

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN



Báo cáo đồ án

Lab: Project 2

Môn học: Toán ứng dụng và thống kê

Người thực hiện:

22127390 - Nguyễn Văn Lê Bá Thành -
22CLC08

Giảng viên:

Cô Phan Thị Phương Uyên
Thầy Nguyễn Ngọc Toàn
Thầy Nguyễn Văn Quang Huy
Thầy Vũ Quốc Hoàng

Ngày 30 tháng 7 năm 2024

Mục lục

1 Lời cảm ơn	2
2 Giới thiệu	3
3 Các hàm phụ trợ và hàm main	4
3.1 Các hàm phụ trợ	4
3.1.1 Hàm read_img	4
3.1.2 Hàm show_img	4
3.1.3 Hàm save_img	4
3.2 Hàm main	5
4 Ý tưởng thực hiện và cài đặt	6
4.1 Điều chỉnh độ sáng của ảnh	6
4.2 Điều chỉnh độ tương phản của ảnh	7
4.3 Lật ảnh	8
4.3.1 Lật ảnh dọc	8
4.3.2 Lật ảnh ngang	9
4.4 Chuyển đổi từ ảnh màu sang các loại ảnh khác	9
4.4.1 Chuyển từ ảnh màu sang ảnh xám	9
4.4.2 Chuyển từ ảnh màu sang ảnh sepia	11
4.5 Làm mờ/sắc nét ảnh	12
4.5.1 Làm mờ ảnh	14
4.5.2 Làm sắc nét ảnh	16
4.6 Cắt ảnh theo kích thước (cắt ở trung tâm)	17
4.7 Cắt ảnh theo khung tròn	18
4.8 Cắt ảnh theo khung 2 elip chéo nhau	19
4.9 Phóng to/Thu nhỏ 2x	22
4.9.1 Phóng to ảnh 2x	22
4.9.2 Thu nhỏ ảnh 2x	23
5 Mức độ hoàn thành	24

1 Lời cảm ơn

Nếu không có sự giúp đỡ và hướng dẫn của nhiều người và từ nhiều nguồn khác nhau, đồ án này sẽ không thể hoàn thành.

Dầu tiên, em muốn cảm ơn các nhà phát triển và cộng tác viên của thư viện Numpy, Pillow và Matplotlib. Tài liệu có sẵn trên trang web của họ rất hữu ích trong việc làm quen và áp dụng để cài đặt các hàm cần thiết cho đồ án này.

Em cũng muốn cảm ơn các thầy/cô hướng dẫn vì những lời khuyên hữu ích và những bài giảng với các kiến thức cần thiết trong suốt môn học này. Sự hiểu biết của em về dự án này đã được nâng cao hơn nhiều nhờ những lời giải thích rõ ràng và sự cởi mở với các câu hỏi.

Em đặc biệt muốn cảm ơn ChatGPT một mô hình ngôn ngữ lớn, vì đã giúp thu thập thông tin và tóm tắt thông tin về việc thực hiện các phép biến đổi trên hình ảnh và cách hoạt động của các chức năng liên quan.

2 Giới thiệu

Xử lý ảnh là một phương pháp dùng để thực hiện các thao tác trên hình ảnh nhằm cải thiện hoặc trích xuất các thông tin hữu ích. Đây là một loại xử lý tín hiệu mà đầu vào là một hình ảnh, chẳng hạn như một bức ảnh hoặc một khung hình video, và đầu ra có thể là một hình ảnh hoặc một tập hợp các đặc tính hoặc thông số liên quan đến hình ảnh đó. Lĩnh vực này là một nền tảng quan trọng của công nghệ kỹ thuật số hiện đại, ảnh hưởng đến nhiều ứng dụng như nhiếp ảnh, hình ảnh y tế, hình ảnh vệ tinh, và thị giác máy tính.

Trong đồ án này, em được yêu cầu thực hiện các chức năng xử lý ảnh như: điều chỉnh độ sáng, độ tương phản, lật ảnh theo chiều ngang hoặc dọc, chuyển đổi ảnh màu thành ảnh xám, chuyển đổi ảnh màu thành ảnh sepia, làm mờ ảnh, làm sắc nét ảnh, áp dụng khung tròn và khung hai elip cheo nhau cho ảnh.

Các phần sau được tổ chức như sau, Phần 3 đưa ra chi tiết về các hàm phụ trợ và hàm main. Ý tưởng và mô tả về cài đặt của các chức năng sẽ được đưa ra trong Phần 4. Chi tiết đánh giá về các ảnh kết quả sẽ được cung cấp ở Phần 5.

3 Các hàm phụ trợ và hàm main

3.1 Các hàm phụ trợ

3.1.1 Hàm read_img

- **Mô tả hàm:** read_img được sử dụng để đọc hình ảnh từ đường dẫn cho trước.
- **Tham số đầu vào:** nhận một biến string chứa đường dẫn đến hình ảnh cần nén
- **Kết quả đầu ra:** một object có kiểu dữ liệu "PIL.Image.Image"
- **Các bước thực hiện:** sử dụng hàm open trong thư viện Pillow để đọc hình ảnh từ đường dẫn và gán vào biến raw_image [1].

3.1.2 Hàm show_img

- **Mô tả hàm:** show_img giúp đưa hình ảnh ra ngoài màn hình.
- **Tham số đầu vào:** nhận biến img là một object có kiểu dữ liệu "PIL.Image.Image" hoặc "ArrayLike". Trong đó kiểu dữ liệu "ArrayLike" là các object có thể được chuyển đổi thành một mảng [2].
- **Kết quả đầu ra:** hình ảnh được thể hiện trên màn hình
- **Các bước thực hiện:** dùng method show của thư viện matplotlib sẽ xử lý ảnh được truyền vào [3] sau đó gọi plt.show() để hiển thị các hình trước đó.

3.1.3 Hàm save_img

- **Mô tả hàm:** save_img là hàm giúp lưu hình ảnh với đường dẫn cho sẵn.
- **Tham số đầu vào:** hàm nhận hai biến số đầu vào img là ma trận điểm ảnh ndarray và img_path là đường dẫn đến tập tin đích.
- **Kết quả đầu ra:** kết quả trả ra của hàm này là một file hình ảnh được lưu ở đường dẫn đích.
- **Các bước thực hiện:** ta sử dụng hàm save trong thư viện numpy và truyền tham số img_path vào [1].

*Lưu ý: vì hàm save_img được sử dụng sau khi quá trình xử lý ảnh đã hoàn tất, và đã được chuyển đổi thành object có kiểu dữ liệu "PIL.Image.Image"

3.2 Hàm main

- **Mô tả hàm:** main là xử lý input và gọi các hàm để thực thi các chức năng.
- **Các bước thực hiện:** ta sử dụng hàm save trong thư viện numpy và truyền tham số img_path vào.

B1: Cho phép người dùng nhập vào tên của ảnh và thuật toán muốn thực hiện

B2: Tiếp theo ta tiền hành phân tách img_path để lấy tên ảnh và định dạng cả ảnh bằng các kỹ thuật indexing và hàm split sau đó lưu lại để sử dụng sau.

B3: Ta sử dụng hàm read_img và truyền vào img_path để lấy được ma trận điểm ảnh gốc.

B4: Do object trả về từ hàm read_img có kiểu "PIL.Image.Image" nên ta chuyển đổi object này thành ndarray bằng lệnh "pic = np.array(raw_img).astype(np.uint8)" với hàm astype [4] dùng để ép kiểu ma trận đầu ra về kiểu dữ liệu uint8 để thuận tiện cho việc thực hiện các chức năng xử lý ảnh.

