước 2. Tính tọa độ tâm điểm của ảnh (**center**) dựa vào kích thước của ảnh và tính bán kính của hình tròn bằng cách lấy lấy giá trị nhỏ hơn giữa chiều dài và rộng của bức ảnh và chia cho 2

$$radius = \left\lfloor \frac{\min(width, height)}{2} \right\rfloor$$

Bước 3. Tạo hai mảng 2D (**y**) và (**x**) đại diện cho lưới các điểm ảnh trong hình ảnh

Bước 4. Tính khoảng cách Euclidean từ mỗi điểm ảnh đến tâm của ảnh. Dựa vào khoảng cách này, tạo một mặt nạ (**circular_mask**) với giá trị True cho các điểm ảnh nằm trong vùng hình tròn và False cho các điểm ảnh nằm bên ngoài vùng hình tròn

$$Euclidean\ distance = \sqrt{(x - x_{center})^2 + (y - y_{center})^2}$$

Bước 5. Áp dụng mặt nạ để đặt giá trị của các điểm ảnh nằm bên ngoài vùng hình tròn thành 0 (biến thành màu đen) và giữ nguyên ảnh với các vùng nằm ở trong hình tròn

Bước 6. Trả về hình ảnh đã được cắt dưới dạng np.array (**circular_img**)

9/ Hàm cắt ảnh theo khung 2 hình elip chéo nhau

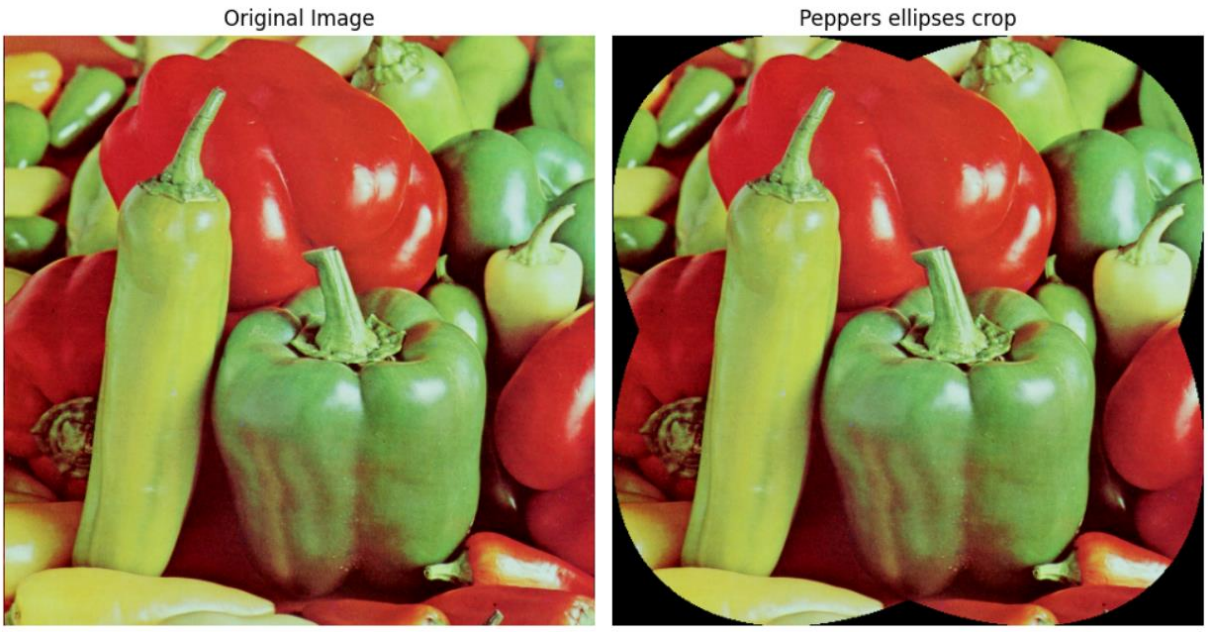
- Ý tưởng: Cắt bức ảnh bằng một khung 2 hình elip đặt chéo nhau, giữ lại ảnh trong khung tròn và bôi đen toàn bộ hình ở ngoài khung. Sử dụng boolean mask để thực hiện

- Tên hàm: ellipses_crop(img : np.array)

- Đầu vào: Một bức ảnh đã chuyển sang dạng array (**img**)

- Đầu ra: : Một bức ảnh đã được cắt theo khung 2 hình elip với phần ảnh trong hình tròn được giữ nguyên và bên ngoài hình tròn được bôi đen (**ellipses_img**)

- Ảnh demo:



- Mô tả thuật toán:

Bước 1. Nhập vào ảnh (**img**) (biểu diễn bằng mảng các giá trị số)

Bước 2. Lấy chiều dài và rộng của ảnh để tính tọa độ tâm điểm của ảnh (**center**), tính toán bán trục chính (**s_major**), bán trục phụ (**s_minor**) của hình elip và bán kính hình (**radius**)

$$s\ major = \ width$$
$$s\ minor = \frac{width}{(\sqrt{2}+1)}$$
$$radius = \frac{\min(width, height)}{2}$$

Bước 3. Tạo hai mảng 2D (**y**) và (**x**) biểu diễn lưới pixel của ảnh

Bước 4. Tính toán khoảng cách từ mỗi pixel đến tâm của ảnh và lưu vào hai mảng **dist1** và **dist2**

$$dist1 = (x - y_{center}) + (y - x_{center})$$
$$dist2 = (x - y_{center}) - (y - x_{center})$$

Bước 5. Tính toán khoảng cách từ mỗi pixel đến tâm của ảnh (**mask1** và **mask2**) và tạo một mặt nạ boolean với giá trị True cho các pixel nằm trong vùng hình elip

$$mask1 = \frac{-dist2^2}{s\ major * \sqrt{2}} + \frac{dist1^2}{s\ minor * \sqrt{2}}$$
$$mask2 = \frac{dist1^2}{s\ major * \sqrt{2}} + \frac{-dist2^2}{s\ minor * \sqrt{2}}$$

Bước 6. Kết hợp **mask1** và **mask2** bằng phép OR để tại thành mặt nạ với 2 hình elip đặt chéo nhau

Bước 7. Áp dụng mặt nạ lên ảnh và đặt tất cả các giá trị bên ngoài mặt nạn thành 0 và giữ nguyên ảnh ở bên trong mặt nạ

Bước 8. Trả về hình ảnh đã được cắt dưới dạng np.array (**ellipses_img**)

TÀI LIỆU THAM KHẢO

Applied-Mathematics-and-Statistics by NgocTien0110 - Github

Convert an Image to Grayscale in Python - DelftStack

How to convert a color image into sepia image - DyClassroom

Grayscale and Color in Images - PTC

Image Processing 101 - Dynamsoft

Kernel (Image processing) - Wikipedia

Box blur - Wikipedia