Riêng đối với ảnh xám do tính chất của mình ảnh xám chỉ có một giá trị màu nên ta cần chuyển đổi giá trị này thành 3 kênh màu như ảnh màu với giá trị ở ba kênh màu là giống nhau sử dụng np.stack. Để biết ảnh xám đang có dạng nào ta sử dụng thuộc tính shape để kiểm tra nếu shape trả về 2 có nghĩa ma trận điểm ảnh xám này chỉ có một giá trị và ta sẽ dùng np.stack [5] để chuyển đổi ma trận này thành ma trận điểm ảnh có 3 kênh màu.

B5: Sau đó dựa vào lựa chọn thuật toán đã nhập từ trước, ta thực hiện thuật toán tương ứng và kết quả của thuật toán sẽ là một ma trận có kiểu dữ liệu "numpy.ndarray".

B6: Sử dụng method fromarray của thư viện Image để chuyển đổi ma trận kết quả thành object có kiểu dữ liệu "PIL.Image.Image".

B7: Xây dựng tên file mới bao gồm tên file_<tên chức năng>.<định dạng của ảnh gốc>.

B8: In hình ảnh được trả ra với hàm show_img.

B9: Lưu ảnh được trả ra với hàm save_img.

4 Ý tưởng thực hiện và cài đặt

Trong phần này mỗi chức năng xử lý ảnh sẽ được trình bày từ phần ý tưởng bao gồm các phép toán, các phép biến đổi cần thiết để đạt được mục đích của chức năng. Tiếp đến phần cài đặt chi tiết sẽ mô tả cách mà chức năng được cài đặt bằng ngôn ngữ Python trên nền tảng Jupyter Notebook.

4.1 Điều chỉnh độ sáng của ảnh

- **Ý tưởng thực hiện:**

Để tăng hoặc giảm sáng cho một ảnh, ta có thể tăng hoặc giảm giá trị các kênh màu của các điểm ảnh với cùng một giá trị như nhau. Giá trị tăng càng lớn thì ảnh càng sáng và ngược lại.

- **Cài đặt:**

- **Tên hàm:** change_brightness

- **Tham số đầu vào:** Hàm nhận hai đối số đó là img (ma trận điểm ảnh gốc) và brightness (giá trị để tăng hoặc giảm sáng).

- **Kết quả đầu ra:** Ma trận điểm ảnh sau khi được tăng/giảm với giá trị brightness.

- **Chi tiết thực hiện:**

1. Sử dụng hàm astype được cung cấp bởi thư viện numpy để chuyển đổi ma trận điểm ảnh gốc về kiểu dữ liệu có độ chính xác cao hơn ở đây chúng ta sẽ sử dụng kiểu dữ liệu float32 bằng lệnh "img = img.astype(np.float32)", do ở kiểu dữ liệu uint8 chỉ chứa được 256 giá trị nên sẽ có khả năng sẽ xảy ra tràn số dẫn đến ảnh kết quả có hiệu ứng dị thường.

2. Xây dựng ảnh kết quả bằng cách cộng hai ma trận ban đầu với giá trị mà người dùng nhập vào bằng cách sử dụng lệnh "result = np.clip(img + brightness, 0, 255)". Với "img + brightness" sẽ cộng giá trị trong brightness vào trong 3 kênh màu của mỗi điểm ảnh nhờ phép tính **element-wise** và hàm np.clip sẽ đảm nhận việc giới hạn các kênh màu không vượt quá khoảng [0, 255].

Element-wise: là thực hiện phép tính trên từng phần tử của các vector hoặc ma trận. Ví dụ, nếu bạn có hai vector a và b cùng kích thước, việc tính toán a + b element-wise nghĩa là thực hiện phép cộng giữa từng cặp phần tử tương ứng trong a và b, và trả về một vector mới chứa tổng của từng cặp phần tử đó.

3. Trả về ma trận điểm ảnh có các kênh màu đã tăng/giảm.

4.2 Điều chỉnh độ tương phản của ảnh

- **Ý tưởng thực hiện:**

Độ tương phản của ảnh là mức độ khác biệt về màu sắc giữa các vùng sáng và tối trong ảnh. Nó là độ lớn khoảng cách giữa các điểm ảnh sáng nhất và tối nhất của ảnh. Để tăng độ tương phản thì ta cần phải tăng khoảng cách giữa các điểm ảnh sáng và tối.

Trong đồ án này, để tương đồng với chức năng điều chỉnh độ sáng của ảnh đó là người dùng có thể chọn được giá trị tăng giảm của độ tương phản trong khoảng [-255,255], và để làm được như thế ta phải sử dụng đến công thức như sau [6]:

$$F = \frac{259(C + 255)}{255(259 - C)} \quad (1)$$

Trong đó C là giá trị contrast nằm trong khoảng [-255,255]

Công thức 1 được sử dụng để tìm ra hệ số điều chỉnh độ tương phản F, hệ số này có giá trị trong khoảng [0, 129.5][6]. Sau đó, hệ số điều chỉnh này sẽ được sử dụng trong công thức sau để tìm ra giá trị mới của một kênh màu [6]:

$$R' = F(R - 128) + 128 \quad (2)$$

- **Cài đặt:**

- **Tên hàm:** change_contrast
- **Tham số đầu vào:** Hàm nhận hai đối số đó là img (ma trận điểm ảnh gốc) và contrast (giá trị để tăng hoặc giảm độ tương phản).
- **Kết quả đầu ra:** Ma trận điểm ảnh sau khi được tăng giảm với giá trị contrast.
- **Chi tiết thực hiện:**

1. Sử dụng hàm astype được cung cấp bởi thư viện numpy để chuyển đổi ma trận điểm ảnh gốc về kiểu dữ liệu có độ chính xác cao hơn ở đây chúng ta sẽ sử dụng kiểu dữ liệu float32 bằng lệnh "img = img.astype(np.float32)", do ở kiểu dữ liệu uint8 chỉ chứa được 256 giá trị nên sẽ có khả năng sẽ xảy ra tràn số dẫn đến ảnh kết quả có hiệu ứng dị thường.
2. Cài đặt công thức 1 và tính giá trị hệ số điều chỉnh F với giá trị của đối số contrast bằng lệnh "factor = (259 * (contrast + 255)) / (255 * (259 - contrast))".
3. Xây dựng ma trận điểm ảnh kết quả bằng cách sử dụng công thức 2 với R là ma trận điểm ảnh bằng lệnh "result = np.clip(factor * (img - 128) + 128, 0, 255)" và dùng hàm np.clip sẽ đảm nhận việc giới hạn các kênh màu không vượt quá khoảng [0, 255].
4. Trả về ma trận điểm ảnh kết quả đã thay đổi độ tương phản.

4.3 Lật ảnh

Như đã biết một ảnh có bản chất là ma trận các điểm ảnh, do đó để lật ngang hoặc dọc một ảnh, ta có thể đảo cột hoặc dòng của ma trận điểm ảnh. Sau đây là một ma trận điểm ảnh thường thấy:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Trong đó a_{ij} là một điểm ảnh có ba kênh màu RGB.

4.3.1 Lật ảnh dọc

- **Ý tưởng thực hiện:**

Đối với lật ảnh dọc ta cần đổi vị trí các giá trị ở hàng thứ n với hàng đầu tiên để tạo ra một ma trận mới với hàng đầu tiên của ma trận mới là các giá trị ở hàng thứ n của ma trận cũ.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \rightarrow \begin{bmatrix} a_{m1} & a_{m2} & \dots & a_{mn} \\ \vdots & \vdots & \ddots & \vdots \\ a_{11} & a_{12} & \dots & a_{1n} \end{bmatrix}$$

- **Cài đặt:**

- **Tên hàm:** flip_img
- **Tham số đầu vào:** Hàm nhận hai đối số đó là img (ma trận điểm ảnh gốc) và direction (hướng lật của ảnh trong trường hợp này direction sẽ là vertical).
- **Kết quả đầu ra:** Ma trận điểm ảnh sau khi được lật dọc.
- **Chi tiết thực hiện:**

1. Kiểm tra đối số direction để rẽ nhánh. Nếu direction là "vertical" thì thực hiện lật ảnh dọc, nếu là "horizontal" thì thực hiện lật ảnh ngang.
2. Dùng lệnh "flipped_arr = img[::-1, :]" để tạo ra một ma trận mới dựa trên ảnh đầu vào. Trong python cú pháp để thực hiện slicing là `list[<start>:<stop>:<step>]` [7] do đó khi ta sử dụng "`[::-1]`" ta sẽ lấy toàn bộ phần tử của list và với step có giá trị là -1 thì ta có thể đảo ngược list đó. Vậy "`img[::-1, :]`" có thể được hiểu là đảo ngược các phần tử theo hàng và giữ nguyên cột và các kênh màu của ảnh đầu vào.
3. Trả về ma trận điểm ảnh kết quả đã được lật theo chiều dọc.

4.3.2 Lật ảnh ngang

- **Ý tưởng thực hiện:**

Đối với lật ảnh dọc ta cần đổi vị trí các giá trị ở cột thứ n với cột đầu tiên để tạo ra một ma trận mới với cột đầu tiên của ma trận mới là các giá trị ở cột thứ n của ma trận cũ.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \rightarrow \begin{bmatrix} a_{1n} & \dots & a_{12} & a_{11} \\ \vdots & \ddots & \vdots & \vdots \\ a_{mn} & \dots & a_{m2} & a_{m1} \end{bmatrix}$$

- **Cài đặt:**

- **Tên hàm:** flip_img
- **Tham số đầu vào:** Hàm nhận hai đối số đó là img (ma trận điểm ảnh gốc) và direction (hướng lật của ảnh trong trường hợp này direction sẽ là horizontal).
- **Kết quả đầu ra:** Ma trận điểm ảnh sau khi được lật ngang.
- **Chi tiết thực hiện:**
 1. Kiểm tra đối số direction để rẽ nhánh. Nếu direction là "vertical" thì thực hiện lật ảnh dọc, nếu là "horizontal" thì thực hiện lật ảnh ngang.
 2. Dùng lệnh "flipped_arr = img[:, ::-1]" để tạo ra một ma trận mới dựa trên ảnh đầu vào. Trong python cú pháp để thực hiện slicing là `list[<start>:<stop>:<step>]` [7] do đó khi ta sử dụng "`::-1`" ta sẽ lấy toàn bộ phần tử của list và với step có giá trị là `-1` thì ta có thể đảo ngược list đó. Vậy "`img[:, ::-1]`" có thể được hiểu như sau, đảo ngược các phần tử theo cột và giữ nguyên hàng và các kênh màu của ảnh đầu vào.
 3. Trả về ma trận điểm ảnh kết quả đã được lật theo chiều ngang.

4.4 Chuyển đổi từ ảnh màu sang các loại ảnh khác

4.4.1 Chuyển từ ảnh màu sang ảnh xám

- **Ý tưởng thực hiện:**

Như đã biết một ảnh màu có bản chất là ma trận các điểm ảnh có 3 kênh màu RGB, tuy nhiên đối với ảnh xám thì các điểm ảnh chỉ có một kênh màu có giá trị nằm trong khoảng [0, 255].

Vậy để có thể chuyển đổi từ ảnh màu sang ảnh xám ta cần phải gộp giá trị của 3 kênh màu thành 1 giá trị duy nhất. Theo [8], ta có thể sử dụng trọng số (đóng góp) của từng màu được quy định sẵn là x,y,z. Dựa vào đó ta có thể xác định công thức tính giá trị của một điểm ảnh trong ảnh xám là:

$$GrayColor = xRED + yGREEN + zBLUE \text{ với } (x + y + z = 1) \quad (3)$$

Cũng theo [8], ta có thể tạo một vector trọng số để chứa trọng số x,y,z cho từng kênh màu:

$$Weight = [0.299 \quad 0.587 \quad 0.114]$$

Để tìm được giá trị của một điểm ảnh xám từ một điểm ảnh có 3 kênh màu RGB, áp dụng công thức 3 ta tính tích vô hướng giữa vector điểm ảnh RGB với vector trọng số.

$$GrayColor = [R \quad G \quad B] \times [0.299 \quad 0.587 \quad 0.114] = [0.299 * R + 0.587 * G + 0.114 * B]$$

Thực hiện tính toán này trên toàn bộ ma trận điểm ảnh của ảnh màu ta sẽ thu được một ma trận mới chứa các điểm ảnh xám.

- Cài đặt:

- **Tên hàm:** convert_to_grayscale
- **Tham số đầu vào:** Hàm nhận một đối số đó là img (ma trận điểm ảnh gốc).
- **Kết quả đầu ra:** Ma trận điểm ảnh xám được chuyển đổi từ ma trận điểm ảnh RGB.
- **Chi tiết thực hiện:**
 1. Tạo vector trọng số bằng thư viện numpy sử dụng hàm np.array với các trọng số lần lượt là 0.299, 0.587, 0.114 như đã nêu ra ở phần ý tưởng thực hiện.
 2. Sử dụng hàm matmul cung cấp bởi thư viện numpy [9] để thực hiện phép nhân ma trận img với ma trận weight. Ta có ma trận đầu vào có kích thước $m \times n \times 3$ vậy khi nhân với vector trọng số có kích thước 3×1 thì ma trận kết quả sẽ có kích thước $m \times n \times 1$.
 3. Để đồng bộ với hàm show_img sử dụng hàm imshow của thư viện matplotlib và giới hạn các hàm hỗ trợ có thể dùng nên trong đồ án này chỉ hỗ trợ xử lý ảnh có ba kênh màu.
 4. Vì lý do trên ma trận kết quả sau khi nhân sẽ được chuyển đổi thành ảnh có ba kênh màu thay vì chỉ có một kênh màu như ảnh xám bình thường. Lệnh được sử dụng đó là "stacked_greyscale_img = np.stack((result,)*3, axis=-1)" với "(result,)*3" tạo ra một tuple chứa 3 ma trận result và hàm np.stack sử dụng tuple này để chồng 3 ma trận result theo trục mới với đối số "axis=-1" và ma trận ảnh xám sẽ có kích thước như ảnh màu RGB là $m \times n \times 3$.
 5. Trả về ma trận điểm ảnh xám với ba kênh màu có cùng giá trị.

4.4.2 Chuyển từ ảnh màu sang ảnh sepia

- **Ý tưởng thực hiện:**

Tương tự như chuyển đổi ảnh màu thành ảnh xám, một điểm ảnh sepia sẽ có cách tính tương tự tuy nhiên ảnh sepia sẽ có 3 kênh màu giống như ảnh RGB và giá trị của một kênh màu được tính dựa trên trọng số của kênh đó.

Điều khiến ảnh sepia khác biệt so với ảnh xám đó là ảnh xám chỉ có một giá trị màu và có một vector trọng số được áp dụng để tính giá trị đó. Còn đối với sepia, mỗi kênh màu được tính với trọng số khác nhau như sau [10]:

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

Trong đó, các giá trị ở hàng đầu tiên là trọng số cho kênh màu RED, các giá trị ở hàng thứ hai là trọng số cho kênh màu GREEN và các giá trị ở hàng cuối cùng là trọng số cho kênh màu BLUE. Từ ma trận trọng số này, ta có được công thức tính giá trị các kênh màu cho một điểm ảnh sepia như sau:

$$New_red = 0.393 * R + 0.769 * G + 0.189 * B \quad (4)$$

$$New_green = 0.349 * R + 0.686 * G + 0.168 * B \quad (5)$$

$$New_blue = 0.272 * R + 0.534 * G + 0.131 * B \quad (6)$$

Thực hiện tính toán này trên toàn bộ ma trận điểm ảnh của ảnh màu ta sẽ thu được một ma trận mới chứa các điểm ảnh sepia.

- **Cài đặt:**

- **Tên hàm:** convert_to_sepia
- **Tham số đầu vào:** Hàm nhận một đối số đó là img (ma trận điểm ảnh gốc).
- **Kết quả đầu ra:** Ma trận điểm ảnh sepia được chuyển đổi từ ảnh màu RGB.
- **Chi tiết thực hiện:**

1. Tạo ma trận trọng số bằng thư viện numpy sử dụng hàm np.array với các trọng số như đã nêu ra ở phần ý tưởng thực hiện.
2. Sử dụng hàm matmul cung cấp bởi thư viện numpy [9] để thực hiện phép nhân ma trận img với ma trận weight chuyển vị, do tính chất của phép nhân ma trận ta cần chuyển vị ma trận weight để phép toán thực hiện đúng như các công thức 4 5 6 đã nêu ở trên. Ta có ma trận đầu vào có kích thước $m \times n \times 3$ vậy khi nhân với vector trọng số có kích thước 3×1 thì ma trận kết quả sẽ có kích thước $m \times n \times 1$.
3. Sau đó sử dụng hàm np.clip của thư viện numpy để giới hạn lại các giá trị vượt quá khoảng giá trị [0, 255].
4. Trả về ma trận điểm ảnh sepia.

4.5 Làm mờ/sắc nét ảnh

Đối với hai chức năng làm mờ và làm sắc nét ảnh sử dụng chung hai hàm phụ trợ nên chúng sẽ được mô tả ở đây để tránh việc lặp lại gây dài dòng cho báo cáo. Hai hàm phụ trợ này bao gồm:

- **Hàm add_padding**

- **Mô tả hàm:** add_padding giúp ta thêm các viền xung quanh ma trận điểm ảnh gốc.
- **Tham số đầu vào:** Nhận hai đối số đó là img (ma trận điểm ảnh gốc) và kernel (kernel được sử dụng).
- **Kết quả đầu ra:** Ma trận điểm ảnh gốc với các viền xung quanh
- **Các bước thực hiện:**
 1. Lấy thông tin về số hàng của kernel với "kernel_height = kernel.shape[0]" và số hàng và số cột của ảnh với "img_height, img_width = img.shape[:2]".
 2. Tính toán viền xung quanh với "padding = kernel_height // 2" với toán tử "//" sẽ làm tròn kết quả xuống số nguyên gần nhất (ví dụ kernel_height = 3 thì số padding cần là 3 // 2 = 1).
 3. Khởi tạo một ma trận không có kích thước lớn hơn ma trận điểm ảnh gốc với số hàng bằng số hàng của ma trận điểm ảnh gốc cộng hai lần padding (một viền trên và một viền dưới), tương tự với cột ta cũng cộng hai lần padding (một viền phải và một viền trái) bằng lệnh "padded_img = np.zeros((img_height+2*padding, img_width+2*padding, 3))".
 4. Ta sao chép các phần tử trong điểm ảnh gốc vào ma trận không vừa tạo với điểm bắt đầu ở cả hàng và cột là padding và điểm kết thúc là số hàng và số cột của ma trận điểm ảnh gốc cộng với padding.
 5. Trả về ma trận điểm ảnh có các viền xung quanh.

- **Hàm perform_convolution**

- **Mô tả hàm:** perform_convolution giúp ta thực hiện phép tích chập trên hai ma trận.
- **Tham số đầu vào:** Nhận ba đối số đó là img (ma trận điểm ảnh gốc), kernel (kernel được sử dụng), và padded_img (ma trận điểm ảnh gốc có viền xung quanh).
- **Kết quả đầu ra:** Ma trận kết quả sau khi thực hiện phép tích chập.
- **Các bước thực hiện:**
 1. Lấy thông tin về số hàng và cột của kernel với "kernel_height, kernel_width = kernel.shape[:2]" và số hàng, số cột cùng với số kênh màu của ảnh với "img_height, img_width, channels = img.shape[:3]".

2. Tạo ra tuple gồm shape chứa các thông tin bao gồm (img_height, img_width, kernel_height, kernel_width và channels).
3. Tạo ra tuple strides bằng cách gọi thuộc tính strides [11] của padded_img chứa các thông tin (padded_img.strides[0], padded_img.strides[1], padded_img.strides[0], padded_img.strides[1], padded_img.strides[2]). Trong đó:
 - * padded_img.strides[0] là số byte cần để di chuyển đến hàng kế tiếp.
 - * padded_img.strides[1] là số byte cần để di chuyển đến cột kế tiếp.
 - * padded_img.strides[2] là số byte cần để di chuyển đến channel kế tiếp.

Ví dụ, một ma trận điểm ảnh có kích thước $512 \times 512 \times 3$ có kiểu dữ liệu float32, đối với hàng ta cần phải bỏ qua $512 * 3 * 32/8 = 6144$ bytes để đến hàng kế tiếp, đối với cột ta cần phải bỏ qua $3 * 4 = 12$ bytes và đối với channel ta cần bỏ qua 4 bytes.

4. Khởi tạo một sliding windows [12, 13] với hai tuple shape và strides đã được tạo từ trước với lệnh "sliding_windows = np.lib.stride_tricks.as_strided(padded_img, shape=shape, strides=strides)". Với lệnh này, một sliding window sẽ được áp lên ma trận padded_img để tạo ra các ma trận con phục vụ cho phép tích chập.

Ví dụ, một ma trận điểm ảnh có kích thước $512 \times 512 \times 3$ có kiểu dữ liệu float32 và kernel được chọn có size là 3 với hai tuple ở trên ta có shape = (514, 514, 3, 3, 3) (img_height và img_width đã thêm padding) và strides = (6168, 12, 6168, 12, 4). Nhìn vào shape ta có thể thấy:

- * Sliding windows có 514 window cho cả chiều cao và chiều dài của ảnh.
- * Mỗi window có kích thước là 3×3 .
- * Mỗi phần tử trong window là 3 kênh màu.

Để sliding window biết phải di chuyển thế nào ta sẽ dùng đến strides để đảm bảo sliding windows truy cập đúng bộ nhớ.

5. Sử dụng hàm np.tensordot cung cấp bởi thư viện numpy [14] để thực hiện phép nhân tensor với "result = np.tensordot(sliding_windows, kernel, axes=((2, 3), (0, 1)))", trong đó phép tích chập được thực hiện trên mỗi window của sliding_windows thông qua axes = (2,3) với kernel. Với mỗi channel, chúng sẽ được nhân tương ứng với phần tử trong kernel sau đó lấy tổng lại để được giá trị một channel của một điểm ảnh trong ma trận ảnh kết quả, lặp lại quá trình cho toàn bộ channel ta được một điểm ảnh hoàn chỉnh trong ma trận điểm ảnh kết quả. Tiếp tục quá trình này cho toàn bộ điểm ảnh trong padded_img ta được ma trận điểm ảnh kết quả hoàn thiện.
6. Trả về ma trận điểm ảnh kết quả.

4.5.1 Làm mờ ảnh

- **Ý tưởng thực hiện:**

Để làm mờ ảnh, ta cần tính toán lại toàn bộ điểm ảnh có trong ảnh gốc. Điều này có thể được thực hiện bằng cách phân bố lại màu sắc của một điểm ảnh bằng cách áp dụng một bộ lọc Gaussian kernel [15] lên ảnh.

Gaussian kernel là một ma trận vuông số học với các giá trị có được thông qua phân phối Gauss. Kích thước của ma trận cần phải là số lẻ vì với ma trận có kích thước lẻ ta sẽ luôn có vị trí trung tâm của ma trận. Các giá trị của bộ lọc này biểu thị cho trọng số màu của các điểm ảnh mà giá trị đó được đặt lên. Khi áp dụng Gaussian Kernel lên ảnh, mỗi điểm ảnh sẽ được thay thế bằng trung bình của các điểm ảnh xung quanh nó, với trọng số lớn nhất tại điểm trung tâm của ma trận. Các điểm ảnh càng xa điểm trung tâm, trọng số càng nhỏ.

Gaussian kernel có thể có nhiều giá trị và kích thước và kernel có kích thước càng lớn thì mức độ làm mờ càng cao nhưng đôi lại sẽ tốn nhiều thời gian và bộ nhớ hơn cho việc tính toán. Để tính các giá trị trọng số trong Gaussian kernel ta sử dụng hàm phân phối chuẩn Gauss [16]:

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

Trong đó, x và y là khoảng cách của vị trí đang xét đến vị trí trung tâm của ma trận và σ là độ lệch chuẩn trong phân phối Gauss.

Dựa vào tài liệu được cung cấp [15] ta có các ví dụ về Gaussian kernel cơ bản thường được sử dụng để làm mờ hoặc làm sắc nét ảnh ảnh:

$$\begin{aligned} Gaussian_Kernel_3x3 &= \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \\ Gaussian_Kernel_5x5 &= \frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} \end{aligned}$$

Để tính được giá trị màu cho các điểm ảnh khi làm mờ, ta dùng phép tích tích chập (convolution) cho hai ma trận điểm ảnh gốc và Gaussian kernel. Phép tích tích chập hay còn gọi là convolution [17] là phép toán phổ biến trong xử lý ảnh, cách thực hiện của phép tích tích chập được hiểu như sau:

- Với mỗi giá trị x_{ij} trong ma trận điểm ảnh gốc lấy ra một ma trận con sao cho kích thước của ma trận con đó bằng với kích thước của Gaussian kernel W và có x_{ij} là trung tâm, ma trận này được gọi là A.
- Thực hiện phép tích chập trên ma trận A và W sẽ thu được giá trị màu của điểm ảnh kết quả y_{ij} . Gán kết quả này vào ma trận kết quả Y, lặp lại quá trình này với mọi điểm ảnh x_{ij} sẽ thu được các điểm ảnh y_{ij} tương ứng.

Tuy nhiên, phép tính tích chập sẽ gặp vấn đề khi thực hiện trên các điểm ảnh x_{ij} nằm ở rìa khi ta không thêm tìm được các ma trận con có x_{ij} là trung tâm. Do đó, ta cần sử dụng các padding [18] để hỗ trợ việc này, padding là sẽ là các điểm ảnh có giá trị 0 được thêm vào ảnh gốc sao cho luôn đảm bảo sẽ tìm được ma trận con A để thực hiện phép tính tích chập.

Để hiểu rõ hơn về phép tích chập và thuật toán làm mờ ta thực hiện làm mờ trên một ma trận 2×2 với Gaussian kernel 3×3 .

Bước 1: Thêm các padding vào ma trận gốc

$$X = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \rightarrow X = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Bước 2: Xét phần tử x_{11} trong ma trận gốc, lấy ra ma trận con A sao cho x_{11} làm trung tâm ma trận A. Thực hiện tích chập giữa A và bộ lọc Gaussian Kernel W.

$$X \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow A \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 3 & 4 \end{bmatrix} \times W \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \frac{9}{8} \rightarrow Y \begin{bmatrix} \frac{9}{8} & \dots \\ \dots & \dots \end{bmatrix}$$

Bước 3: Lặp lại bước hai cho ba giá trị còn lại trong ma trận gốc

$$X \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow A \begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & 0 \\ 3 & 4 & 0 \end{bmatrix} \times W \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \frac{21}{16} \rightarrow Y \begin{bmatrix} \frac{9}{8} & \frac{21}{16} \\ \dots & \dots \end{bmatrix}$$

$$X \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow A \begin{bmatrix} 0 & 1 & 2 \\ 0 & 3 & 4 \\ 0 & 0 & 0 \end{bmatrix} \times W \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \frac{3}{2} \rightarrow Y \begin{bmatrix} \frac{9}{8} & \frac{21}{16} \\ \frac{3}{2} & \dots \end{bmatrix}$$

$$X \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \rightarrow A \begin{bmatrix} 1 & 2 & 0 \\ 3 & 4 & 0 \\ 0 & 0 & 0 \end{bmatrix} \times W \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \frac{7}{4} \rightarrow Y \begin{bmatrix} \frac{9}{8} & \frac{21}{16} \\ \frac{3}{2} & \frac{7}{4} \end{bmatrix}$$

Bước 4: Kết thúc và Y là ma trận kết quả đại diện cho ảnh gốc sau khi được làm mờ.

- Cài đặt:

- **Tên hàm:** apply_blur_filter
- **Tham số đầu vào:** Hàm nhận hai đối số đó là img (ma trận điểm ảnh gốc) và size (kích thước kernel).
- **Kết quả đầu ra:** Ma trận điểm ảnh được làm mờ.
- **Chi tiết thực hiện:**
 1. Dựa vào đối số size để khởi tạo kernel.
 2. Gọi hàm add_padding và truyền vào hai tham số là img và kernel với kết quả trả về là một ma trận có viền xung quanh với trung tâm là ma trận điểm ảnh gốc.
 3. Gọi hàm perform_convolution và truyền vào ba tham số là padded_img, img và kernel để thực hiện phép tích chập với kết quả trả về là ma trận điểm ảnh được làm mờ.
 4. Trả về ma trận điểm ảnh được làm mờ.

4.5.2 Làm sắc nét ảnh

- Ý tưởng thực hiện

Dộ sắc nét trong ảnh là một yếu tố quan trọng trong chất lượng hình ảnh, nó phản ánh mức độ rõ ràng và chi tiết của các chi tiết nhỏ trong ảnh. Một bức ảnh sắc nét sẽ có các đường viền rõ ràng, các chi tiết nhỏ được hiển thị một cách rõ ràng và không bị mờ hoặc nhòe.

Làm sắc nét ảnh (sharpening) là quá trình xử lý hình ảnh nhằm tăng cường các chi tiết và làm cho các đường viền trong ảnh trở nên rõ ràng hơn. Để làm được điều này ta cũng cần tính toán lại giá trị cho các điểm ảnh để cho ra một ảnh mới có độ sắc nét cáo hơn. Do đó, ta cần sử dụng một kernel khác so với Gaussian kernel đó là [19]:

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

Tương tự như làm mờ ảnh ta thực hiện phép tính tích chập với các bước đã được nêu ở trên cho hai ma trận điểm ảnh gốc và kernel với kết quả là ma trận điểm ảnh có các cạnh và chi tiết được làm sắc nét.

- Cài đặt:

- **Tên hàm:** apply_sharpen_filter
- **Tham số đầu vào:** Hàm nhận hai đối số đó là img (ma trận điểm ảnh gốc) và size (kích thước kernel).
- **Kết quả đầu ra:** Ma trận điểm ảnh được làm sắc nét.
- **Chi tiết thực hiện:**
 1. Dựa vào đối số size để khởi tạo kernel.
 2. Gọi hàm add_padding và truyền vào hai tham số là img và kernel với kết quả trả về là một ma trận có viền xung quanh với trung tâm là ma trận điểm ảnh gốc.
 3. Gọi hàm perform_convolution và truyền vào ba tham số là padded_img, img và kernel để thực hiện phép tích chập với kết quả trả về là ma trận điểm ảnh được làm sắc nét.
 4. Trả về ma trận điểm ảnh này.

4.6 Cắt ảnh theo kích thước (cắt ở trung tâm)

- Ý tưởng thực hiện:

Cắt ảnh là quá trình loại bỏ bớt các dòng, cột theo một cách nào đó để tạo ra một ảnh mới có kích thước bé hơn ảnh gốc. Trong đó án này, do ảnh được truyền vào là một ảnh vuông có kích thước $n \times n$ nên đối với chức năng cắt trung tâm ta sẽ xoá đi các dòng và cột ở ngoài rìa của ma trận điểm ảnh.

Để thực hiện việc cắt dòng và cột dựa trên các tham số đầu vào chúng ta sẽ sử dụng kỹ thuật slicing trong python để xác định được các điểm ảnh cần giữ lại [20].

- Cài đặt:

- **Tên hàm:** crop_img_center
- **Tham số đầu vào:** Hàm nhận hai đối số đó là img (ma trận điểm ảnh gốc) và percentage (Tỉ lệ phần trăm giữ lại của ảnh gốc $0 < \text{percentage} \leq 100$).
- **Kết quả đầu ra:** Ma trận điểm ảnh được cắt ở phần trung tâm với kích thước giảm percentage%.
- **Chi tiết thực hiện:**
 1. Kiểm tra giá trị phần trăm hợp lệ, nếu giá trị phần trăm không nằm trong khoảng $(0, 100]$, hàm sẽ in ra thông báo lỗi.
 2. Lấy thông tin về chiều cao và chiều rộng của ảnh thông qua "img_height, img_width = img.shape[:2]".

3. Tính toán kích thước của vùng cần cắt bằng cách lấy `img_height` và `img_width` nhân với `percentage / 100` sau đó ép kiểu về `int`.
4. Xác định các tọa độ bắt đầu của vùng cần cắt, bằng cách lấy kích thước của ảnh trừ cho kích thước của vùng cần cắt tất cả chia 2 với "`start_y = (img_height - crop_height) // 2`" và "`start_x = (img_width - crop_width) // 2`".
5. Xác định các tọa độ kết thúc của vùng cần cắt, bằng cách lấy kích thước của ảnh cộng cho kích thước của vùng cần cắt với "`end_y = start_y + crop_height`" và "`end_x = start_x + crop_width`".
6. Sử dụng các tọa độ đã tính toán để cắt vùng trung tâm của ảnh với "`cropped_img = img[start_y:end_y, start_x:end_x]`".
7. Trả về ma trận điểm ảnh đã cắt.

4.7 Cắt ảnh theo khung tròn

- **Ý tưởng thực hiện:**

Để cắt ảnh theo khung tròn ta cần tìm một phương trình hình tròn để áp lên ảnh và xác định các điểm ảnh nằm ngoài đường tròn và gán các điểm ảnh đó thành màu đen. Do đó, ta cần sử dụng đến phương trình đường tròn đã được học từ chương trình trung học phổ thông. Phương trình đường tròn có tâm tại (a, b) bán kính R :

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

Với R bằng một nửa chiều cao của ảnh để đảm bảo rằng đường tròn sẽ nằm trong ảnh mà không bị tràn ra ngoài với tâm (a, b) là điểm ảnh nằm ở vị trí trung tâm.

Để một điểm có tọa độ (i, j) thuộc đường tròn (C) điểm đó phải thoả điều kiện sau: $(i - a)^2 + (j - b)^2 \leq R^2$. Áp dụng điều này lên ảnh ta có, mỗi điểm ảnh có tọa độ là (i, j) nên để biết được điểm ảnh có nằm trong đường tròn hay không ta dùng điều kiện nêu trên để kiểm tra, nếu nằm trong đường tròn thì ta đến điểm ảnh tiếp theo nếu nằm ngoài đường tròn ta thay đổi màu sắc của điểm ảnh thành màu đen $(0, 0, 0)$.

- **Cài đặt:**

- **Tên hàm:** `crop_img_circle`
- **Tham số đầu vào:** Hàm nhận một đối số đó là `img` (ma trận điểm ảnh gốc).
- **Kết quả đầu ra:** Ma trận điểm ảnh được cắt theo khung tròn.
- **Chi tiết thực hiện:**

1. Sử dụng hàm `np.copy` của thư viện `numpy` để tạo ra một ma trận giống hệt với ma trận điểm ảnh đầu vào và gán vào biến `result`. Điều này sẽ giúp việc thay đổi trên ma trận `result` không làm ảnh hưởng đến ma trận ảnh gốc.

2. Lấy thông tin về số hàng của ảnh thông qua "size = result.shape[0]". Sau đó tính toán toạ độ trung tâm và bán kính của đường tròn với "center_index = radius = size//2". Trong đó, ta sử dụng toán tử "//" để kết quả của phép toán "size//2" làm tròn xuống số nguyên gần nhất.
3. Bằng cách sử dụng hàm np.tile được cung cấp bởi thư viện numpy [21] ta có lệnh "matrix_index_j = np.tile(np.arange(size), reps=(size, 1))" với np.arange(size) [22] sẽ tạo ra một list bắt đầu từ 0 đến size-1 và np.tile nhận đối số là list trả về của np.arange(size) và reps = (size, 1) có nghĩa là kết quả trả về của np.tile sẽ là một ma trận chứa toạ độ j của các điểm ảnh bắt đầu từ 0 đến size-1.
4. Ta tiếp tục tạo một ma trận chứa toạ độ i của các điểm ảnh bằng cách dùng hàm np.copy để sao chép ma trận toạ độ j đã tạo và chuyển vị ma trận đó.
5. Áp dụng điều kiện ở trên để kiểm tra xem khoảng cách từ điểm ảnh có toạ độ (i, j) có thuộc đường tròn hay không, ta dùng lệnh sau để thực hiện "masks = ((matrix_index_i-center_index)**2 + (matrix_index_j-center_index)**2) > (radius**2)", trong đó:
 - * matrix_index_i và matrix_index_j là hai ma trận toạ độ được miêu tả ở trên
 - * Phép toán $(matrix_index_i-center_index)**2 + (matrix_index_j-center_index)**2$ được dùng để tính khoảng cách từ điểm ảnh có toạ độ (i, j) đến tâm của đường tròn.
 - * Sau đó kết quả này được so sánh với bình phương bán kính " $(radius**2)$ ", Sau đó lưu kết quả của phép so sánh vào toạ độ (i, j) của ma trận masks.
6. Do masks là một ma trận 2 chiều nên ta có thể gọi "result[masks]" để truy cập vào các điểm ảnh, và nếu điểm ảnh đó mang giá trị true trong masks có nghĩa rằng điểm ảnh đó nằm ngoài đường tròn và đổi 3 kênh màu của điểm ảnh đó thành màu đen [0, 0, 0].

4.8 Cắt ảnh theo khung 2 elip chéo nhau

- **Ý tưởng thực hiện:**

Tương tự như cắt ảnh theo khung tròn, cắt ảnh theo khung elip cũng cần xác định phương trình đường tròn và tìm các điểm ảnh nằm ngoài hai elip chéo nhau, tuy nhiên không giống với hình tròn hình elip có phương trình phức tạp hơn nhiều so với đường tròn. Như được học ở trung học phổ thông, ta có phương trình ellip tổng quát là:

$$(E) : \frac{(x - m)^2}{a^2} + \frac{(y - n)^2}{b^2} = 1 \quad (7)$$

Với (m, n) là toạ độ tâm của ellip, a là bán trục lớn và b là bán trục nhỏ của ellip.

Tiếp đến ta cần xoay hai ellip này một góc α quanh tâm, để elip có các trục nằm trên các đường chéo của bức ảnh (hình vuông), ta cần quay elip một góc 45 độ hoặc 135 độ. Phương trình elip sau khi quay một góc α sẽ trở thành (tham khảo thêm [23] về chứng minh chi tiết):

$$(E) : \frac{((x - m)\cos(\alpha) - (y - n)\sin(\alpha))^2}{a^2} + \frac{((x - m)\sin(\alpha) + (y - n)\cos(\alpha))^2}{b^2} = 1 \quad (8)$$

Tuy nhiên điều khó khăn nhất đó chính là tìm ra được điều kiện để elip tiếp xúc với cạnh của hình vuông đây là bài toán "Ellipse cực đại nội tiếp trong một hình vuông" [24]. Để elip tiếp xúc với tất cả các cạnh của hình vuông, ta cần giải quyết bài toán toán học để tìm mối liên hệ giữa trục dài, trục ngắn của elip và kích thước của bức ảnh. Điều kiện là (tham khảo thêm [25] về chứng minh chi tiết):

$$a^2 + b^2 = \frac{\text{cạnh}^2}{2} \quad (9)$$

Khi đó, ta chọn các giá trị a và b sao cho thỏa mãn điều kiện này. Ví dụ, chọn:

$$a^2 = \frac{3}{4}\text{cạnh}^2 \quad \text{và} \quad b^2 = \frac{1}{4}\text{cạnh}^2$$

Cuối cùng từ công thức 8 và 9, ta có thể suy ra được phương trình Ellip tâm (m, n) nghiên góc α là:

$$(E) : \frac{((x - m)\cos(\alpha) - (y - n)\sin(\alpha))^2}{\frac{3}{4}\text{cạnh}^2} + \frac{((x - m)\sin(\alpha) + (y - n)\cos(\alpha))^2}{\frac{1}{4}\text{cạnh}^2} = 1 \quad (10)$$

Khi đã có được phương trình Elip 10, ta sẽ tìm được các điểm ảnh nào nằm bên trong, bên ngoài Elip. Với một điểm ảnh có tọa độ (i, j) thì điều kiện để điểm ảnh này nằm bên trong Elip là:

$$\frac{((i - m)\cos(\alpha) - (j - n)\sin(\alpha))^2}{\frac{3}{4}\text{cạnh}^2} + \frac{((i - m)\sin(\alpha) + (j - n)\cos(\alpha))^2}{\frac{1}{4}\text{cạnh}^2} <= 1 \quad (11)$$

Cuối cùng sử dụng các phép toán logic để xác định các điểm ảnh nằm ngoài cả hai elip và thay đổi giá trị màu của chúng thành màu đen.

- **Cài đặt:**

- **Tên hàm:** crop_img_ellipse
- **Tham số đầu vào:** Hàm nhận một đối số đó là img (ma trận điểm ảnh gốc).
- **Kết quả đầu ra:** Ma trận điểm ảnh được cắt theo 2 khung ellip chéo nhau.
- **Chi tiết thực hiện:**

1. Sử dụng hàm np.copy của thư viện numpy để tạo ra một ma trận giống hệt với ma trận điểm ảnh đầu vào và gán vào biến result. Điều này sẽ giúp việc thay đổi trên ma trận result không làm ảnh hưởng đến ma trận ảnh gốc.

2. Lấy thông tin về số hàng của ảnh thông qua "size = result.shape[0]". Sau đó tính toán toạ độ trung tâm và bán kính của đường tròn với "center_index = radius = size//2". Trong đó, ta sử dụng toán tử "//" để kết quả của phép toán "size//2" làm tròn xuống số nguyên gần nhất.
3. Tính hai giá trị a^2 và b^2 sử dụng công thức (9) với hai lệnh "a_pow_2 = 0.75*(size**2)/2.0" và "b_pow_2 = 0.25*(size**2)/2.0".
4. Bằng cách sử dụng "matrix_index_j = np.tile(np.arange(size), reps=(size, 1))" với np.arange(size) sẽ tạo ra một list bắt đầu từ 0 đến size-1 và np.tile nhận đối số là list trả về của np.arange(size) và reps = (size, 1) có nghĩa là kết quả trả về của np.tile sẽ là một ma trận chứa toạ độ j của các điểm ảnh bắt đầu từ 0 đến size-1.
5. Ta tiếp tục tạo một ma trận chứa toạ độ i của các điểm ảnh bằng cách dùng hàm np.copy để sao chép ma trận toạ độ j đã tạo và chuyển vị ma trận đó.
6. Gọi hàm find_outlier_pixels để xác định các điểm ảnh nằm ngoài elip nghiêng góc 135 độ ($3*\pi/4$) và góc 45 độ ($\pi/4$), hàm này có cấu trúc như sau:
 - * **Tên hàm:** find_outlier_pixels
 - * **Tham số đầu vào:** Hàm nhận vào 4 đối số bao gồm a_pow_2, b_pow_2 (các giá trị bình phương của trục lớn và trục nhỏ của elip), center_index (tọa độ tâm elip), alpha (Góc quay của elip) và matrix_i, matrix_j (ma trận lưu trữ tọa độ dòng và cột của tất cả các điểm ảnh).
 - * **Kết quả đầu ra:** Ma trận boolean, đánh dấu các điểm ảnh nằm ngoài elip (True nếu nằm ngoài, False nếu nằm trong).
 - * **Chi tiết thực hiện:**
 - (a) Cài đặt công thức 11 để tính giá trị của phương trình ellip với toạ độ (i, j) sau khi quay góc α .
 - (b) Xác định các điểm ảnh nằm ngoài elip bằng cách kiểm tra giá trị của phương trình ellip có lớn hơn 1 hay không " $ellipse > 1$ ".
 - (c) Trả về ma trận kết quả chứa các điểm ảnh nằm ngoài ellip.
7. Kết hợp hai mặt nạ (masks1 và masks2) để xác định các điểm ảnh nằm ngoài cả hai elip bằng cách sử dụng toán tử logic AND (np.logical_and) [26].
8. Thay đổi màu các điểm ảnh nằm ngoài cả hai elip thành màu đen [0, 0, 0].
9. Trả về ma trận điểm ảnh kết quả.

4.9 Phóng to/Thu nhỏ 2x

4.9.1 Phóng to ảnh 2x

- **Ý tưởng thực hiện:**

Đối với chức năng phóng to ảnh ta phải thực hiện việc thêm các điểm ảnh vào ma trận điểm ảnh gốc để tăng kích thước của ảnh. Do đó, chúng ta sẽ tạo ra một ma trận điểm ảnh mới bằng cách lặp lại các điểm ảnh có sẵn để thêm các hàng và cột mới với mỗi điểm ảnh sẽ được lặp lại 3 lần và tạo thành một vùng điểm ảnh có chung giá trị như sau:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \rightarrow \begin{bmatrix} a & a & b & b \\ a & a & b & b \\ c & c & d & d \\ c & c & d & d \end{bmatrix}$$

Với cách thực hiện này ta có thể tạo ra ma trận ảnh mới có thêm các hàng và cột mới để tăng kích thước ảnh và vẫn giữ nguyên được các chi tiết trong ảnh.

- **Cài đặt:**

- **Tên hàm:** zoom_in
- **Tham số đầu vào:** Hàm nhận một đối số đó là img (ma trận điểm ảnh gốc).
- **Kết quả đầu ra:** Ma trận điểm ảnh đã thu nhỏ.
- **Chi tiết thực hiện:**

1. Sử dụng hàm np.repeat cung cấp bởi thư viện numpy [27] để thực hiện việc thêm các hàng và cột và gán ma trận điểm ảnh kết quả vào biến kết quả với lệnh "zoomed_in_image = np.repeat(np.repeat(img, 2, axis=0), 2, axis=1)" Trong đó:
 - * "np.repeat(img, 2, axis=0)" sẽ thực hiện việc lặp lại các điểm ảnh trong ma trận theo hàng do axis=0, và số lần lặp lại là 2 do ta đang cần phóng to ảnh x2.
 - * np.repeat(np.repeat(img, 2, axis=0), 2, axis=1)" sẽ thực hiện việc lặp lại các điểm ảnh trong ma trận theo cột do axis=1, với cùng số lần lặp là 2 và ma trận được dùng để lặp là ma trận kết quả của lệnh "np.repeat(img, 2, axis=0)".
2. Trả về ma trận điểm ảnh này.

4.9.2 Thu nhỏ ảnh 2x

- **Ý tưởng thực hiện:**

Đối với thu nhỏ ảnh để thực hiện được chức năng này chúng ta cần phải loại bỏ bớt các điểm ảnh để thu nhỏ kích thước nhưng vẫn phải giữ lại được các đặc trưng của ảnh. Do đó, chúng ta sẽ tạo ra một ma trận điểm ảnh mới bằng cách lấy các hàng và cột của ma trận điểm ảnh gốc nhưng bỏ qua các hàng và cột có số lẻ (nghĩa là ta chỉ lấy các hàng và cột có số lẻ 0, 2, 4, 6,...) như sau:

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \rightarrow \begin{bmatrix} a & c \\ i & k \end{bmatrix}$$

Với cách thực hiện này ta vừa có thể giảm bớt các điểm ảnh bằng cách bỏ qua các hàng và cột lẻ nhưng vẫn giữ được cấu trúc của ảnh mặc dù một số chi tiết có thể bị mất trong quá trình giảm kích thước.

- **Cài đặt:**

- **Tên hàm:** zoom_out
- **Tham số đầu vào:** Hàm nhận một đối số đó là img (ma trận điểm ảnh gốc).
- **Kết quả đầu ra:** Ma trận điểm ảnh đã thu nhỏ.
- **Chi tiết thực hiện:**

1. Sử dụng kỹ thuật slicing trong python để lấy ra các điểm ảnh có nằm trên các hàng và cột chẵn và gán vào ma trận điểm ảnh kết quả với "zoomed_out_image = img[::-2, ::2]" với "::2" tương tự với chức năng flip ta bắt đầu từ 0 đến size - 1 với số bước là 2 để đảm bảo giá trị trả về luôn là số chẵn.
2. Trả về ma trận điểm ảnh này.

5 Mức độ hoàn thành

STT	Chức năng/Hàm	Mức độ hoàn thành	Ảnh kết quả
1	Thay đổi độ sáng	100%	
2	Thay đổi độ tương phản	100%	
3.1	Lật ảnh ngang	100%	
3.2	Lật ảnh dọc	100%	
4.1	RGB thành ảnh xám	100%	
4.2	RGB thành ảnh sepia	100%	

Bảng 1: Bảng đánh giá mức độ hoàn thành và hình ảnh kết quả cho từng chức năng

STT	Chức năng/Hàm	Mức độ hoàn thành	Ảnh kết quả
5.1	Làm mờ ảnh (size = 3)	100%	
5.2	Làm sắc nét ảnh	100%	
6	Cắt ảnh theo kích thước	100%	
7.1	Cắt ảnh theo khung tròn	100%	
7.2	Cắt ảnh theo khung elip	100%	
8	Hàm main	100%	
9.1	Phóng to 2x	100%	
9.2	Thu nhỏ 2x	100%	

Bảng 2: Bảng đánh giá mức độ hoàn thành và hình ảnh kết quả cho từng chức năng

Tài liệu

- [1] Pillow. Image module — pillow (pil fork) 6.2.1 documentation. <https://pillow.readthedocs.io/en/stable/reference/Image.html>. Accessed on 20/7/2024.
- [2] NumPy. Typing (numpy.typing) — numpy v2.0 manual. <https://numpy.org/doc/stable/reference/typing.html>. Accessed on 20/7/2024.
- [3] matplotlib. matplotlib.pyplot.show — matplotlib 3.5.1 documentation. https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.show.html. Accessed on 20/7/2024.
- [4] NumPy. numpy.ndarray.astype — numpy v2.0 manual. <https://numpy.org/doc/stable/reference/generated/numpy.ndarray.astype.html>. Accessed on 20/7/2024.
- [5] NumPy. numpy.stack — numpy v1.24 manual. <https://numpy.org/doc/stable/reference/generated/numpy.stack.html>. Accessed on 20/7/2024.
- [6] Dreamland Fantasy Studios. Image processing algorithms part 5: Contrast adjustment. <https://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-5-contrast-adjustment/>, 2010. Accessed on 21/7/2024.
- [7] NumPy. Indexing on ndarrays — numpy v1.22 manual. <https://numpy.org/doc/stable/user/basics.indexing.html>. Accessed on 21/7/2024.
- [8] Dreamland Fantasy Studios. Image processing algorithms part 3: Greyscale conversion,” dreamland fantasy studios. <https://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-3-greyscale-conversion/>, 2010. Accessed on 21/7/2024.
- [9] NumPy. numpy.matmul — numpy v1.22 manual. <https://numpy.org/doc/stable/reference/generated/numpy.matmul.html>. Accessed on 21/7/2024.
- [10] Stack Overflow. How is a sepia tone created? <https://stackoverflow.com/questions/1061093/how-is-a-sepia-tone-created>. Accessed on 22/7/2024.
- [11] NumPy. numpy.ndarray.strides — numpy v2.0 manual. <https://numpy.org/doc/stable/reference/generated/numpy.ndarray.strides.html>. Accessed on 24/7/2024.
- [12] NumPy. numpy.lib.stride_tricks.as_strided — numpy v1.23 manual. https://numpy.org/doc/stable/reference/generated/numpy.lib.stride_tricks.as_strided.html. Accessed on 24/7/2024.
- [13] Cyrille Rossant. Ipython cookbook - 4.6. using stride tricks with numpy. <https://ipython-books.github.io/46-using-stride-tricks-with-numpy/>. Accessed on 24/7/2024.
- [14] NumPy. numpy.tensordot — numpy v1.24 manual. <https://numpy.org/doc/stable/reference/generated/numpy.tensordot.html>. Accessed on 24/7/2024.
- [15] Wikipedia Contributors. Kernel (image processing). [https://en.wikipedia.org/wiki/Kernel_\(image_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing)). Accessed on 24/7/2024.

- [16] www.sciencedirect.com. Gaussian distribution - an overview | sciencedirect topics. <https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/gaussian-distribution>. Accessed on 24/7/2024.
- [17] R. Krishna. Coding gaussian blur operation in python from scratch. <https://medium.com/@rohit-krishna/coding-gaussian-blur-operation-from-scratch-in-python-f5a9af0a0c0f>. Accessed on 24/7/2024.
- [18] P. Vemulapati. Image padding techniques: Zero padding (part 1). https://medium.com/@Orca_Thunder/image-padding-techniques-zero-padding-part-1-669de127ba59. Accessed on 24/7/2024.
- [19] V. Powell. Image kernels explained visually. <https://setosa.io/ev/image-kernels/>. Accessed on 24/7/2024.
- [20] Stack Overflow. crop center portion of a numpy image. <https://stackoverflow.com/questions/39382412/crop-center-portion-of-a-numpy-image>. Accessed on 25/7/2024.
- [21] NumPy. numpy.tile — numpy v2.0 manual. <https://numpy.org/doc/stable/reference/generated/numpy.tile.html>. Accessed on 25/7/2024.
- [22] NumPy. numpy.arange — numpy v1.21 manual. <https://numpy.org/doc/stable/reference/generated/numpy.arange.html>. Accessed on 25/7/2024.
- [23] Pen and Paper Science. Rotation matrix derivation (step-by-step prove). <https://www.youtube.com/watch?v=EZufiIwwqFA>. Accessed on 26/7/2024.
- [24] users.math.uoc.gr. Maximalellipse. <http://users.math.uoc.gr/~pamfilos/eGallery/problems/MaximalEllipse.html>. Accessed on 26/7/2024.
- [25] K. Nhan. khanhnhan1512/image-processing. <https://github.com/khanhnhan1512/Image-Processing>. Accessed on 26/7/2024.
- [26] NumPy. numpy.logical_and — numpy v2.0 manual. https://numpy.org/doc/stable/reference/generated/numpy.logical_and.html. Accessed on 26/7/2024.
- [27] NumPy. numpy.repeat — numpy v1.24 manual. <https://numpy.org/doc/stable/reference/generated/numpy.repeat.html>. Accessed on 27/7/2024.